Høgskolen i Østfold

MASTER'S THESIS

Automatic Speech Recognition Within Air Traffic Control Domain

Lars Formoe and Dan Bruun Mygind

June 15th, 2023

Applied Computer Science
Faculty of Computer Science, Engineering and Economics

Abstract

This thesis investigates the complex issue of automatic speech recognition (ASR) in specialized contexts, focusing predominantly on air traffic control (ATC) communications involving various accents, noisy environments, and the challenges presented by limited training data. Acknowledging the considerable advancements made by state-of-the-art frameworks such as wav2vec 2.0 when fine-tuned on niche corpora, we also underscore the enduring problem of minimal or absent transcribed data in specific areas such as ATC.

Our research applies the wav2vec-U 2.0 model, an advanced unsupervised ASR framework capable of learning from raw audio and unpaired text. This reduces the need for time-consuming and costly transcriptions. Additionally, we conduct experiments using self-supervised pre-trained wav2vec 2.0 models, weak supervision with the pre-trained Whisper model, and supervised models based on Deep Speech 2 and Transformer networks. Each model has been meticulously trained on the ATC corpora ATCOSIM and Hiwire to scrutinize the models' effectiveness in handling domain-specific ASR. Our findings reveal that the fine-tuning of the pre-trained wav2vec 2.0 model results in a downstream model exhibiting exceptional accuracy on ATCOSIM, with a Word Error Rate (WER) of just below 1%. This strongly indicates wav2vec 2.0's capabilities in dealing with the challenges posed by low-resource domains like ATC communications.

Furthermore, our application of wav2vec-U 2.0 on ATCOSIM shows promising results utilizing a generative approach to ASR. In the broader context, we reflect upon the implications of these results for the wider research community within ASR and propose potential directions for building an ATC base model for continuous ASR development using real-life ATC communications data.

Keywords: automatic speech recognition · wav2vec 2.0 · wav2vec unsupervised · unsupervised learning · generative adversarial network · air traffic control communications · downstream · transfer learning

Acknowledgements

First and foremost, we would like to express our deepest gratitude to Kongsberg Defence & Aerospace (KDA) for the opportunity to explore automatic speech recognition as part of their remote towers systems (RTS).

We sincerely thank Espen Løkke, our primary contact from KDA, and our dedicated academic adviser, Dr. Hasan Ogul. Their contribution to fostering an inclusive and inspiring platform for open dialogue and exchanging ideas, concepts, and theories has been invaluable. This dynamic forum has been instrumental in shaping our work throughout the project.

While KDA has generously provided the project guidelines and supervision, it is important to note that they were not involved in the direct execution of this research. As such, KDA cannot be held accountable for any potential inaccuracies or oversights that may have been committed by the authors.

Our appreciation also goes to the Department of Computer Science and Communication at Østfold University College. Their provision of crucial hardware resources enabled the experiments integral to our research.

We are grateful to our families for their constant belief in us and the sacrifices they have made to support our academic pursuits. This achievement is as much yours as it is ours.

Thank you.

Contents

List of Figures

List of Tables

List of Listings

Chapter 1

Introduction

This study is a cooperative endeavor between Kongsberg Defence & Aerospace (KDA) and Østfold university college (HiØ), investigating the potential of incorporating automatic speech recognition (ASR) technology within applications tailored to support air traffic controllers operating remote towers systems (RTS) developed by KDA. The primary goal is to ascertain the feasibility and benefits of integrating state-of-the-art ASR techniques underpinned by artificial intelligence (AI) into specific applications designed for remote air traffic control (ATC) contexts.

ASR technology transforms audio input into textual output, requiring several conversions from analog sound waves to digital representations that can be associated with sequential characters, yielding coherent text. This study focuses on recent advancements in ASR systems that utilize deep neural networks, which have shown substantial progress in diminishing dependence on vast quantities of transcribed data by leveraging end-to-end architectures.

The research thoroughly reviews the current literature on ASR technology and delves into the complexities of deep neural network-based ASR systems. In addition, it evaluates various end-to-end models that have significantly improved ASR performance while reducing the need for extensively labeled corpora. Through analyzing these advancements, the study aims to lay a foundation for integrating ASR into applications crafted for remote ATC, offering insights into potential benefits and challenges associated with such systems.

In summary, this research contributes to the ongoing dialogue on optimizing air traffic management by assessing the prospects of integrating cutting-edge ASR technologies into applications for remote ATCs. By examining the latest developments within the field and their practical implications, this study can potentially guide future research and development efforts toward creating more efficient ATC operations.

1.1 Motivation

RTS represent an integrated approach to the remote orchestration of ATC duties, employing sophisticated electro-optical sensor technology as an integral system component. This technology essentially extends the most progressive sensor technology in military applications globally.

The Electro-Optical Sensor Suite is designed as a rotating platform housing a visual and infrared camera capable of 360-degree imaging (Figure 1.1.) It further accommodates a pan-tilt platform, which includes a visual zoom camera, a fixed lens infrared camera, a laser range finder, and a signal lamp. This intricate combination of advanced technologies enables the comprehensive

capture and transmission of critical sensory data[1].

Figure 1.1: Image showing the camera tower, installed at Berlevåg lufthavn.

Currently, the RTS is recognized as the most extensive remote tower operation worldwide, with deployment across 11 airports in Norway. Additional locations are being considered for future deployment, indicating the system's expansion potential. From the control center in Bodø, Norway (Figure 1.2), simultaneous operation of multiple airports can be done from a single workstation.

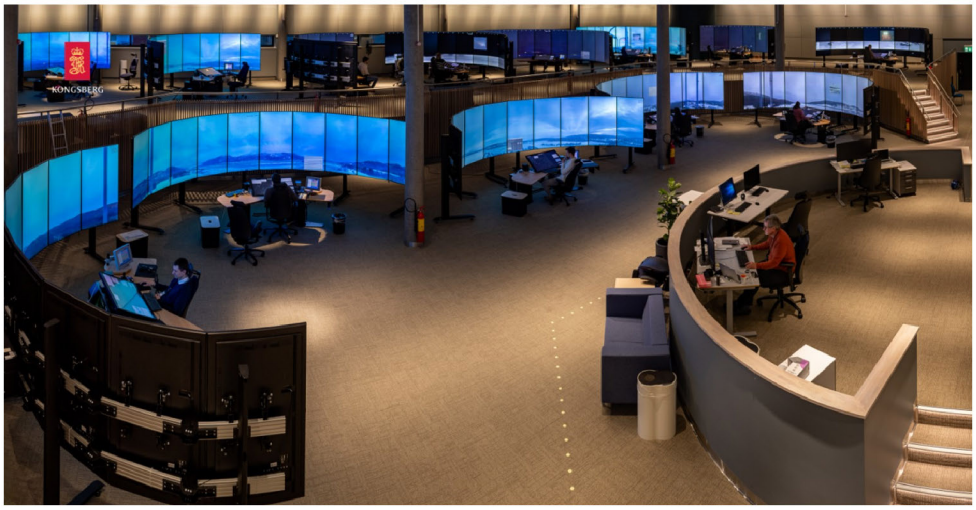


Figure 1.2: From the central control station in Bodø.

This workstation has ten high-resolution screens displaying live image feeds from various remote locations. Furthermore, a large screen directly in front of the controller at the workstation provides critical information in real-time (Figure 1.3).

[1] https://www.kongsberg.com/kda/what-we-do/defence-and-security/aviation-security/remote-and-digital-towers

Figure 1.3: Controller workstation.

This setup facilitates efficient monitoring and control of multiple locations simultaneously, effectively advancing the reach and effectiveness of air traffic management practices. This innovative approach to ATC exemplifies technological advancements in the field and underscores the potential for further enhancements in operational efficiency and safety.

KDA is interested in further enhancing the functionality of RTS by investigating the prospective benefits of real-time analysis and processing of audio communications between pilots and air traffic controllers. The proposed enhancement involves the generation of textual transcripts from these communications, which could be augmented with visual cues for controllers upon detection of anomalies. This strategic integration of ASR and natural language processing (NLP) techniques aims to expand the capabilities of remote ATC systems, thereby improving their operational efficiency.

The provision of real-time text transcripts of audio communications seeks to reduce the cognitive load on controllers, potentially contributing to overall safety improvements. Given that aircraft callsigns remain constant, identifying an aircraft through ASR can assist controllers in recognizing the current aircraft in communication. This can be facilitated by visually highlighting the aircraft on the controller's screen or adding additional graphical cues to the transcript, such as color codes.

The primary objective of these proposed enhancements is to aid air traffic controllers by providing confirmation tools designed to augment situational awareness. This would liberate controllers' cognitive resources, enabling them to manage two or more airports simultaneously without compromising safety levels. In addition, integrating these advanced ASR and NLP techniques with existing systems can transform the landscape of ATC operations, making them more efficient, safe, and capable of handling increased air traffic volumes.

1.2 Research questions

First, in order to suggest a suitable technology for KDA we need to get an understanding of the current research being done in the field of ASR specific to ATC. In addition to this, we also need to conduct our own experiments to get a measure of how well the different methodologies perform on ATC domain audio:

RQ 1 What are the current State-of-the-Art technologies for ATC ASR, and how can we effec-

tively implement these?

Second, an important consideration in addressing the challenges associated with working with ATC audio collected from airports in Norway is the stringent requirement for data security, given the sensitive and confidential nature of the data being handled. Detailed specifications of these restrictions and regulations cannot be disclosed in this document due to their confidentiality. However, it is unequivocal that any leakage or mishandling of this sensitive data could have severe implications. Therefore, proposed solutions to the ASR challenge must adhere to strong safety guidelines. Notably, the training data must not be exposed, uploaded, or subjected to third-party analysis under any circumstances. This non-negotiable requirement safeguards against potential data breaches, underscoring the primacy of data security in the development and implementation of these solutions

RQ 2 Which solutions meet the requirements for implementing ASR in KDA RTS?

Chapter 2

Background

This chapter introduces ASR and its significance in modern society. We discuss popular voice assistants employed by prominent tech giants such as Apple, Amazon, Microsoft, and Google to establish familiarity. Subsequently, we offer a concise historical overview of ASR development, tracing its origins from early integrated circuit-based designs in the 1950s to the current revolution facilitated by the combination of deep neural network (DNN) and hardware advancements capable of handling the extensive computational demands required for state-of-the-art ASR systems.

Moreover, we look at some of the underlying technologies that drive modern ASR advancements. This includes an examination of key breakthroughs that have facilitated the shift from labor-intensive, meticulously annotated, and synchronized corpora employed in early ASR development to the present-day emergence of end-to-end models that can generate coherent text from audio files without relying on transcribed data.

2.1 Voice assistants

Voice-activated digital assistants, including Apple's Siri[1], Amazon's Alexa[2], Microsoft's Cortana[3], and Google Assistant[4], share several commonalities in their architecture, functionality, and integration. These digital assistants primarily rely on advanced automatic speech recognition (ASR) systems, natural language processing (NLP), and machine learning (ML) algorithms for processing and interpreting spoken commands issued by users. These algorithms enable the understanding of complex linguistic structures and facilitate the conversion of speech signals into text or actionable commands.

A central component of these digital assistants is their ability to perform various core functions, such as answering questions, setting reminders, providing directions, sending messages, and controlling smart home devices. In addition to their core ASR, NLP, and ML components, these digital assistants most likely also employ various optimization techniques and algorithms to enhance their performance.

Moreover, these voice assistants leverage cloud-based computing infrastructures to process and store user data, allowing for continuous learning and improving recognition capabilities. This distributed architecture enables rapidly scaling computational resources, ensuring that the digital assistants remain highly responsive and adaptive to changing user demands and requirements.

[1] `https://developer.apple.com/documentation/sirikit`
[2] `https://developer.amazon.com/en-US/alexa`
[3] `https://github.com/microsoft/cortana-skills-samples`
[4] `https://developers.google.com/assistant`

The recent advancements in ASR technology culminate with persistent research efforts across multiple decades, tracing back to the 1950s. A comprehensive understanding of present-day ASR challenges necessitates an examination of the field's historical context and pivotal developments. Since the inception of early systems, numerous influential breakthroughs have been made, shaping the trajectory of ASR research and contributing to the present state of the technology. By appreciating the evolution of ASR, one can better contextualize current challenges, elucidate the underlying factors, and inform future research directions. Furthermore, this historical lens offers critical insights into the ongoing pursuit of enhanced ASR performance, resilience, and applicability across diverse scenarios, ultimately facilitating the design and implementation of robust, versatile, and efficient systems that cater to the needs of contemporary users and applications.

2.2 Early development

The evolution of Automatic Speech Recognition (ASR) has seen numerous significant advancements since its inception. Beginning with early systems limited by computational resources, the focus was predominantly on recognizing isolated words or digits. Researchers laid the foundation for more advanced feature extraction techniques as technology progressed, subsequently enhancing ASR systems' performance.

2.2.1 Audrey

Bell Labs' Automatic Digit Recognizer was an early speech recognition system developed in the 1950s [23]. Audrey was designed to recognize isolated spoken digits from zero to nine using analog electronic circuits and a unique feature extraction approach. Researchers at Bell Telephone Laboratories aimed to develop a system capable of understanding spoken numbers to facilitate automated telephone dialing. To achieve this, Audrey analyzed the sound patterns of spoken digits and extracted distinctive features. These were matched to predefined templates to recognize the spoken digits accurately.

Despite its limitations, it laid the groundwork for further research and development in the field, ultimately leading to more advanced systems like the Hidden Markov Model-based speech recognition systems in the 1970s and 1980s and, more recently, the deep learning-based models.

2.2.2 IBM Shoebox 1962

Developed in the early 1960s by a team of researchers at IBM's Thomas J. Watson Research Center, the Shoebox was an electromechanical system that employed a microphone to receive spoken input from the user. The system was designed to recognize 16 spoken words, including the digits from zero to nine, and basic arithmetic operations like addition, subtraction, multiplication, and division. Upon receiving the speech input, the Shoebox used a set of relay-based circuits to process the voice signals, extract relevant features, and match them to predefined patterns [3].

2.2.3 Linear predictive coding (LPC)

A widely used technique in speech processing for modeling the vocal tract and efficiently representing speech signals. Based on the source-filter model of speech production, LPC aims to capture the spectral envelope of the speech signal by approximating the vocal tract transfer function. This is achieved by estimating a set of linear prediction coefficients, which characterize the speech signal's formant structure and fundamental frequency. LPC gained significant attention in the 1970s through the pioneering work of Bishnu S. Atal [6] and Fumitada Itakura[43]. LPC

research laid the groundwork for numerous advancements in speech processing, particularly in speech coding and compression, and speech recognition technology.

2.2.4 Dynamic time warping (DTW)

DTW is a robust algorithm for measuring the similarity between two-time series, allowing for non-linear alignment between sequences [79]. DTW computes an optimal alignment between two sequences by minimizing the cumulative distance between corresponding points, subject to certain constraints. In early ASR systems, DTW was used for isolated word recognition, where the objective was to identify individual spoken words from a predefined vocabulary [61].

It has been extensively applied in various domains, including Automatic Speech Recognition (ASR), to account for temporal variations in speech signals caused by differences in speaking rate and style among speakers. Template-based ASR systems were created using DTW combined with feature extraction techniques, such as Linear Predictive Coding (LPC) and Mel-Frequency Cepstral Coefficients (MFCCs). However, these systems faced limitations in scaling to large vocabulary sizes and handling continuous speech recognition tasks.

2.2.5 Hidden Markov model

HMMs are a class of statistical models that have played a significant role in developing various applications in speech recognition, bioinformatics, and natural language processing [46]. HMMs are primarily used for modeling sequential data by employing an underlying stochastic process that is not directly observable, hence the term "hidden" [27].

HMMs were initially introduced in the 1960s [12], and since then, they have gained widespread popularity as powerful tools for modeling complex sequential data. Their success can be attributed to the development of the Expectation-Maximization (EM) algorithm and the forward-backward algorithm, which are utilized for model training and inference.

In the past, HMMs were employed as a key component in various applications, such as speech recognition and part-of-speech tagging [20]. However, with the advent of deep learning and the development of more powerful and flexible models like deep neural networks (DNNs), the prominence of HMMs has diminished in recent years.

2.2.6 ARPA Speech Understanding Research program

The program was a groundbreaking initiative launched in the early 1970s to advance the state of speech recognition and natural language understanding technology. As a significant milestone in the history of artificial intelligence, the program aimed to develop systems capable of understanding continuous speech, a significant leap forward from previous systems that recognized only isolated words or digits. Under the program, several speech recognition systems were developed, such as CMU's Harpy system, BBN's Hearsay-II, SDC's HWIM, and SRI's Speech Understanding System (SUS) [76]. Of these, the Harpy system could identify over 90% of a set of sentences from a 1000-word lexicon [49].

2.2.7 Gaussian mixture model (GMM)

GMMs are a probabilistic modeling technique used to represent the distribution of data points in a multidimensional space. They have been widely used for acoustic modeling, mapping acoustic features extracted from speech signals to phonetic units, such as phonemes [25].

A Gaussian Mixture Model combines multiple Gaussian distributions, each representing a specific cluster or component in the data. The overall model is a weighted sum of these individual

Gaussian components, with each element having its mean, covariance matrix, and a weight that reflects the proportion of data points belonging to that component.

GMMs have been used in ASR since the late 1980s to model speech feature probability distributions, such as MFCCs and LPC coefficients, for different speakers, accents, and styles [78]. GMMs were commonly combined with HMMs to create complete acoustic models in traditional ASR systems. HMMs represent the time-varying nature of speech signals and model the phonetic units in spoken language, while GMMs model the acoustic feature probability distribution given a specific state [74].

2.2.8 DragonDictate

Developed by Dragon Systems in the early 1990s, it was a pioneering Automatic Speech Recognition (ASR) software designed for personal computers. The system enabled users to dictate text and issue voice commands to their computers, making ASR technology more accessible to a broader audience.

At its core, DragonDictate relied upon Hidden Markov models for acoustic modeling and statistical language models to capture the structure and context of spoken language. The combination of these models enabled the system to recognize and transcribe continuous speech accurately, albeit with some constraints, such as a limited vocabulary and the necessity for users to pause momentarily between words (a technique referred to as discrete speech) [11].

Despite limitations, it was a success and paved the way for more advanced ASR systems, including Dragon NaturallySpeaking, which was released by Dragon Systems in 1997. Dragon NaturallySpeaking improved upon DragonDictate by allowing users to speak in a more natural, continuous manner without the need for pauses between words.

2.3 Increasing vocabularies and continuous speech

The adoption of HMMs, GMMs, n-gram language models, and improvements in feature extraction methods marked the ASR developments from the 1980s to 2000. In the 1990s, ASR research focused on large vocabulary continuous speech recognition (LVCSR) tasks, which involved recognizing continuous speech with thousands or tens of thousands of words. The development of the DARPA Wall Street Journal (WSJ) task [70] and the Switchboard corpus [32] provided new benchmarks for LVCSR, leading to new techniques, such as decision trees for context clustering [65].

During this period, n-gram language models also became widespread in ASR, providing a simple yet effective way to model the statistical properties of word sequences in natural language and applying Mel-frequency cepstral coefficients (MFCC) for feature extraction, which improved speech signal representation for ASR systems.

2.3.1 Mel-frequency cepstral coefficients (MFCC)

MFCCs have long been, and continue to be, a prominent feature extraction technique in Automatic Speech Recognition (ASR) and other speech processing tasks. In the early 1980s, MFCCs were introduced as an effective method for capturing perceptually relevant characteristics of human speech signals [24]. This development was rooted in psychoacoustic research, particularly the concept of the Mel scale [84]. The Mel scale represents the human perception of pitch changes by assigning frequencies spaced approximately logarithmically, thus simulating how the human cochlea responds to different frequencies [67].

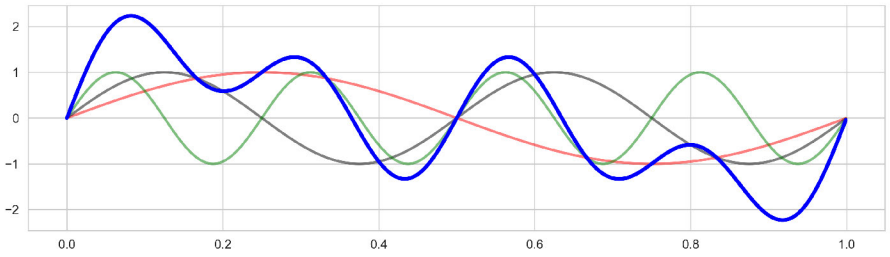
Extracting the MFCCs from an audio signal is typically broken down into the following steps:

1. Frame the signal: The audio signal is divided into short, overlapping frames, typically using a Hamming window, to reduce spectral leakage and maintain continuity between adjacent frames [83]. The frame size is typically between 20-30 ms, with an overlap of 50-75%.

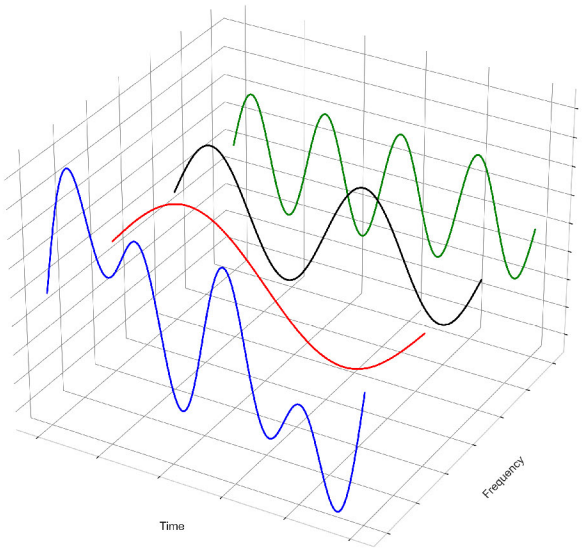
2. Compute the power spectrum: the Fast Fourier Transform (FFT) (Equation 2.1) is applied to each frame, converting it from the time domain to the frequency domain, as illustrated in Figure 2.1. The power spectrum is obtained by taking the squared magnitude of the FFT.

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-i2\pi kn/N} \qquad k = 0, 1, 2, ... \tag{2.1}$$

   where x_n is the time-domain input signal, N is the length of the input signal, X_k is the frequency-domain output signal, and i is the imaginary unit.



(a) Time domain.



(b) Frequency domain.

Figure 2.1: Fast Fourier Transform illustration. Here the blue signal is the sum of the red, black, and green signals. Applying the short-time Fourier transform, the blue signal is decomposed into the individual signals that make up its composition.

3. Mel filterbank processing: The power spectrum is then passed through a set of triangular filters, called the Mel filterbank, spaced according to the Mel scale as seen in Equation 2.2, where f is the frequency of the sound wave in hertz (Hz). The number of filters in the filter bank is usually between 20 and 40. Each filter's output is the sum of the power spectral components within its triangular window.

$$Mel(f) = 2595 \cdot \log_{10}(1 + \frac{f}{700}) \tag{2.2}$$

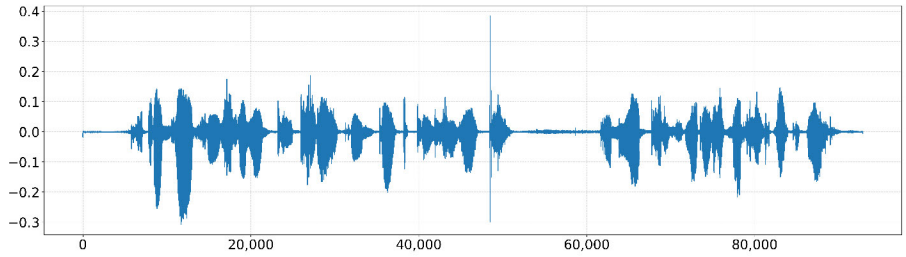
4. Logarithmic compression: Calculate the logarithm of the energy in each Mel-scaled filter output. Logarithmic compression is applied to reduce the dynamic range and model the logarithmic perception of loudness in the human auditory system.
5. Discrete Cosine Transform (DCT): The DCT is performed on the log-filterbank energies as seen in Equation 2.3, where x_n are the input values, X_k are the transformed values, and N is the total number of input values. This step decorrelates the filterbank coefficients and results in a more compact representation, typically retaining the first 12-13 coefficients.

The DCT coefficients are the MFCCs.

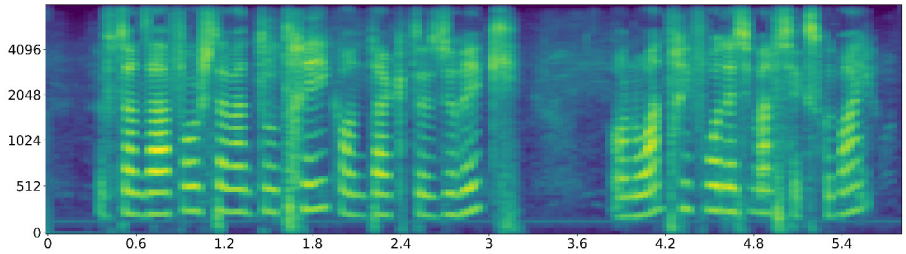
$$X_k = \sum_{n=0}^{N-1} x_n \cos\left[\frac{\pi}{N}\left(n + \frac{1}{2}\right)k\right], \quad k = 0, 1, \ldots, N-1 \tag{2.3}$$

6. Delta and Delta-Delta coefficients: In some applications, the first and second derivatives of MFCCs, called delta and delta-delta coefficients, are also computed and appended to the feature vector. This provides information about the dynamics and acceleration of the spectral shape, which can improve the performance of ASR systems.

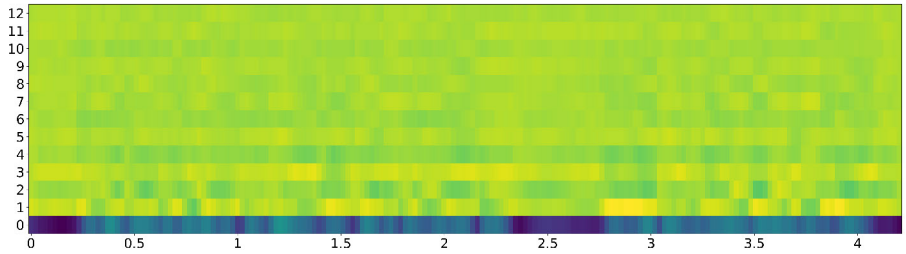
The result of this process can be seen in Figure 2.2. The first cepstral coefficient (index 0) represents the average power in the signal frame. In some applications, it is often discarded or replaced with other features (e.g., energy, pitch), as it does not provide significant information about the spectral shape. The higher-order cepstral coefficients capture various spectral details, with lower-index coefficients representing broader spectral features, while higher-index coefficients correspond to finer spectral details. In most ASR applications, the first 12 or 13 MFCCs (index 1 to 12 or 13) are used as they capture the most relevant information for speech recognition.



(a) Waveplot of the raw audio signal.



(b) Mel-scale spectrogram. The y-axis shows the frequency range in Hz, the x-axis shows time.



(c) MFCCs with index position on the y-axis.

Figure 2.2: The MFCC process.

In the 1990s, MFCCs emerged as the state-of-the-art feature extraction technique for HMM-based ASR systems, surpassing previous techniques such as LPC. MFCCs have been successful in the ASR field because they are capable of representing the spectral envelope of speech signals. This representation is crucial as it encapsulates speaker-specific information and phonetic content, vital for accurate speech recognition.

2.3.2 N-gram models

N-gram models are a class of probabilistic models that estimate the likelihood of a sequence of words by considering the conditional probability of the next word given the preceding N-1 words [47]. By approximating the probability of a word sequence while considering the limited context of N-1 prior words, N-gram models reduce the complexity of language models while still capturing critical contextual dependencies [57]. For instance, bigram models (N=2) compute the probability of a word based on the preceding word, while trigram models (N=3) condition the probability of a word on the two preceding words, and so on.

The primary purpose of utilizing N-gram models in ASR is to leverage contextual information about word sequences to resolve ambiguities and refine the output generated by the acoustic model. By incorporating language models into the decoding process, ASR systems can enhance recognition accuracy by selecting word sequences with higher likelihoods in the target language,

thereby mitigating the effects of recognition errors caused by acoustic variability and other factors.

Large text corpora are typically employed to estimate N-gram models, providing the necessary statistical evidence to compute conditional probabilities for various word sequences. In addition, smoothing techniques such as Good-Turing are often applied to ensure robust probability estimation and address the data sparsity issue [17].

Despite their simplicity and reliance on the Markov assumption, N-gram models have been the standard in ASR and other natural language processing tasks until recently. However, with the emergence of deep neural networks, the predominance of N-gram models in applications such as ASR is being contested [40].

2.4 Deep Neural Networks (DNN)

DNNs have delivered breakthroughs in various domains and outperformed earlier technologies once considered state-of-the-art. These networks consist of multiple interconnected layers of artificial neurons capable of learning complex representations and patterns from large amounts of data. The key advantage of DNNs is their ability to extract features and representations from raw input data automatically. This task previously relied on hand-crafted features and expert knowledge. For instance, DNNs have demonstrated superior performance in tasks previously dominated by HMMs, including image classification, natural language processing, and speech recognition [40]. The rise of DNNs can be attributed to several factors, including advances in hardware, such as graphics processing unit (GPU), which enable faster training of large-scale networks, and the availability of large amounts of data, facilitating their ability to learn increasingly sophisticated representations.

2.4.1 Recurrent Neural Network (RNN)

RNNs represent a distinct category of artificial neural networks designed to process and model sequential data by capturing the temporal dependencies present within input sequences [28]. In contrast to feedforward neural networks, which generate output based solely on the current input, RNNs utilize hidden states that retain information from preceding time steps, facilitating the identification and learning of patterns. This unique attribute renders RNNs particularly suitable applications such as speech recognition [55].

The architectural composition of an RNN encompasses input, hidden, and output layers. The input layer receives the input sequence, while the hidden layer processes the input recurrently at each time step and sustains the hidden state. Subsequently, the output layer generates the ultimate output, contingent upon the hidden state [58]. The defining characteristic of RNNs lies in recurrent connections within the hidden layer, which permits information sharing across time steps, thus effectively empowering the network to "remember" prior inputs. This capacity to retain information from previous time steps and leverage it in processing future inputs sets RNNs apart from other neural network architectures.

Training RNNs usually necessitates using the backpropagation through time (BPTT) algorithm, which modifies the conventional backpropagation algorithm employed in training feedforward neural networks [90]. BPTT operates by temporally unfolding the recurrent structure of the RNN, effectively transforming it into an equivalent feedforward network with multiple layers corresponding to each time step. This process allows for the computation of gradients at each time step and the subsequent update of the network's weights in a manner that considers the temporal dependencies inherent in the input sequences [91].

While RNNs can model temporal dependencies, they encounter the vanishing and exploding

gradient problem when processing lengthy input sequences [13]. This challenge emerges during training when the gradients either diminish or expand excessively, making it difficult for the network to learn dependencies spanning long intervals effectively. The vanishing gradient issue is particularly problematic, as it results in the network needing to be more capable of capturing crucial information from earlier time steps, thus impeding its performance in tasks that require an understanding of long-term dependencies [69].

2.4.1.1 Long Short-Term Memory (LSTM)

LSTM is a specialized type of RNN architecture introduced to address the limitations of traditional RNNs in learning long-range dependencies within input sequences [41]. As a result, LSTMs have been particularly successful in various sequence-to-sequence learning tasks, such as natural language processing, speech recognition, and machine translation. The key innovation in LSTM networks is the introduction of a memory cell. This differentiable data structure can store information over extended periods, allowing the network to learn and remember information across long input sequences. A set of gating mechanisms regulates this memory cell [30]:

Input Gate: controls the flow of information from the current input and the previous hidden state into the memory cell. It utilizes a sigmoid activation function, which outputs a value from 0 to 1 to determine the degree to which the input should be incorporated. The input gate's output is elementwise multiplied by the candidate memory cell state, generated by applying a tanh activation function to a linear combination of the current input and previous hidden state, allowing selective updates to the memory cell state.

Forget Gate: a critical component of an LSTM cell that manages the retention of information in the memory cell across time steps. Like the input gate, the forget gate employs a sigmoid activation function, producing an output in the range of 0 to 1, which signifies the proportion of the previous memory cell state to retain. A value close to 0 indicates that the corresponding element of the memory cell state should be forgotten, while a value close to 1 suggests retaining the information. The forget gate's output is elementwise multiplied by the previous memory cell state to regulate information retention.

Output Gate: modulates the contribution of the memory cell state to the current hidden state, which is used to compute the network's output. The output gate operates similarly to the input and forget gates, using a sigmoid activation function to produce a value in the range of 0 to 1 for each element of the memory cell state. The output gate's output is elementwise multiplied by the memory cell state after applying a tanh activation function, yielding the updated hidden state. This mechanism enables the LSTM cell to selectively output information from the memory cell state based on the current input and previous hidden state.

LSTMs have been further extended and refined through various modifications and improvements, such as the introduction of peephole connections, which allow the gates to access the memory cell's internal state directly. LSTM networks represent a significant advancement in RNN architectures, enabling the effective learning of long-range dependencies in sequence data. Their memory cell and gating mechanisms have proven instrumental in addressing the limitations of traditional RNNs and have contributed to the success of LSTMs in various applications.

2.4.1.2 Gated Recurrent Unit (GRU)

GRU is a variant of RNN architecture introduced to address the vanishing gradient problem [18]. Like LSTMs, GRUs have gating mechanisms that regulate the flow of information within the recurrent hidden layers. However, GRUs employ a simpler architecture with fewer gates, resulting in a more computationally efficient model while still being able to capture long-range dependencies in sequential data effectively.

The GRU architecture was introduced as an alternative to LSTMs in the context of machine translation and natural language processing tasks. Since its development, GRUs have been widely adopted and have demonstrated competitive performance in various applications, including speech recognition, time series prediction, and even computer vision.

The architecture of GRU consists of two gates: the update gate and the reset gate. These gates work in tandem to control the information flow within the recurrent hidden layers, enabling the model to learn and maintain relevant information across time steps.

Update Gate: responsible for determining the extent to which the hidden state from the previous time step should be retained. Like the gates in an LSTM, the update gate uses a sigmoid activation function to generate values between 0 to 1 for each element of the hidden state. An output close to 0 indicates that the previous hidden state should be replaced with new information, whereas a value close to 1 implies retaining the existing hidden state.

Reset Gate: The reset gate, also employing a sigmoid activation function, modulates the incorporation of the previous hidden state when computing the candidate hidden state for the current time step. The output of the reset gate is elementwise multiplied by the previous hidden state, which is then combined with the current input and passed through a tanh activation function to generate the candidate hidden state. A value close to 0 for the reset gate output implies that the previous hidden state should have minimal influence on the candidate's hidden state. In contrast, a value close to 1 indicates that the previous hidden state should be considered.

The final hidden state for the current time step in a GRU is computed as a linear interpolation between the previous hidden state and the candidate hidden state, controlled by the update gate. This interpolation mechanism allows the GRU to balance the retention of long-range dependencies and the incorporation of new information, enabling the model to learn complex temporal patterns in sequential data.

Despite their reduced complexity compared to LSTMs, GRUs have demonstrated competitive performance in various tasks, including natural language processing, speech recognition, and time series prediction [19]. The choice between GRUs and LSTMs often depends on the specific application and computational constraints. Both architectures have proven effective in modeling sequential data with long-range dependencies.

2.4.2 Connectionist temporal classification (CTC)

CTC was introduced in 2006 as a novel training criterion for recurrent neural networks, specifically designed for sequence labeling tasks such as ASR, where the alignment between input and output sequences is unknown [35].

Previous ASR systems typically relied on separate acoustic and language models to convert spoken language into written text. The acoustic model, often based on HMMs or GMMs, captured the relationship between speech signals and corresponding phonetic units [44]. The training process involved estimating the parameters of these models using a large, labeled corpus, where the speech features aligned with the correct phonetic units. The language model, usually an N-gram model, was trained on text data to estimate the probability of word sequences [16]. During the decoding process, the acoustic and language models were combined to find the most likely transcription of the input speech signal, considering both the acoustic evidence and the linguistic context. These systems often required hand-crafted features and precise alignments between input and output sequences.

CTC is primarily used with RNNs due to their ability to model time-series data effectively, although it can also be applied to other deep learning architectures. It utilizes a novel approach

to labeling sequences by introducing a unique "blank" symbol alongside the standard output labels. The blank symbol enables the model to output a variable-length prediction sequence, which is subsequently collapsed into the final target label sequence. This process eliminates the need for an explicit alignment between input and output sequences, implicitly allowing the model to learn the optimal alignment during training.

Forward pass: During the forward pass, the input sequence is fed into the neural network, which computes a probability distribution over the extended label set (including the blank symbol) for each time step. The network output at each time step can be viewed as an unnormalized probability distribution over the labels and the blank symbol.

Forward-backward algorithm: The forward-backward algorithm is employed to compute the probability of the target label sequence given the unnormalized probabilities obtained in the forward pass. This algorithm recursively computes the forward and backward probabilities by summing over all possible alignments of the input sequence to the target sequence, considering the blank symbol.

CTC loss calculation: The CTC loss, which measures the negative log-likelihood of the target label sequence given the input sequence, is calculated based on the forward and backward probabilities. CTC training aims to minimize this loss, increasing the likelihood of the correct target sequence.

Backward pass: During the backward pass, the gradients of the CTC loss with respect to the network parameters are computed using the backpropagation algorithm. These gradients are then used to update the network parameters through an optimization process, such as stochastic gradient descent or its variants.

Decoding: Once the model has been trained, it can generate label sequences for new input sequences. The decoding process involves finding the most probable label sequence given the output of the trained network. Several decoding algorithms are available, such as the best-path and prefix search decoding, which aim to find the optimal label sequence efficiently.

CTC can implicitly learn the optimal alignment between input and output sequences during training, allowing it to model the temporal dependencies in speech signals directly. The network inputs raw speech features and generates character or phoneme probabilities at each time step. The CTC loss function ensures that the network learns to align the output sequence with the input sequence without requiring precise alignments or hand-crafted features.

CTC simplifies the ASR pipeline and reduces the system's complexity by integrating acoustic and language modeling tasks into a single neural network architecture. This end-to-end approach results in a more streamlined and efficient ASR system. It eliminates the need for hand-engineered features, separate acoustic and language models, and precise alignments between input and output sequences. CTC-based ASR systems have demonstrated significant improvements in performance and scalability compared to traditional ASR systems, solidifying their position as a powerful and efficient approach to sequence labeling tasks, particularly in the field of ASR.

2.4.3 Convolution Neural Network (CNN)

A CNN is a specialized artificial neural network tailored to process grid-like structured data, such as images or time-series data [51]. Due to their outstanding performance in computer vision tasks, including image classification, object detection, and segmentation, CNNs have gained significant traction in recent years. Convolutional operations, which facilitate learning local patterns in input data, are the cornerstone of CNNs, making them highly effective for image processing tasks where features like edges or textures are often localized within small regions [50].

Furthermore, CNNs can autonomously learn these features without prior knowledge or manual feature engineering, conferring a substantial advantage.

The ability of CNNs to exploit spatial relationships between input data renders them ideal for object recognition tasks. In addition, CNNs learn hierarchical features, enabling them to recognize complex patterns by building upon simpler ones, which makes them applicable to a diverse range of applications.

Convolutional layers, the building blocks of CNNs, are critical in capturing local patterns and features in input images or feature maps. These layers perform convolution operations by applying filters, also known as kernels, designed to detect specific patterns like edges, corners, and textures. The filters are composed of learnable parameters (weights) and are optimized during training to learn the most informative features for the given task.

The output of a convolutional layer consists of a set of feature maps, each corresponding to a particular filter. The dimensions of these feature maps depend on factors such as the number of filters, their spatial dimensions, and the stride [33]. The stride, the step size for moving the filter across the input, contributes to the network's computational efficiency by reducing the spatial dimensions of feature maps when larger than one.

Padding, the process of adding extra pixels around the input image or feature maps, is applied during the convolution operation to control the output feature maps' spatial dimensions. It is beneficial for preserving spatial resolution, especially in deep architectures where successive convolutional layers can significantly reduce the resolution.

Following the convolution operation, an element-wise non-linear activation function is applied to the output feature maps to introduce non-linearity into the network. This is crucial for learning complex, non-linear relationships between input and output data and modeling a broader range of functions.

The Rectified Linear Unit (ReLU) is a popular activation function in modern CNN architectures, known for its computational efficiency and effectiveness in addressing the vanishing gradient problem [62]. The ReLU function is defined as $f(x) = max(0, x)$, where x represents the input value. By setting negative input values to zero and retaining positive values, ReLU offers a piecewise-linear, non-linear transformation that accelerates the training process convergence. Activation functions play a vital role in introducing non-linearity into CNNs, enabling them to model complex input-output data relationships. The ReLU function and its variants are prevalent in modern CNN architectures, given their computational efficiency and ability to mitigate the vanishing gradient problem.

Pooling layers are essential for reducing the spatial dimensions of feature maps and enhancing the network's computational efficiency and translational invariance. Typically placed between consecutive convolutional layers, pooling layers condense the spatial information of feature maps while retaining the most significant features. This results in a more robust network that can recognize patterns even when subjected to small spatial shifts. This property is handy in object recognition and classification tasks where the precise spatial location of a feature might be less important than its presence in the input image.

A CNN architecture often uses one or more fully connected layers, with the final fully connected layer generating the output predictions. The output layer uses a softmax activation function for multi-class classification tasks, converting logits into class probabilities [14]. In addition, a linear activation function is employed in regression tasks to generate continuous-valued outputs.

CNNs have emerged as a promising ASR technique due to their ability to model local and hierarchical features crucial for speech pattern recognition. The most significant advantage of using CNNs in ASR is their ability to automatically learn relevant features from input data without manual feature engineering. CNNs exploit local dependencies in time-frequency representation

to capture critical patterns, such as phonemes, necessary for speech recognition.

In a typical CNN-based ASR system, input spectrograms or MFCCs pass through multiple convolutional and pooling layers to learn high-level, abstract features representing speech signals' acoustic properties. Fully connected or recurrent layers (e.g., LSTM or GRU) then use these features to model temporal dependencies in speech data. Finally, the output layer generates a probability distribution over target phoneme or subword classes using a softmax activation function. CNN-based ASR systems have outperformed traditional techniques like GMM-HMMs-based systems and have become essential in state-of-the-art ASR systems like hybrid CNN-HMM or end-to-end architectures.

2.5 Common corpora

LibriSpeech is a large-scale corpus of read English speech comprising approximately 1,000 hours of speech data derived from audiobooks in the public domain [68]. The corpus is designed for Automatic Speech Recognition (ASR) research and evaluation and includes over 2,500 unique speakers and over 100,000 utterances. The recordings align with their corresponding text transcripts at the word level, providing ground truth for training and evaluating ASR systems. The corpus is publicly available and has been, and continues to be, used as a benchmark for numerous studies in ASR, including the development and evaluation of deep learning-based models.

The Switchboard corpus is a corpus of telephone conversations in English collected in the early 1990s by the Texas Instruments corporation [32]. The corpus contains transcriptions of over 2,400 two-sided telephone conversations between native speakers of American English, covering various topics and speech styles. Each conversation is approximately 5 to 10 minutes long, and the corpus includes word-level and phonetic transcriptions, speaker information, and metadata. Due to its size, diversity, and realistic nature, the Switchboard corpus has been widely used for research in speech recognition, natural language processing, and other fields.

The Wall Street Journal (WSJ) Continuous Speech Recognition (CSR) Corpus is a comprehensive collection of read English speech, encompassing roughly 80 hours of audio data from WSJ's business and economic news articles [70]. The corpus is tailored for Automatic Speech Recognition (ASR) research and analysis, featuring 400 hours of speech and text data of 47 million words. The recorded speech aligns with its corresponding text transcripts at the word level, providing an authoritative source for the training and evaluation of ASR systems. The WSJ Corpus is publicly accessible and has been extensively utilized as a benchmark in numerous ASR studies, including creating and evaluating advanced deep learning-based models.

The TIMIT Acoustic-Phonetic Continuous Speech Corpus (TIMIT) is a detailed collection of read English speech with approximately 5 hours of audio data from reading tasks [29]. The corpus is designed for ASR research and phonetic and acoustic analysis, featuring 630 unique speakers from 8 major dialect regions in the United States and more than 6,300 utterances. The recorded speech is annotated at the word, phoneme, and stress level, providing a rich source for training and evaluating ASR systems. The TIMIT corpus is publicly available and has been extensively used as a benchmark in numerous ASR studies, including creating and assessing sophisticated machine learning and deep learning-based models.

2.5.1 Word error rate metric

Word error rate (WER) [92] is ASR research's most widely used accuracy metric. It evaluates ASR systems' performance and compares different models and techniques. After accounting for insertions, deletions, and substitutions, it measures the percentage of words in the reference transcription that differ from the recognized words produced by the ASR system (Equation 2.4). Table 2.1 shows three examples of how substitution, deletion, and insertion can affect a model's

WER score.

$$WER = \frac{\text{Substitutions} + \text{Deletions} + \text{Insertions}}{\text{Total number of words}} \tag{2.4}$$

Table 2.1: WER examples.

WER 9.1%		
11 words	1 substitution	1 / 11 = 0.0909 ≈ 0.091
hotel echo x ray downwind two five for touch and go hotel echo x ray dowin two five for touch and go		
WER 28.6%		
14 words	2 substitutions, 1 deletion, 1 insertion	4 / 14 = 0.2857 ≈ 0.286
hello sydney good evening jetstar four seventy five climbing flight level two eight zero -(hello) soydne +(tower) good evening fontak four seventy five climbing flight level two eight zero		
WER 66.7%		
6 words	3 substitutions, 1 insertion	4 / 6 = 0.666 ≈ 0.667
sierra alpha papa sion ground hello hotel alpha +(ba) pa sion ground hellheur		

Chapter 3

Related work

This chapter investigates learning strategies employed in Automated Speech Recognition (ASR) within the academic research context. It commences by examining supervised learning methods, including prominent models such as Deep Speech 1 and 2 and the transformative Transformer model. These models rely on labeled corpora to establish precise mappings between inputs and outputs, forming a foundational understanding of ASR methodologies.

The chapter introduces self-supervised learning techniques, specifically focusing on innovative technologies like wav2vec and its refined variant, wav2vec 2.0. These models leverage both labeled and unlabeled data to enhance performance.

Furthermore, the chapter explores the significance of unsupervised learning, particularly the novel wav2vec Unsupervised model. This approach becomes crucial when labeled data is scarce, as it uncovers inherent patterns and relationships within unlabeled data.

Additionally, the chapter delves into the practical applications of ASR in complex and high-stakes environments, such as air traffic control (ATC). Given the intricate linguistic nature of ATC communications, ASR research and implementation play a vital role in this domain.

3.1 ATC Communications Characteristics

ATC is a fundamental component of aviation safety and efficiency, offering vital instructions and guidance to pilots to ensure the aircraft's secure and orderly movement. To facilitate clear and unambiguous communication, specific rules and guidelines have been established that must be followed, in English, as effective communication between pilots and air traffic controllers is of paramount importance [73]. These communication guidelines include using standardized phraseology, employing clear and concise language, repeating or confirming instructions to ensure understanding, acknowledging the receipt of instructions to confirm compliance, and reporting essential information promptly to maintain situational awareness [7].

Although they seem to have apparent similarities, ATC communications differ significantly from standard spoken English and subsequently pose unique challenges for ASR systems compared to other domains, primarily due to the following factors:

Specialized vocabulary and phraseology: ATC communications involve domain-specific terminology and standardized phrases unique to aviation. This specialized language may need to be better represented in general ASR training data, potentially leading to lower recognition accuracy.

Accents and dialects: Pilots and air traffic controllers come from various regions and countries, leading to a wide range of accents and dialects in ATC communications. ASR systems need to

be robust enough to recognize speech across these variations.

Radio communication artifacts: ATC communications typically occur over radio channels, which can introduce noise, distortion, and signal degradation. These artifacts can make speech recognition more challenging for ASR systems.

Background noise: The ATC environment often includes background noise from aircraft engines, weather, and other sources. ASR systems must perform well in these noisy conditions to accurately transcribe ATC communications.

Multi-speaker and overlapping speech: ATC communications involve multiple speakers, including pilots and controllers, who may talk simultaneously or with a significant difference in utterances per second. ASR systems need to be capable of handling overlapping speech and distinguishing between different speakers.

Time-sensitive information: ATC communications involve critical information for maintaining air traffic safety and efficiency; therefore, ASR systems must be able to process and transcribe speech in real-time to ensure that the relevant information is available promptly for decision-making.

High-stakes environment: Errors in ASR transcription can have severe consequences in ATC communications, as misunderstandings or miscommunications can compromise safety. ASR systems must achieve high accuracy and reliability to suit this high-stakes environment.

Given these unique challenges, ASR systems designed for ATC communications must be specialized and optimized to handle the aviation environment's domain-specific language, noise conditions, and real-time requirements.

3.2 Supervised learning

This approach involves training a machine learning model utilizing a labeled corpus, where each input data instance (speech signal) is accompanied by its corresponding output transcript (label). Through this process, the model endeavors to establish a mapping from inputs to outputs, to minimize the disparity between the model's predictions and the true labels, generally quantified through a loss function. However, this learning paradigm necessitates the availability of substantial amounts of annotated data, which can prove to be a time-intensive and costly endeavor, particularly in the case of low-resource languages.

3.2.1 Deep Speech

Introduced in 2014 by Baidu[1], Deep Speech is an end-to-end ASR system designed to simplify the speech recognition pipeline by leveraging deep learning techniques [39]. The system addresses the limitations of traditional ASR systems, which typically consist of multiple complex stages, including feature extraction, acoustic modeling, and language modeling. By adopting a deep learning approach, Deep Speech reduces the complexity of ASR, resulting in a single, end-to-end model that directly maps input acoustic features to output character transcriptions.

The architecture of Deep Speech consists of three main layers: input, recurrent, and output layers. The input layer processes the raw audio features, such as log Mel-filterbank energies, which represent the spectral content of the input speech signal. These features are fed into the recurrent layer, which models the temporal dependencies in the speech data. Deep Speech employs bi-directional Recurrent Neural Networks (RNNs) with Long Short-Term Memory (LSTM) units to capture context from past and future speech frames, enabling the model to learn complex and long-range temporal dependencies.

[1] `https://www.baidu.com`

The model produces character probabilities in the output layer, decoded into the final transcription using the CTC loss function. During training, the model learns to minimize the CTC loss, which measures the likelihood of the target transcription given the input acoustic features. In training, the model uses stochastic gradient descent with a learning rate schedule that decays the learning rate over time. It is regularized using dropout and weight clipping to prevent overfitting.

One of the main advantages of Deep Speech's end-to-end approach is that it eliminates the need for hand-crafted features and separate modeling stages, which can introduce errors and limit the system's ability to learn from the data. Instead, by directly learning the mapping from input features to output characters, the model can exploit the raw acoustic information and automatically discover useful representations for speech recognition.

Deep Speech demonstrated that an end-to-end deep learning approach could achieve competitive performance on challenging ASR tasks, outperforming traditional, multi-stage systems in some instances. In the original Deep Speech paper, the authors show that the model outperforms hybrid DNN-HMM models on the Switchboard corpus with ~ 4 percentage points using the WER metric.

Overall, Deep Speech represents a significant milestone in applying deep learning to speech recognition, showcasing the potential of end-to-end models to simplify the ASR pipeline and achieve high performance. The system's success laid the foundation for future research in end-to-end ASR, leading to more advanced models.

Deep Speech 2 incorporates additional convolutional layers before the recurrent layers, allowing the model to capture local structures in the input data more effectively [4]. These layers employ multiple kernel sizes, capturing different levels of granularity in the spectral features. Deep Speech 2 also introduces batch normalization, which accelerates training and improves generalization by normalizing the activations at each layer.

Designed to scale efficiently on large-scale distributed GPU clusters, it enables training deeper and broader networks on massive amounts of data. This scalability allowed Baidu to train Deep Speech 2 on a corpus of approximately 12,000 hours of multilingual and multitask supervised data, showing a WER of 8.46% for regular data and 5.15% for the LibriSpeech test-clean set. It also demonstrated the effectiveness of transfer learning in ASR by pretraining the model on a large corpus and then fine-tuning it on smaller, domain-specific corpora.

3.2.2 Transformers

First introduced in 2017, transformer networks are a class of neural architectures that have revolutionized various domains in natural language processing, such as machine translation and text classification [88]. Transformers rely on self-attention mechanisms to process input sequences, providing a more efficient and flexible alternative to traditional recurrent and convolutional neural networks.

The Transformer architecture features two key parts: an encoder and a decoder, built from a series of sub-layers. The encoder converts the input sequence into context-sensitive representations using self-attention mechanisms and feed-forward neural networks. This design enables the encoder to capture the full context of the input sequence. The decoder, in turn, creates the output sequence based on the encoder's output and its own previously generated tokens. Each decoder layer contains two self-attention mechanisms and a feed-forward network. The first self-attention mechanism is masked to ensure output dependency on known outputs only, while the second attends to all encoder outputs, enabling context transfer from input to output. The Transformer model thus effectively intertwines the roles of the encoder and decoder to produce context-aware sequential outputs.

Transformer networks have proven to be a powerful tool in ASR, especially when combined with other AI technologies. A CNN-based local feature extraction combination with a transformer encoder called a "Conformer" achieved remarkable results in 2020 [36]. With a WER of 1.9% on the LibriSpeech corpus, it demonstrated the effectiveness of this fusion approach. By leveraging the strengths of both components, the Conformer model enhanced the performance and capabilities of end-to-end ASR systems, opening up new possibilities for accurate and efficient speech recognition.

3.3 Self-supervised learning

Learning vector representations from a high amount of labeled or unlabeled data is a recent trend to leverage the learned representations to improve performance on a local task for which a relatively small corpus is available. It is accomplished by creating a pretext, or auxiliary, task where the model learns to predict or reconstruct certain aspects of the input data based on other parts. In the case of ASR, a common strategy involves masking out sections of the input audio signal and challenging the model to predict the missing segments from the context provided by the unmasked sections.

3.3.1 wav2vec

The original wav2vec framework introduced in 2019 presents a novel approach for unsupervised pre-training of neural networks to improve ASR [80]. The wav2vec model is designed to learn meaningful representations of raw audio waveforms, which can be subsequently fine-tuned for supervised speech recognition tasks. The idea behind wav2vec is to leverage the structure in raw audio waveforms to learn high-level features that can benefit ASR. The model consists of a multi-layer CNN that operates directly on the raw audio waveform, followed by a context aggregation module that captures higher-level temporal dependencies across the audio signal.

The learning process is unsupervised and relies on a contrastive predictive coding (CPC) objective, an unsupervised learning objective designed to create valuable representations of data by predicting future samples from past contexts in a contrastive manner [66]. CPC works in two main stages: encoding and prediction. In the encoding phase, an encoder network (often a CNN or RNN) processes input data (like audio waveforms or image pixels) into a sequence of latent representations or embeddings. In the prediction phase, another network, the autoregressive model, uses the context of past embeddings to predict future embeddings. Using a contrastive loss, the predicted future embeddings are compared against the true future embeddings.

The contrastive loss is the key to CPC. It encourages the model to make the predicted future embeddings as similar as possible to the actual future embeddings (positive samples) while making them as dissimilar as possible from other embeddings in the batch (negative samples). This is typically achieved through a softmax function, resulting in a loss that can be optimized with gradient descent. By learning to predict future embeddings, the model implicitly learns to capture valuable information in the embeddings about the structure and content of the data. The hope is that these embeddings, learned without any task-specific supervision, can serve as a helpful starting point for downstream tasks, reducing the amount of labeled data required or improving performance.

This approach has been efficient for audio and speech data, as in the wav2vec model, but has also been applied to other types of data like images and text. Once the unsupervised pre-training is completed, wav2vec representations can be fine-tuned using supervised data for the target ASR task. The fine-tuning process typically involves training a shallow classifier on top of the pre-trained wav2vec representations to predict phonemes, characters, or words, depending on the desired output granularity.

The original wav2vec paper demonstrated significant improvements in ASR performance on the LibriSpeech corpus compared to models trained from scratch or with traditional feature extraction methods, such as MFCCs or filter banks. The results highlighted the potential of unsupervised pre-training for ASR and set the stage for further advancements of wav2vec models, such as wav2vec 2.0.

3.3.2 wav2vec 2.0

wav2vec 2.0 builds upon the original wav2vec framework by introducing a novel contrastive loss that operates in the latent space of quantized representations, effectively turning the unsupervised learning problem into more of a supervised learning problem [10]. The model architecture consists of two main parts: a feature encoder and a context network.

The feature encoder in wav2vec 2.0 transforms raw audio into a sequence of latent representations or embeddings through a stack of convolutional layers, capturing local audio patterns. This results in continuous representations encapsulating short-term audio characteristics, which are discretized into a codebook of audio features.

The context network, a transformer using self-attention mechanisms, models the dependencies within the input sequence to capture higher-level temporal dependencies across these representations. This acknowledges the relational nature of audio elements over time. Training focuses on correctly classifying accurate quantized representations at each time step, contributing to more efficient training and superior performance in downstream speech recognition tasks.

3.4 Unsupervised learning

Unsupervised learning operates, as the name suggests, without a labeled corpus. In this paradigm, algorithms identify inherent patterns, structures, and relationships within the provided data, which can be highly beneficial when extensively labeled corpora are hard to come by. The approach often involves learning vector representations from significant amounts of data to enhance task-specific performance. This includes creating a pretext task, wherein the model predicts or reconstructs aspects of the input data based on the remaining data. This process allows the model to understand complex features within the data, thereby enhancing its understanding of the data's underlying structure and context.

3.4.1 wav2vec Unsupervised

This novel approach introduced in 2021 enables ASR without reliance on transcribed data [9]. Its novelty lies in its fully unsupervised training approach, which allows the model to learn directly from unlabeled audio data and random text data from an unrelated source in the same language.

Two key components at the wav2vec-U framework are the feature encoder and the context network. The feature encoder is a convolutional neural network (CNN) that directly processes the raw audio waveform, creating a sequence of latent speech representations. The context network is a transformer model that processes these latent representations, capturing the higher-level temporal dependencies in the audio data.

The unsupervised training of wav2vec-U involves a two-step process. First, the model is pre-trained on a large corpus of unlabeled audio data. In the pre-processing stage, wav2vec-U performs various feature engineering steps on the speech representations. These include k-means clustering of input frames, dimensionality reduction using principal component analysis (PCA), merging adjacent PCA features with the same k-means cluster, and mean pooling of features from adjacent timesteps to align with the length of phone sequences. This pre-training

process mirrors that of wav2vec 2.0, using a contrastive loss function that encourages the model to correctly identify true future audio samples from negative ones. This helps the model learn meaningful representations of the audio data.

The second stage involves fine-tuning the model in an unsupervised manner. This is achieved by training the model to predict pseudo labels generated by running the pre-trained model on the unlabeled data and using its predictions as 'ground truth.' This novel approach allows the model to learn from its predictions without needing human-annotated labels.

The pseudo labels are iteratively refined over multiple training rounds using a generative adversarial network (GAN) setup [34]. This adversarial process involves two main components: a generator and a discriminator. The generator is the pre-trained wav2vec 2.0 model, fine-tuned on the pseudo labels, while the discriminator is another transformer model trained to differentiate between the pseudo and true labels.

The GAN procedure begins with the generator creating an initial set of pseudo labels from the raw audio data. These pseudo labels serve as the 'generated' data in the GAN setup. The discriminator is then trained to differentiate between these generated pseudos and true pseudo labels. In turn, the generator is updated to fool the discriminator by making the pseudo labels more similar to the true labels. This adversarial process results in gradually refining the pseudo labels over several iterations. This GAN-based approach allows wav2vec-U to overcome the lack of labeled data challenge. It enables the model to improve its pseudo labels iteratively, hence its ASR task performance.

3.4.2 wav2vec-U 2.0

In this updated framework, findings suggest that the pre-processing step in wav2vec-U may not be necessary, as it was discovered that a generator could effectively perform unsupervised ASR without speech segmentation [56]. The generator can achieve the desired output frequency by utilizing the raw speech representation sequence as input, which refers to the number of phones predicted per second. Adjusting the stride of the convolutional neural network was identified as an effective method to control the output frequency. This approach demonstrated consistent results across various languages tested in the study. The downsampling process involved treating consecutive outputs sharing the same most probable phone as a single output through random sampling. In addition, replacing PCA-based dimensionality reduction with batch normalization along the time axis of the wav2vec 2.0 features was also a viable alternative.

The input consists of the raw representations extracted from the wav2vec 2.0 model, eliminating the need for a complex pre-processing pipeline. The end-to-end learning approach addresses the limitations associated with hand-crafted pre-processing, as it directly learned the mapping from speech representations to phone sequences. Since speech transcriptions were not available in the unsupervised setting, the generator relied solely on adversarial training to establish the mapping between speech representations and phone units.

However, it was observed that the generator could potentially fulfill the adversarial criterion without learning the correct mapping by consistently generating the most common n-grams, regardless of the input speech. To mitigate this issue, an auxiliary objective function was introduced to facilitate self-supervised learning. This involved reconstructing pseudo labels derived from the input audio using k-means clustering on MFCC features with 64 clusters. Incorporating the auxiliary loss served a dual purpose: it provided content-based regularization to ensure the generator's output maintained a close relationship with the input speech, and it offered explicit guidance by approximating the underlying phone sequence with pseudo labels derived from speech.

3.5 General ASR

Significant advancements have been made in ASR for native-spoken English. However, the performance of state-of-the-art models in low-resource languages has received limited attention. In a 2021 publication, researchers explored applying the wav2vec 2.0 model to address ASR challenges in Mandarin, English, Japanese, Arabic, German, and Spanish [93]. It demonstrated a notable 20% relative improvement compared to earlier studies. This investigation sheds light on the potential of leveraging wav2vec 2.0 in diverse language settings and highlights its capacity to enhance ASR performance in low-resource language scenarios.

While unsupervised learning in ASR is highly relevant due to the challenges posed by low-resource languages, published literature exploring the use of wav2vec-U 2.0 to tackle such tasks remains scarce. A notable exception is a late 2022 study using Uyghur, Kazakh, and Kyrgyz audio [26]. Despite only having access to 1.8 hours of Kyrgyz data from the Common Voice corpus [5], the study achieved a character error rate (CER) of 14.9%. This accomplishment underscores the potential of unsupervised learning in handling ASR tasks in resource-constrained domains.

Regarding robustness in unsupervised speech recognition, a study analyzed the application of non-corresponding text data in the same language across frequently employed corpora in ASR studies [53]. The research encompasses three mismatch scenarios: varying speech and text corpora, using noisy/spontaneous speech, and adjusting the volume of speech and text data. Experimental outcomes demonstrated that domain mismatches reduced performance; however, pre-training the self-supervised model on targeted speech data significantly improved the phoneme error rate. The results from this research indicate a relationship between the quality of the text input for the GAN and the resultant performance metrics. Notably, it was observed that out-of-domain text data generally yielded inferior results, reinforcing the necessity of text data from the same domain to achieve optimal performance.

A comprehensive survey conducted in 2022 investigated the landscape of self-supervised learning approaches for ASR, encompassing more than 30 models developed since 2015 [59]. The study provided a detailed analysis of existing benchmarks and examined the progress in achieving efficient zero-resource learning. Notably, the survey highlighted the exceptional performance of Wav2vec 2.0 in speech recognition and its broad applicability to various downstream tasks. Furthermore, the wide accessibility of pre-trained Wav2vec 2.0 models and their widespread adoption within the ASR community have made them the preferred choice for ASR applications.

3.6 ATC domain specific ASR

A study published in 2020 outlines the CleanSky EC-H2020 ATCO2 project, which seeks to develop an ASR-based platform for ATC environments using the Kaldi toolkit [45]. The Kaldi toolkit is an acclaimed open-source software package for building advanced ASR systems, offering a flexible and modular framework with a comprehensive range of tools, algorithms, and training techniques [71]. Kaldi supports various ASR components, including acoustic modeling with HMMs, GMMs, DNNs, language modeling, and decoding algorithms. In addition, its modular design facilitates experimentation and the development of new algorithms. Over 170 hours of ATCo speech data was used to train an array of state-of-the-art ASR models (at that time), including several DNN architectures. The results demonstrated an average WER of 7.75% across four databases. A 35% relative WER improvement was recorded on one test set using a Time Delay Neural Network (TDNN) and Factorized TDNN (TDNN-F) system with byte-pair encoding. The optimal system combination of CNN+TDNNF attained a WER of 5.0% on ATCOSIM with a trigram language model.

A study deviating from the wav2vec 2.0-based research presented an ASR model devised explicitly for the air traffic control (ATC) domain, intending to transcribe ATC communications to text and extract operationally pertinent information [8]. Based on the Deep Speech architecture, the model exhibited a WER of 17% on a composite ATC communication test set. Incorporating an N-gram language model alongside the acoustic model improved accuracy by 26%. Interestingly, transfer learning and parameter fine-tuning with a pre-trained model did not augment accuracy. The results underscored the anticipated benefit of employing diverse data sources during training, leading to more generalizable models with better accuracy on varying test corpora.

Moreover, this research illustrates a Named Entity Recognition (NER) based call-sign extraction method and a rule-based grammar system for extracting runway information. The authors envision that increasing the training data and refining the ASR accuracy could bolster the precision of call-sign extraction. They also suggest potential improvements for ASR models in the ATC domain, such as better language models, semi-supervised learning, superior priors or context, and increased quantities of transcribed ATC speech data. These enhancements could facilitate various practical applications, from real-time safety monitoring to speech assistant tools for air traffic controllers.

A key challenge in ATC is the limited available data. A recent study addressed this challenge by employing transfer learning techniques and leveraging multiple Chinese speech corpora [54]. This research proposes an unsupervised pre-training strategy involving a masking technique to diversify samples first utilized to extract speech representations from unlabeled data. This is followed by transfer learning to fine-tune a pre-trained model for ATC-specific ASR tasks. This two-step approach leverages common ATC terminology, using a joint corpus of existing and newly transcribed samples to enrich the training data, supplemented by speed perturbation to enhance the corpus quality. The effectiveness of this approach is validated using three real ATC corpora, demonstrating significant improvement in ASR performance compared to traditional supervised training methods. The study showed promising results, with an overall reduction of Character Error Rate (CER) of 2% when compared to the supervised learning approach. In future work, the authors propose integrating adversarial learning for real-time data augmentation and unifying the framework with the ASR model.

Another recently published study proposes a framework combining Self-Supervised Learning and Multi-Task Learning in a two-stage training paradigm [37]. It utilizes the wav2vec 2.0 model to learn universal acoustic representations and then fine-tunes it through joint training with ASR, Speaker Role Identification, and Language Identification tasks. An attention-guided feature aggregation module is introduced to capture task-specific representations, while an uncertainty-based loss combination strategy balances the loss weights for each task. The study uses a corpus of 1032 hours of unlabeled real-world Chinese and English ATC audio. Experimental evaluation demonstrates superior performance compared to competitive baselines across all tasks, with a Label Error Rate (LER) of 2.6% for this novel framework compared to an LER of 13.9% for a Conformer-based model.

Recent academic research has concentrated on exploiting large-scale, unlabeled speech data for constructing robust E2E acoustic models through self-supervised learning, with the goal of fine-tuning these models on downstream tasks. However, fewer studies have analyzed the impact of domain shift, where data properties significantly diverge between the pre-training and fine-tuning phases. A study published in late 2022 ambitiously addresses this domain shift scenario, scrutinizing the resilience of pre-trained Wav2Vec 2.0 and XLS-R models on the novel domain of ATC for downstream ASR tasks [94]. By benchmarking these models against open-source ATC databases, the research reports relative Word Error Rate (WER) reductions of 20% to 40% compared to hybrid-based ASR baselines, achieved merely by fine-tuning E2E models with limited labeled data. The study also highlights gender bias within ATCOSIM, with speech

models demonstrating systematically superior performance on female recordings. The results highlight the potential of implementing pre-trained wav2vec 2.0 models for ASR tasks within specialized domains.

Using pre-trained wav2vec 2.0 models for ASR tasks in the air traffic control domain underscores the value of transfer learning and domain-specific fine-tuning. The versatility and adaptability of wav2vec 2.0 models make them ideal candidates for a range of specialized ASR tasks, including those in the air traffic control domain. Furthermore, the end-to-end nature of these models, as evidenced by their superior performance compared to hybrid-based ASR systems, emphasizes the importance of investigating and developing more streamlined and efficient ASR architectures.

Chapter 4

Methodology

This chapter offers a concise description of the key aspects involved in our research of Automatic Speech Recognition (ASR) technologies in the Air Traffic Control (ATC) domain, as described in Chapter 3. In accordance with RQ1 described in Section 1.2, our methodology has been meticulously designed to facilitate a thorough understanding of the multifaceted aspects associated with the development and evaluation of ASR systems while ensuring the reliability and reproducibility of our research process. The chapter is structured into seven primary sections: Schedule, Setup, Corpora, and the four ASR Technologies used in our research.

The Schedule section delineates an exhaustive timeline outlining the project's various stages, from initial planning and preparatory phases to data acquisition, implementation, and evaluation. The timeline functions as a guiding framework, enabling seamless execution of the project and the effective monitoring of progress throughout the research.

Subsequently, the Setup section expounds upon the hardware and software prerequisites necessary for the execution of the research, detailing the specifications of computational resources, programming languages, and tools employed.

In the Corpora section, we present the assortment of the data employed in our exploration. These corpora, comprising proprietary speech data and transcriptions, were selected to represent a broad spectrum of speakers, accents, and acoustic environments. This strategy ensures a comprehensive evaluation of the ASR technologies under scrutiny.

The last sections delves into the deployed methodologies and algorithms of the four ASR technologies utilized throughout our research. By providing a lucid and extensive account of our methodology, we aim to contribute to the progress of ASR research and inspire further exploration within the field. This condensed exploration aims to provide a clear understanding of how the research questions in Section 1.2 are comprehensively addressed.

4.1 Schedule

The workflow throughout the project was heavily centered around the agile methodology[1], meticulously designed to accommodate a structured approach to achieving results within the time frame of the project. This approach, particularly efficient for managing complex projects, involves recurring biweekly meetings with the client and the supervising professor, fostering a space for continuous presentation, integration, feedback, and dynamic adjustments.

Our workflow is cyclical, divided into a literature phase, two distinct loops, and a documentation phase, each serving a unique purpose in the overall project life cycle as shown in Figure 4.1.

[1] `https://agilemanifesto.org`

The literature phase is concerned with project planning and conducting exhaustive literature research. This phase aims to set the project's scope and direction, taking into account the goals and requirements set academically and by the client. During this stage, a rigorous literature review is conducted to ascertain the state-of-the-art in the field of ASR and map relevant corpora utilized.

Upon completing the literature phase, we transition into loop 1, which focuses on data preparation, framework setup, implementation, and evaluation of the selected supervised and self-supervised technologies. In this loop, the theoretical knowledge garnered from the previous stage is transformed into tangible custom executables, enabling learning sequences with hyperparameter testing and model evaluation.

In alignment with the established pattern, loop 2 delves into an unsupervised generative approach to the subject matter, leveraging the self-supervised models utilized in loop 1. Owing to the intricate nature of unsupervised learning and the early stage of the framework's development, the central objective revolves around generating outcomes that can be benchmarked against prior work, facilitating a comprehensive evaluation, and fostering continuous improvement in the development process.

The three-phase process culminates in an extensive documentation effort comprising two deliverables: a conference paper and a Master's thesis. The paper concisely presents the project's aims, discoveries, and outcomes. The more extensive thesis delves into granular details of the project's stages. It serves as a comprehensive chronicle of the project's evolution from its genesis to fruition, underscoring the meticulous and adaptable approach that steered its trajectory toward success.

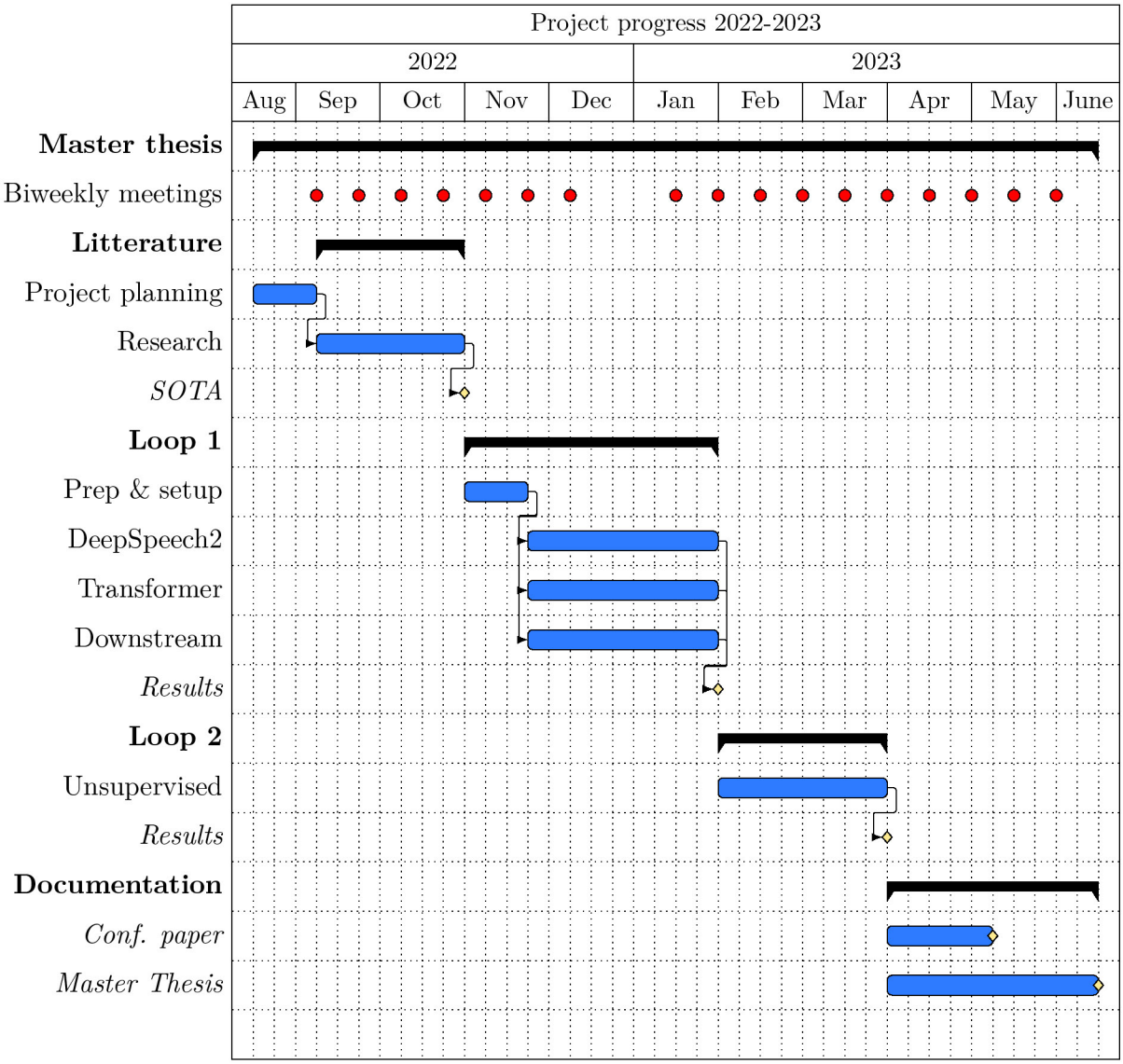


Figure 4.1: Gantt chart showing the projected loops and sprints outlining the timeline of the thesis project.

4.2 Setup

In order to successfully execute the project loops, the hardware and software setup is of paramount importance. In this section, we describe the key elements and considerations for creating a reliable and efficient environment for project implementation. The setup process is divided into two sub-sections: Hardware and Software.

The Hardware sub-section outlines the essential components and specifications to ensure optimal performance and seamless project execution. These components provide a solid foundation for the various project stages and minimize potential bottlenecks.

The Software sub-section delves into the necessary tools and applications that support the project's requirements and goals.

4.2.1 Hardware

In Automatic Speech Recognition (ASR), the computational requirements for development and training are substantial, necessitating robust hardware configurations, and resources are essential to facilitate the experiments described in our research. Throughout the process, most computational activities have been conducted on resources provided by the Faculty of Computer Sciences, part of Østfold university college (HiØ) (Figure 4.2).

The group had shared access to a dedicated machine-learning node containing two NVIDIA A100 Tensor Core graphics processing unit (GPU)s[2], each boasting 80 GB of RAM. The A100 GPUs, based on NVIDIA's Ampere architecture, are optimized for large-scale performance, making them ideal for handling the complex tensor operations needed in machine-learning categories that require substantial resources, like Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN), Generative Adversarial Networks (GAN), and Transformers.

Furthermore, the high memory allowance of 80 GB RAM per GPU is particularly beneficial for processing large corpora, enabling faster data manipulation and model training. This configuration is complemented by 256 AMD EPYC 7763 processors[3]. These processors provide a substantial boost in processing power. In addition, the high core count allows for efficient parallel computation and multitasking, essential for the data-intensive workloads in machine-learning and ASR domain. However, it should be noted that the ASR frameworks utilized in our experiments would not be achievable on the hardware of standard consumer-grade quality.

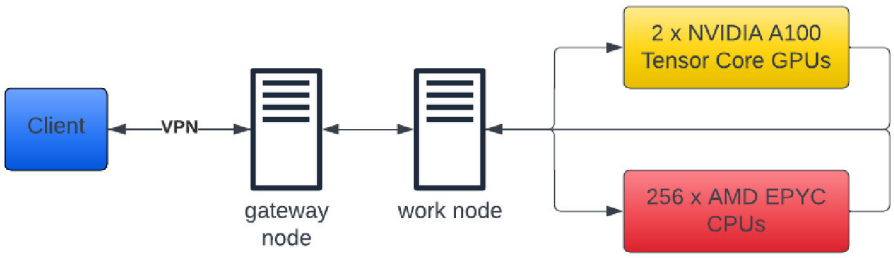


Figure 4.2: Illustration of the hardware setup.

[2] https://www.nvidia.com/en-us/data-center/a100/
[3] https://www.amd.com/en/products/cpu/amd-epyc-7763

4.2.2 Software

In the course of our research, we utilized a range of different tools to conduct experiments across various project phases. Python[4] emerged as the dominant programming language throughout the project. For pre-processing and post-processing tasks, we relied on Jupyter Notebooks[5] and the JetBrains Dataspell[6] integrated development environment (IDE), leveraging the strengths of each tool to achieve our objectives. Notebooks, with their cell-based execution and inline visualizations, proved to be the ideal choice for gaining an overview of the corpus, manipulating or structuring data, and producing superior post-processing in terms of visualizations.

Due to the resource-heavy nature of ASR implementation, testing, and training, only pre-and-post-processing was conducted on low-resource machines. Everything else must be utilized on hardware similar to that described in 4.2.1. While using the setup as shown in Figure 4.2 enabled the runtime of the frameworks, it also offered several challenges in terms of connectivity, general workflow, development, and testing. While remote accessibility through IDEs is possible, it proved useless due to the two-step login setup paired with a virtual private network (VPN) creating environment latency and unstable connectivity. The user interaction was reduced to a Terminal platform with command-line execution and minimalistic capacities using the embedded IDE provided by the UNIX[7] operating system. This has contributed to a more time-consuming process with less dynamic development functionality and overview.

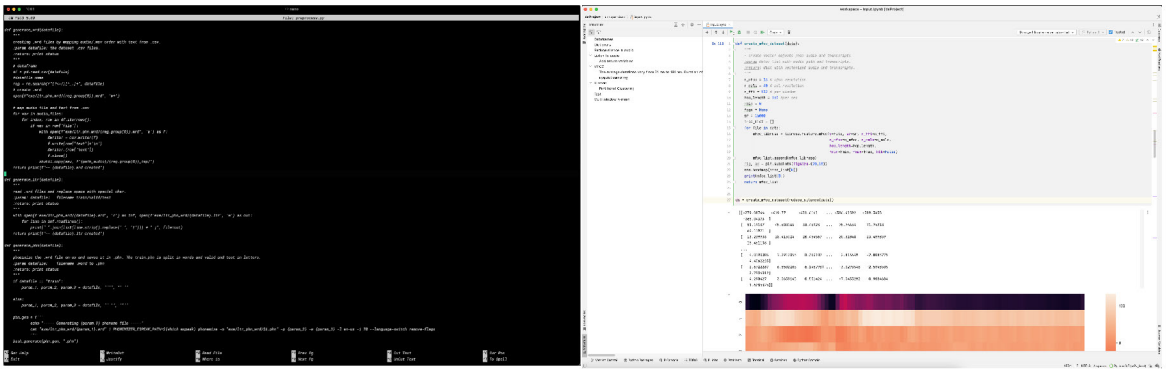


Figure 4.3: Nano and Notebook IDEs interface

For environment and package management, we utilized Anaconda[8] to create enclosed environments for each framework implemented. The dependencies needed in each environment can often be a puzzle to assemble due to poor documentation, versions, and variations in hardware configurations.

4.3 Corpora

Data is the foundation of machine learning, allowing algorithms to learn and adapt to various domains. In supervised learning, labeled data helps models establish relationships between input and output variables, while unsupervised learning relies on data to identify patterns or associations. The quality, quantity, and diversity of the data impact model performance, generalization, and prediction capabilities. Well-crafted machine learning architectures paired with

[4] `https://www.python.org`
[5] `https://jupyter.org`
[6] https://www.jetbrains.com/dataspell/
[7] https://www.opengroup.org/membership/forums/platform/unix
[8] `https://docs.anaconda.com`

few samples or low-quality data will reflect that in the output. While many free high-quality corpora exist, it becomes gradually more difficult to access in domains where labeling costs are high and in the research field with a narrow audience like air traffic control (ATC). This limitation is not present in general automatic speech recognition (ASR) where many options are publicly available.

Furthermore, the scientific community has adopted the large-scale LibriSpeech [68] as the primary benchmark corpus to measure state-of-the-art. It consists of 960 hours of English speech data with corresponding transcriptions. Divided into two sub-corpus, consisting of a clean and noisy version, to facilitate the development of ASR models under various conditions. The clean sub-corpus comprises 460 hours of relatively clean recordings. In contrast, the noisy sub-corpus consists of 500 hours of recordings with more challenging listening conditions due to factors such as varying recording quality, background noise, and speaker variability. The corpora include speech from a diverse range of speakers, with more than 2300 speakers in total, ensuring a variety of accents, speaking styles, and ages. Through results on LibriSpeech indicates the current best-performing research.

General models will not apply to the ATC domain due to their distinctive linguistical characteristics. To address and get an overview of available corpora in the domain-specific area, we created an overview shown in Table 4.1.

Table 4.1: ATC specific corpora.

Corpus	Audio (hours)	Samples	S.rate (KHz)	Published	$	Ref.
Hiwire	28.3	32296	16	2007	50€/3500€	[81]
ATCOSIM	10.6	10078	32	2008	Free	[42]
ATC Complete	70	-	8	1994	1150$	[31]
ATC Communications	20	-	8	2011	Free	[82]

With no defined ATC benchmark corpus and overall limited access to relevant data. We agreed with the client and supervisor to use the ATCOSIM corpus and purchase the Hiwire corpus to create the data platform for the research.

ATCOSIM and Hiwire are two corpora designed explicitly for the research and development of Automatic Speech Recognition (ASR) systems in the Air Traffic Control (ATC) domain. Both corpora focus on ATC communications, providing valuable resources for exploring and developing ASR systems tailored to the unique challenges presented by the ATC environment.

The ATCOSIM corpus comprises clean simulated ATC operator speech. The corpus was created in collaboration with professional air traffic controllers and covered three different English accents: German, Swiss German, and Swiss French. ATCOSIM contains 10.6 hours of speech data, with 8099 ATC commands, and has been manually transcribed to ensure accurate annotations. The corpus is structured to represent various ATC communication scenarios, such as departures, en-route flights, and arrivals. It includes environmental noise and radio interference to emulate real-world control room conditions.

The Hiwire corpus, on the other hand, focuses on cockpit noise and non-native speech. The Hiwire corpus comprises approximately 30 hours of speech data simulated from authentic en-route ATC communications between air traffic controllers and pilots. The corpus covers six different accents: English, Spanish, French, German, Italian, and Greek. A key feature of the Hiwire corpus is that it includes four levels of realistic environmental noise and radio communication interferences, such as co-channel speech, which are inherent to ATC communications. The cor-

pus has been annotated with transcriptions, and the transcriptions have been time-aligned to facilitate the evaluation of ASR systems. In addition to the speech data, Hiwire also provides valuable metadata, such as aircraft types, flight levels, and positions, which can be leveraged to enrich the context for ASR systems.

In comparing the characteristics of the ATCOSIM and Hiwire corpora, several key differences and similarities can be observed. Both corpora focus on ATC communications and have non-native English-speaking operators/pilots catering to a diverse range of English accents, which reflects the global nature of air traffic control.

However, Hiwire differentiates in the number of accents represented, corpus size, and focus on environmental noise. The four different levels of infused noise contain identical communications and transcripts. Even though the corpus is larger than ATCOSIM, it contains the same sentence compositions multiple times. The corpora also differentiate in sequence length and word distributions, which indicates the complexity of the data.

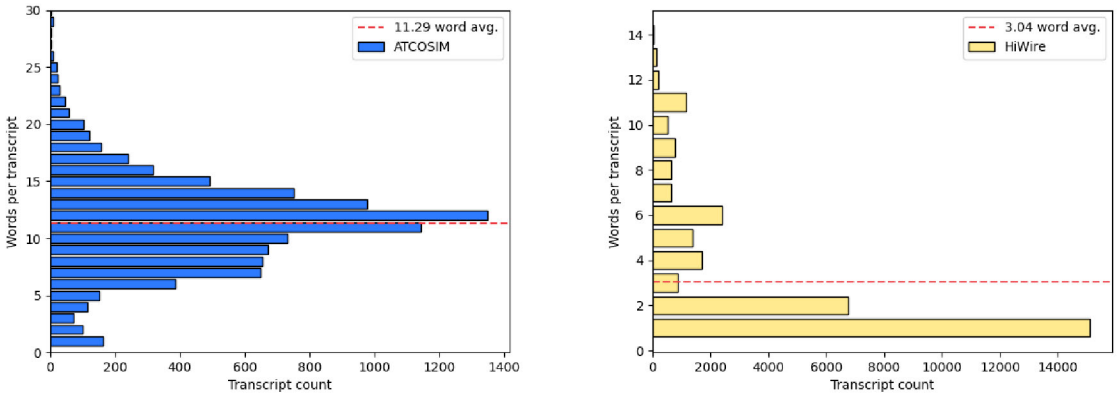


Figure 4.4: Word distribution per transcript in ATCOSIM and Hiwire corpora

The ATCOSIM corpus exhibits an average word distribution of 11.29 words per transcript, while the Hiwire corpus contains only 3.04 words per transcript, as illustrated in Figure 4.4. The word distribution ratio is approximate $\frac{11.29}{3.04} \approx 3.71$, indicating that the ATCOSIM corpus has about 3.71 times more words per transcript in comparison to the Hiwire corpus. The disparity between the two corpora is further demonstrated in Figure 4.5, where an average ATCOSIM sentence contains 63.67 words, and the Hiwire corpus averages 16.56 words.

The word distribution ratio, which is approximate $\frac{63.67}{16.56} \approx 3.84$, suggests that the average word in the ATCOSIM corpus is longer than in the Hiwire corpus. ASR data complexity can be classified into two main aspects: audio quality, which pertains to sampling rate and noise, and content complexity, which refers to the number of words spoken in a sequence.

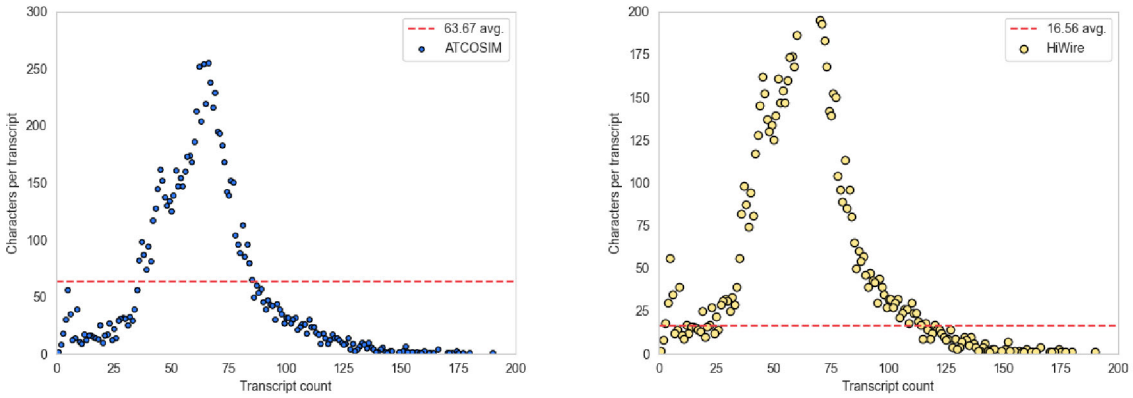


Figure 4.5: Characters distribution per transcript in ATCOSIM and Hiwire corpora

Both corpora possess distinct characteristics, advantages, and constraints. The emulated data should ideally be substituted with actual ATC data for research purposes; however, due to restricted accessibility, this was not an option.

4.3.1 Preprocessing the corpora

To attain a consistent representation of the corpora's audio and transcripts, we implemented a systematic pre-processing procedure to harmonize the sample rate of the audio files and achieve equivalent character representations across both sets. Moreover, we devised a unified file structure, incorporating essential metadata while maintaining references to the associated audio files. This approach ensures congruence in the representation of the speech data and facilitates efficient management and streamlined access to the corpora during subsequent stages of analysis and modeling.

4.3.1.1 Structure

To effectively manipulate and analyze the information contained within the two corpora, it was essential to establish a coherent and streamlined structure that would promote seamless data integration throughout the project's duration. Figure 4.6 depicts the initial architecture of the source data upon acquisition.

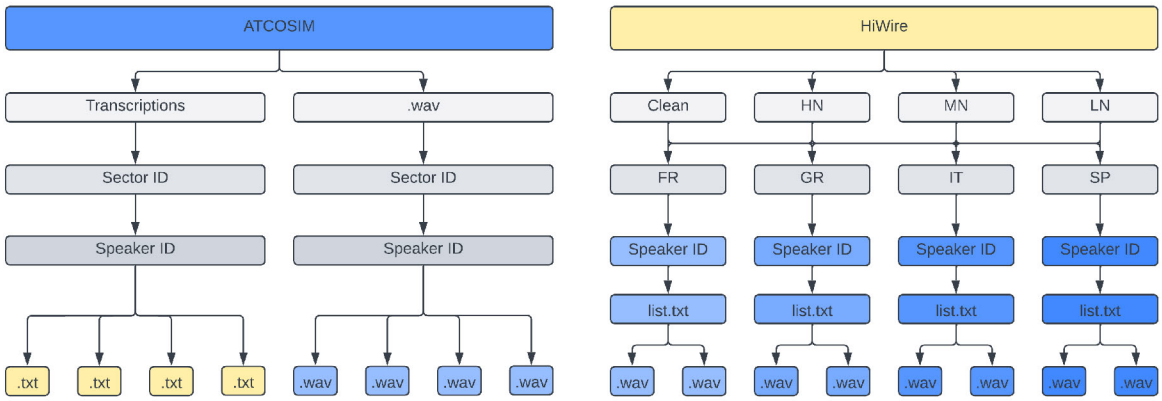


Figure 4.6: Corpus original structure

Streamlining the data architecture and facilitating its integration into Python[9] ecosystems, we consolidated the information into a singular dataframe[10] comprising audio mappings and pertinent features relevant to the project's objectives. Additionally, we extracted the Waveform Audio File (WAV) and structured them within a unified directory, illustrated in Figure 4.7.

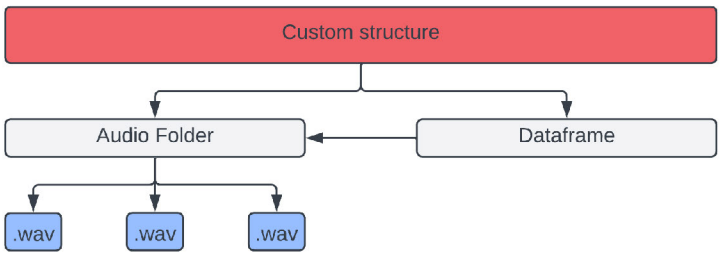


Figure 4.7: Custom structure

The data frame adheres to the Boyce-Codd Normal Form (BCNF) principles, as outlined in the seminal paper by Boyce and Codd [22]. This compliance facilitates seamless integration with a database infrastructure, potentially benefiting future data collection, feature enhancement, or dynamic model training. The corpus comprises $\approx$ 40,000 records, which are organized in four columns: file path, transcript, corpus, and sub-corpus.

4.3.1.2 Audio preprocessing

The Waveform Audio Files (WAV) supplied in the two corpora exhibit sampling rates of 16 and 32 kHz, respectively. To ensure uniformity and facilitate the analysis of these discrete-time signals, it is necessary to harmonize their sampling rates. This can be achieved by employing either upsampling or downsampling techniques. While there is no universally "best" sampling rate, 16 kHz is often considered a good choice for ASR for several reasons.

In the Nyquist-Shannon sampling theorem, a signal can be reconstructed without loss of information if it is sampled at a rate at least twice the highest frequency of interest [64]. Most speech energy lies below 4 kHz, and the human auditory system is most sensitive to frequencies between 1 kHz and 4 kHz [60]. Therefore, a sampling rate of 8 kHz should be theoretically sufficient to capture the perceptually relevant components of speech. However, in practice, a higher sampling rate, such as 16 kHz, is preferred to avoid aliasing artifacts and provide more accurate spectral resolution, resulting in better ASR performance.

[9] `https://www.python.org`
[10] `https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html`

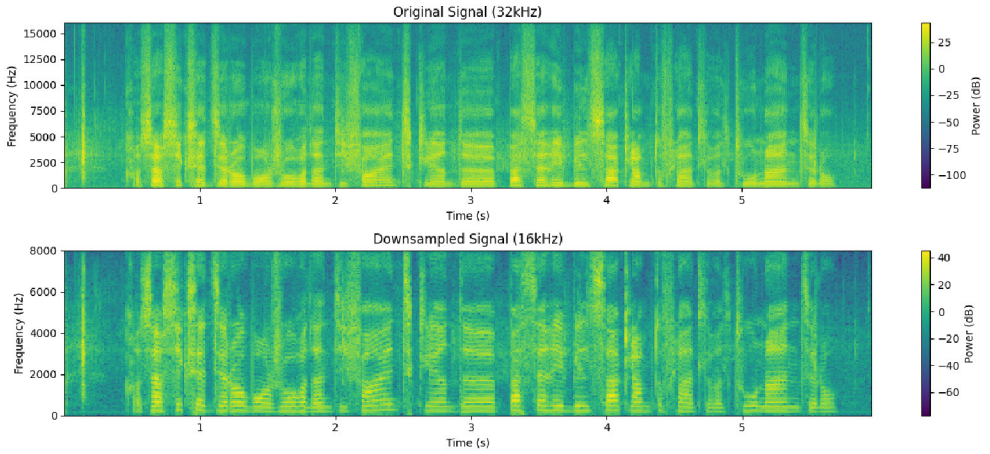


Figure 4.8: Spectrogram illustration of downsampling

In order to downsample the ATCOSIM corpus, we processed the WAV signals represented by the samples $x[n]$ with a sampling rate of 32 kHz. Our goal was to downsample the signal to a sampling rate of 16 kHz. To achieve this, we performed the following steps. First, we use a low-pass filter to remove the high-frequency components from the original signal above 8 kHz (half of the target sampling rate of 16 kHz). This is done to prevent aliasing, which can cause distortions in the downsampled signal. The filter function $H(\omega)$ (Equation 4.1) can be expressed as:

$$H(\omega) = \begin{cases} 1, & \text{if } -\omega_c \leq \omega \leq \omega_c \\ 0, & \text{otherwise} \end{cases} \tag{4.1}$$

Here, ω_c is the cutoff frequency, and the filter will only allow frequencies within the range of $-\omega_c$ to ω_c to pass through.

After filtering, we have a new signal represented by the samples $y[n]$. To downsample, we take every other sample from the filtered signal (Equation 4.2):

$$z[n] = y[2n] \tag{4.2}$$

The resulting signal, $z[m]$, has a sampling rate of 16 kHz. This is best illustrated in Figure 4.8 showing the spectrograms of one ATCOSIM sample at 32 KHz and 16 KHz.

4.3.1.3 Text preprocessing

In order to achieve consistency and uniformity in text sequence input representation, all corpus transcriptions were subjected to pre-processing. This process entailed converting all alphabetic characters to lowercase and removing any special characters not belonging to the alphabet. This measure effectively reduced the unique character count for the Hiwire transcripts, which initially comprised uppercase, lowercase, and special characters. Consequently, this reduction impacts the training cost and the output probability distribution at the output layer. Furthermore, samples containing missing audio files or empty transcriptions were omitted from the corpus to avert data corruption and input uncertainty.

Subsequently, the new custom structure consists of a pre-processed collection of ATCOSIM and Hiwire data. To get comparable results across all models, we partitioned the custom corpus into

training, validation, and testing sets at an overall level and a sublevel. That means we can work with the data as a combined unit or separately as ATCOSIM and Hiwire. We used the split key of 70%, 20%, and 10% distributions, respectively. The distribution resulted in an overall split of 29299/8371/4185, ATCOSIM sub-split of 6677/1907/954, Hiwire sub-split of 22622/6463/3232, and each of Hiwire clean/LN/MN/HN with 5655/1616/808 as shown in Figure 4.9. The samples were selected randomly to ensure a fair representation of the data. This partitioning strategy allowed us to establish a robust evaluation framework across all the implemented models.

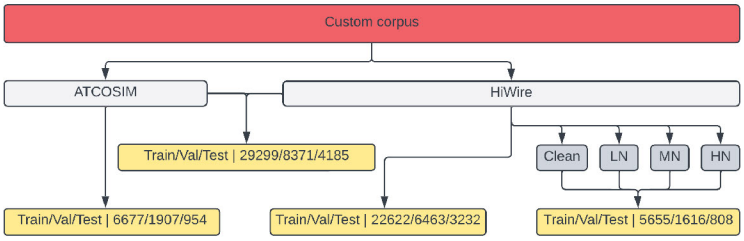


Figure 4.9: Train/validation/test partition

4.4 Deep speech 2

In order to deepen our understanding of ASR technology and gain practical insights, we implemented an architecture paralleling the characteristics of Deep Speech 2, one of the pivotal end-to-end architectures employed in the transition from HMMs to DNNs as described in Section 3.2.1. The informative guide provided by the official Keras website [15] served as a vital resource.

A vocabulary array is initially constructed during the transcript preprocessing stage, encompassing all distinct characters in the corpus transcript. In our case, the vocabulary size amounts to 30 units, including 26 alphabetical letters, numerical digits 1, 2, and 3, and a whitespace character. These units are then mapped to unique integers that function as the models' output.

The initial script incorporates the STFT as the feature representation for the CNN layer in the model. However, considering the demonstrated efficacy of Mel-scaled spectrograms and MFCCs in capturing significant acoustic characteristics in speech data, we have included these alternative features as potential candidates for the final training phase of the model. Given that the sample rate is set at 16 000, in order to keep within the recommended window size of 20-25ms and step size of 10ms for ASR, these parameters do not exceed 400 and 160 (Equation 4.3) in any of the training runs for this model.

$$\begin{aligned} \text{window size} &= \left(\frac{400}{16000}\right) \cdot 1000 \\ &= 25ms \end{aligned} \tag{4.3}$$

$$\begin{aligned} \text{step size} &= \left(\frac{160}{16000}\right) \cdot 1000 \\ &= 10ms \end{aligned}$$

As depicted in Figure 4.10, this model adheres closely to the original Deep Speech 2 architecture. The architecture comprises two CNN layers, each having 32 filters. The initial layer employs a stride of 2, yielding a feature map that is half the dimension of the original. This halved feature map is then used as the input to the second CNN layer. The model's RNN segment incorporates five bidirectional layers, each having 1024 units. The output of the RNN is then fed into a dense layer, the unit count of which mirrors that of a single RNN layer (1024).

The output layer of the model has a dimension of 31 and uses the softmax activation function to predict an output from a single time series step input. The potential output is a token drawn from the pre-defined vocabulary (containing 30 classes) or the special "blank" token. This last token is utilized by the CTC loss function, described in Section 2.4.2, to represent silence or an additional special character outside the established vocabulary.

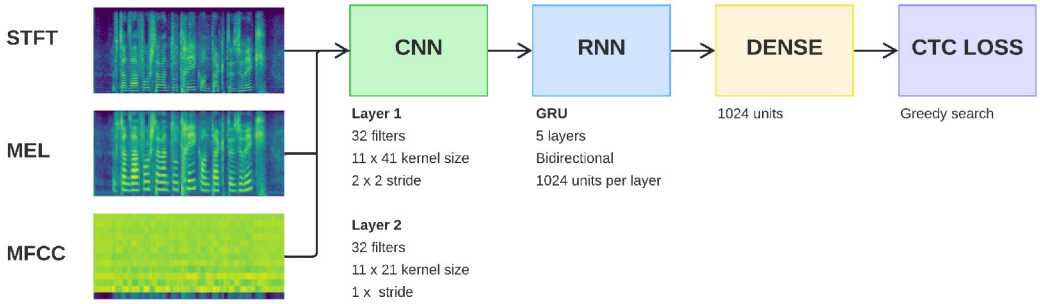


Figure 4.10: Overview of the DNN and its main components.

Alignment and decoding are the two primary functions the CTC loss serves. Alignment involves mapping the sequence of predicted labels produced by the model for each time step to the sequence of true labels, or ground truth transcript, without needing a pre-defined alignment. This is crucial in ASR, where the exact correspondence between audio features and textual characters is typically unknown.

During the training phase, the CTC loss function plays a pivotal role. It contrasts the predicted sequence against the ground truth transcript, in this case, "lufthansa". In order to compute the loss, the algorithm computes the total probability of all possible alignments between the sequence generated by the model and the ground truth. This computation accounts for the reduction of 'blank' labels and the combining of identical sequential characters, thereby enabling the model's output sequence "——lluffthaannsaaaa" to be compacted to align with the ground truth, "lufthansa" (Figure 4.11).

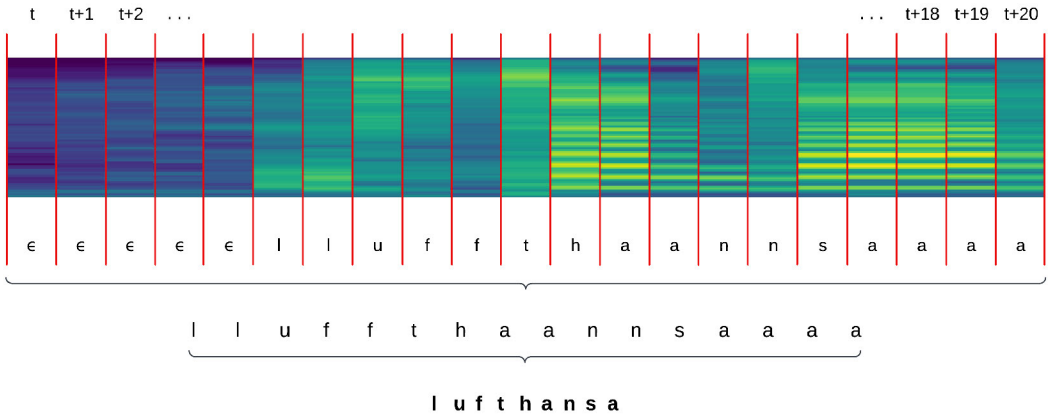


Figure 4.11: CTC decoding the word "Lufthansa" using greedy search with Mel spectrogram as features.

At the commencement of training, the ASR model tends to generate sequences far from the intended target. For example, the model might initially generate a sequence such as "—-hhhanccbbaasaa" in response to the mel-spectrogram of "lufthansa". The chief objective of the training process is to direct this model toward generating coherent and accurate predictions. The model is presented with pairs of Mel spectrogram inputs and their corresponding ground truth transcriptions during training. After each input is processed, the model generates a predicted output sequence. It is at this juncture that the CTC loss function is invoked.

This function takes as input the ground truth, and the model's current predicted sequence then computes the total probability of all possible alignments between these sequences, facilitated by the removal of 'blank' labels and merging repeating characters. The loss function further measures the divergence between the most probable alignment and the ground truth, employing the negative logarithm of the conditional probability of the ground truth given the predicted sequence. This procedure yields a scalar value representing the current prediction's error.

An essential phase in refining the model's predictive capabilities involves minimizing this loss. This is accomplished by backpropagating the error information through the model, leading to an adjustment in its parameters. This iterative process incrementally enhances the model's transcription accuracy. Consequently, over a period of time and given adequate training data, the model's outputs gradually transition from initially seeming nonsensical to sequences that closely mirror meaningful utterances like "lufthansa".

Once the model has been fully trained, it enters the inference phase, wherein it interprets an STFT spectrogram, mel-spectrogram, or MFCC input to generate a sequence that may include 'blank' labels. Decoding techniques such as greedy (best path) decoding or beam search transform this sequence into a legible transcript.

Greedy decoding involves selecting the label with the highest probability at each timestep (Figure 4.12), followed by the 'blank' labels being removed and any recurring characters being merged. Beam search decoding maintains a track of multiple hypotheses at each timestep. It explores extending each hypothesis with every possible label and retains only the top few based on their total probabilities. Once a specific condition is met, such as reaching the end of the sequence or when all active hypotheses surpass a specific score, the hypothesis with the highest overall probability is selected.

While Beam search decoding is generally more computationally intensive, it often yields more accurate results, particularly for longer sequences or when the model's confidence in its pre-

dictions is relatively low. Conversely, greedy decoding is more computationally efficient and straightforward, making it a suitable choice when speed and simplicity need to be prioritized, especially when dealing with shorter sequences where the disparity in accuracy between the two methods may be insignificant. Given the type of corpora utilized in this study, both containing short audio samples and with transcripts averaging three words for Hiwire and eleven words for ATCOSIM (Section 4.3), the added computational cost and complexity of Beam search decoding might not significantly enhance performance, and when considering time constraints and the shared resources available to us, the greedy search decoding was chosen.

	t	t+1	t+2	t+3	t+4	t+5	t+6	t+7	t+8	t+9	t+10	t+11	t+12	t+13	t+14	t+15	t+16	t+17	t+18	t+19	t+20
ε	0.72	0.77	0.89	0.73	0.81	0.01	0.0	0.04	0.06	0.0	0.04	0.05	0.05	0.07	0.02	0.01	0.0	0.04	0.0	0.05	0.05
_	0.03	0.06	0.04	0.0	0.01	0.01	0.0	0.02	0.03	0.04	0.01	0.03	0.01	0.04	0.01	0.02	0.05	0.02	0.06	0.05	0.05
a	0.05	0.06	0.02	0.03	0.07	0.04	0.06	0.05	0.05	0.01	0.04	0.04	0.72	0.64	0.0	0.05	0.06	0.6	0.79	0.88	0.89
b	0.02	0.01	0.06	0.03	0.06	0.06	0.06	0.03	0.01	0.05	0.02	0.03	0.03	0.05	0.04	0.02	0.04	0.0	0.05	0.03	0.03
c	0.05	0.02	0.01	0.03	0.02	0.05	0.05	0.01	0.06	0.05	0.01	0.06	0.01	0.06	0.02	0.05	0.03	0.04	0.02	0.01	0.05
d	0.05	0.02	0.01	0.03	0.0	0.02	0.0	0.04	0.01	0.04	0.03	0.03	0.04	0.03	0.04	0.05	0.0	0.05	0.02	0.02	0.05
e	0.05	0.06	0.06	0.04	0.04	0.01	0.03	0.04	0.06	0.04	0.01	0.01	0.03	0.06	0.02	0.02	0.06	0.03	0.02	0.02	0.02
f	0.03	0.04	0.04	0.05	0.01	0.04	0.01	0.06	0.89	0.67	0.02	0.06	0.01	0.02	0.01	0.03	0.0	0.01	0.05	0.04	0.05
g	0.03	0.02	0.03	0.06	0.06	0.05	0.05	0.05	0.03	0.03	0.02	0.04	0.03	0.04	0.0	0.05	0.01	0.02	0.05	0.02	0.03
h	0.04	0.06	0.05	0.01	0.01	0.0	0.06	0.02	0.01	0.04	0.03	0.9	0.02	0.02	0.04	0.05	0.04	0.05	0.07	0.05	0.05
i	0.02	0.0	0.0	0.05	0.03	0.07	0.05	0.05	0.05	0.06	0.05	0.01	0.02	0.05	0.05	0.01	0.01	0.03	0.05	0.04	0.01
j	0.01	0.02	0.0	0.05	0.01	0.01	0.03	0.01	0.04	0.07	0.05	0.06	0.02	0.01	0.04	0.0	0.04	0.01	0.0	0.05	0.04
k	0.05	0.02	0.03	0.01	0.05	0.01	0.05	0.05	0.06	0.0	0.04	0.03	0.05	0.02	0.04	0.03	0.06	0.01	0.02	0.02	0.04
l	0.05	0.01	0.01	0.06	0.01	0.74	0.66	0.02	0.02	0.07	0.04	0.05	0.03	0.01	0.06	0.03	0.04	0.06	0.06	0.02	0.05
m	0.01	0.06	0.02	0.06	0.04	0.06	0.02	0.01	0.02	0.01	0.05	0.05	0.01	0.03	0.03	0.05	0.01	0.05	0.07	0.02	0.02
n	0.02	0.0	0.04	0.01	0.04	0.04	0.04	0.02	0.06	0.03	0.05	0.01	0.06	0.01	0.82	0.72	0.06	0.03	0.02	0.05	0.05
o	0.04	0.06	0.05	0.0	0.04	0.01	0.05	0.01	0.01	0.05	0.05	0.03	0.02	0.04	0.01	0.04	0.02	0.05	0.07	0.01	0.05
p	0.04	0.01	0.04	0.03	0.01	0.03	0.02	0.03	0.04	0.05	0.04	0.02	0.06	0.01	0.06	0.01	0.0	0.01	0.01	0.05	0.03
q	0.05	0.0	0.05	0.01	0.01	0.04	0.05	0.05	0.06	0.06	0.05	0.02	0.06	0.02	0.03	0.02	0.01	0.05	0.02	0.05	0.02
r	0.01	0.03	0.05	0.06	0.06	0.0	0.04	0.02	0.05	0.02	0.01	0.01	0.04	0.02	0.06	0.06	0.03	0.03	0.05	0.04	0.03
s	0.05	0.04	0.0	0.06	0.03	0.06	0.01	0.03	0.01	0.04	0.01	0.02	0.01	0.05	0.06	0.03	0.85	0.02	0.01	0.03	0.05
t	0.03	0.03	0.04	0.01	0.03	0.03	0.02	0.01	0.06	0.04	0.76	0.04	0.03	0.0	0.05	0.02	0.03	0.05	0.03	0.04	0.0
u	0.02	0.01	0.04	0.01	0.05	0.06	0.03	0.85	0.0	0.0	0.05	0.01	0.02	0.03	0.02	0.06	0.06	0.03	0.03	0.04	0.0
v	0.05	0.04	0.04	0.01	0.04	0.05	0.03	0.03	0.01	0.0	0.04	0.05	0.03	0.03	0.06	0.04	0.06	0.01	0.04	0.0	0.02
w	0.05	0.02	0.01	0.07	0.06	0.04	0.03	0.02	0.02	0.01	0.02	0.01	0.06	0.03	0.06	0.03	0.05	0.02	0.04	0.04	0.01
x	0.02	0.03	0.05	0.01	0.0	0.0	0.0	0.06	0.01	0.02	0.01	0.03	0.05	0.02	0.05	0.05	0.02	0.0	0.04	0.04	0.04
y	0.02	0.02	0.03	0.03	0.01	0.02	0.05	0.04	0.0	0.06	0.01	0.04	0.02	0.06	0.03	0.0	0.01	0.05	0.04	0.04	0.02
z	0.0	0.06	0.05	0.06	0.07	0.03	0.03	0.05	0.05	0.04	0.05	0.06	0.04	0.02	0.01	0.04	0.06	0.0	0.0	0.0	0.03
1	0.01	0.06	0.06	0.02	0.0	0.04	0.01	0.05	0.0	0.02	0.03	0.02	0.02	0.01	0.06	0.02	0.04	0.04	0.0	0.05	0.03
2	0.02	0.04	0.05	0.03	0.01	0.01	0.02	0.02	0.04	0.03	0.04	0.04	0.04	0.05	0.01	0.06	0.0	0.06	0.02	0.02	0.05
3	0.05	0.04	0.02	0.04	0.06	0.04	0.05	0.05	0.05	0.01	0.01	0.03	0.03	0.01	0.02	0.02	0.01	0.05	0.03	0.04	0.02

Figure 4.12: Greedy decoding of the word "Lufthansa" from simulated softmax outputs.

In this model, the *keras.backend.ctc_batch_cost function*[11] is utilized. Despite sparse documentation, it is established that this function applies the CTC loss algorithm to each batch element by taking the following four tensor inputs:

- y_true, encompassing the ground truth labels.
- y_pred, representing the predictions or softmax outputs.
- input_length, indicating the sequence length for each batch element in y_pred.

[11] `https://www.tensorflow.org/versions/r1.15/api_docs/python/tf/keras/backend/ctc_batch_cost`

- label_length, storing the sequence length for each batch element in y_true.

Before training can commence, the corpora must be processed to fit the TensorFlow framework. Each audio and corresponding transcript sample is passed through a function producing a tensor representing the selected feature and a tokenized version of the transcript. This tensor is subsequently fed into the model as the input layer, with a dimensionality equivalent to half the total FFT bins plus one. For instance, if the number of FFT bins is predetermined as 512, the input dimension would be $\frac{512}{2} + 1 = 257$. For the mel-scale features, the input dimension matches the number of mel-bands (60, 70, 80). In the case of MFCCs, the input size depends on including delta and delta-delta coefficients, resulting in an input dimension of either 13 or 39 (13 MFCCs, 13 delta coefficients, 13 delta-delta coefficients).

In the initial stages of exploring the model performance using these three features, we noticed a clear difference in the time each training pass would take. In Table 4.2, using a $\approx$ 6-second audio sample, the difference in the amount of data used for gaining roughly the same result becomes more apparent, and also why MFCCs as feature extraction is preferable, even with delta and delta-delta coefficients added, as its cost is vastly less than that of waveforms or STFT. Still, as an experiment, we continued to use both STFT and Mel-bands as features to compare WER in the initial phase of training the model configurations.

Table 4.2: Deep Speech 2, feature size comparision

Type	FFT bins	Mel-bands	Array shape	Values total
Waveform	.	.	(92884, 1)	92 884
STFT	512	.	(257, 581)	149 317
-	1024	.	(513, 581)	298 053
-	2048	.	(1025, 581)	595 525
Mel	.	60	(60, 581)	34 860
-	.	70	(70, 581)	40 670
-	.	80	(80, 581)	46 480
MFCC	.	.	(13, 581)	7 553
-	.	.	(39, 581)	22 659

In the preliminary trials aimed at identifying the most optimal features, the number of filters and units remained consistent with those detailed in the exemplary script. Hyperparameter tuning involved adjusting window and step size, the number of FFT bins, mel-bands, batch sizes, and dropout rates. These variables were tested in various configurations in a training loop, each lasting for 50 epochs, conducted over two weeks commencing in late January 2023. The top 5 performing configurations of each feature extraction method are listed in Tables 4.3, 4.4 and 4.5.

Table 4.3: Deep Speech 2, STFT as features

corpus	batch size	dropout	w_size	step	FFT bins	WER (test)
ATCOSIM	50	65%	400	160	512	29.4%
-	32	-	-	-	-	33.3%
-	50	-	-	-	1024	99.9%
-	4	80%	-	-	512	100%
-	-	50%	-	-	-	100%
Hiwire clean	50	65%	400	160	512	24.1%
-	32	-	-	-	-	28.6%
-	50	-	-	-	1024	30.0%
-	32	-	256	128	512	38.2%
-	-	-	400	160	1024	51.7%

Table 4.4: Deep Speech 2, Mel-scale as features

corpus	batch size	dropout	w_size	step	FFT bins	Mel bands	WER (test)
ATCOSIM	50	65%	400	160	1024	60	42.1%
-	-	-	-	-	-	70	64.3%
-	4	70%	-	-	-	60	97.6%
-	32	65%	-	-	-	80	98.3%
-	-	-	-	-	-	60	99.5%
Hiwire clean	50	65%	400	160	1024	60	18.3%
-	-	-	-	-	-	70	20.4%
-	-	-	-	-	-	60	22.6%
-	32	-	-	-	-	80	30.1%
-	-	-	-	-	-	60	34.9%

Table 4.5: Deep Speech 2, MFCC as features

corpus	batch size	dropout	w_size	step	FFT bins	Mel bands	WER (test)
ATCOSIM	4	70%	400	160	1024	70	22.1%
-	50	65%	-	-	-	60	29.6%
-	-	-	-	-	-	80	38.7%
-	4	80%	-	-	-	70	41.3%
-	16	65%	-	-	-	70	44.0%
Hiwire clean	50	65%	400	160	1024	80	11.3%
-	-	-		-	-	60	17.1%
-	32	-		-	-	80	22.7%
-	-	-	256	128	512	70	22.3%
-	-	-	400	160	1024	60	24.9%

In light of the findings gathered during the feature method examination, it was determined that using MFCCs in conjunction with delta and delta-delta coefficients would represent the most viable strategy for the ensuing phases of the Deep Speech 2-inspired model. Accordingly, the model underwent minor modifications from its initial configuration affecting the number of filters in the CNN layers, the kernel size, and RNN units (Figure 4.13).

The final model architecture underwent an extensive training run lasting over two weeks, during which various combinations of batch size, number of epochs, dropout rate, window, step size, number of FFT bins, and number of mel-bands were examined. The two corpora were trained in conjunction and isolation as single-domain-type corpora. Notably, the Hiwire corpus, encompassing sub-domain variations of identical data, received particular emphasis in the training process.

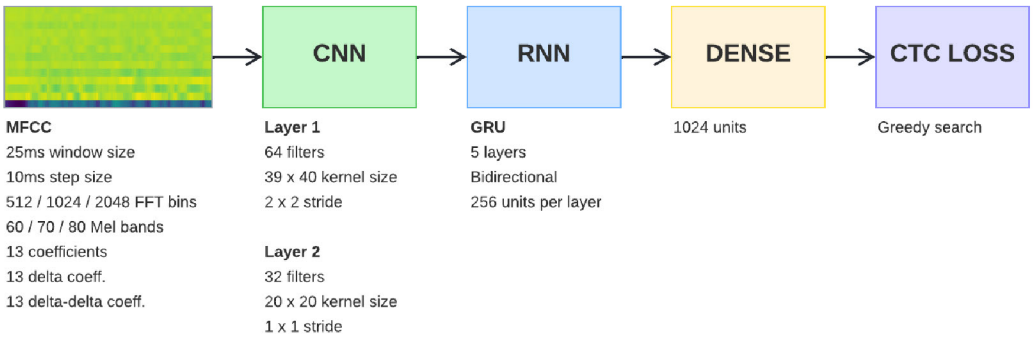


Figure 4.13: The final model configuration.

4.4.1 Post-processing with edit distance

While WER is the most common metric for evaluating the performance of ASR models, it has inherent limitations. Particularly, when deployed on short sentence or solitary word outputs, an error involving a single character propels the error rate to 100 percent, a consequence of notable gravity when such minor discrepancies add up. For instance, consider the scenario where a DS2 model predicts the word 'report' as 'repor'. This minor deviation, albeit a single character, results in a maximal error rate of 100% as per the WER metric.

In order to correct any minor mistakes like the aforementioned made during predictions, the output from the model is processed using the Levenshtein distance, or "edit distance" [52]. It is a metric implemented to quantify the dissimilarity between two strings. It calculates the minimum number of single-character edits (insertions, deletions, or substitutions) needed to transform one string into another.

In the context of ATC, where communication relies on a constrained and particular vocabulary, we have applied this metric for an automatic error correction system using the transcripts as vocabulary and the excellent guide made by Ahmed Fawzy Gad[12]. Our implementation starts by scanning each word in a given transcript to check its presence in the designated vocabulary. Words not found within this vocabulary, perhaps due to transcription errors or variances in pronunciation, are flagged for potential replacement. Subsequently, we compute the Levenshtein distance between the non-corresponding and every word within the controlled vocabulary. This step aims to identify the word in the vocabulary that shares the least "distance" with the non-corresponding word, making it the most probable replacement, as illustrated in Figure 4.14.

[12] `https://blog.paperspace.com/implementing-levenshtein-distance-word-autocomplete-autocorrect`

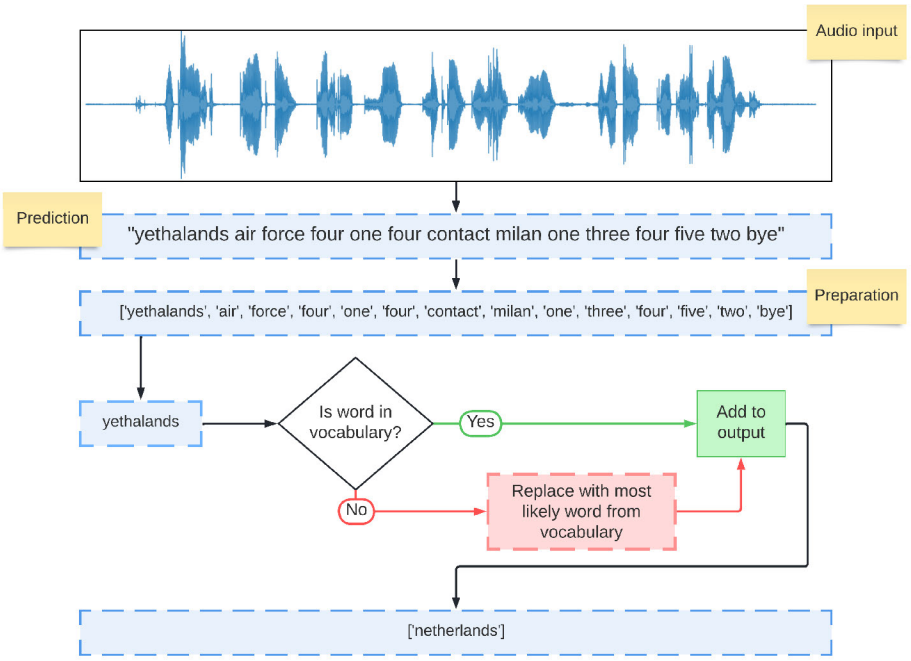


Figure 4.14: Leveraging domain specific vocabulary for post-prediction correction using Levenshtein distance.

4.5 Transformer

In order to get a progressive understanding and hands-on experience of the transformer concept (Section 3.2.2) and observe its impact on automatic speech recognition (ASR), it is paramount to gain insight into the underpinning mechanisms of the transformer (Figure 4.15) due to its pivotal function in the domains of NLP and the ASR processes elucidated in the ensuing methods described in this chapter.

Our implementation is based on the algorithm outlined in the Keras example, "Transformer-based Automatic Speech Recognition" [63], which encapsulates the essential architecture of the transformer combined with a CNN for feature extraction, an approach also referred to as a conformer. In order to incorporate the code, we use the Keras[13] framework, a choice guided by the need for its rich repository of layers and sophisticated toolsets. While Keras supports various back-end engines, it predominantly aligns with the TensorFlow framework[14], providing us with a powerful ecosystem for developing, accessing, and deploying the intended machine learning models.

We chose short-time Fourier transform (STFT) and mel-frequency cepstral coefficients (MFCC) (Section 2.3.1) as standalone inputs for training, to perform a side-by-side comparison of their effectiveness in the Automatic Speech Recognition (ASR) task. STFT's strength lies in generating rich time-frequency representations, thus allowing an intricate understanding of speech dynamics. Conversely, MFCCs, adept at mimicking human auditory perceptions, enable the capture of phonetic subtleties with considerable precision. This strategy of isolating each feature extraction method for separate model training permits an evaluation of their unique performance attributes, leading to a nuanced understanding of their respective utilities under different task conditions and requirements.

[13] `https://keras.io`
[14] https://www.tensorflow.org

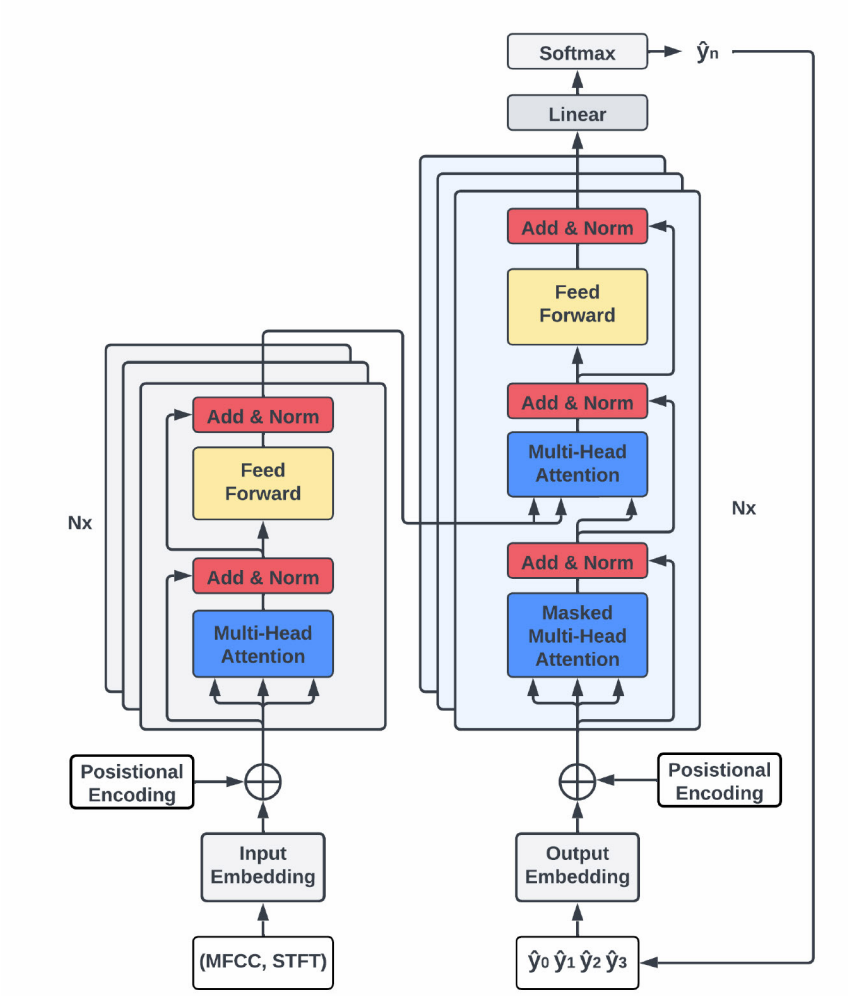


Figure 4.15: The Transformer architecture

Before the input is transformed, each audio file is sampled into timesteps representing a segment of the file. We implement various frame lengths, frame steps, and range of frequency bins to explore the impact on prediction capabilities.

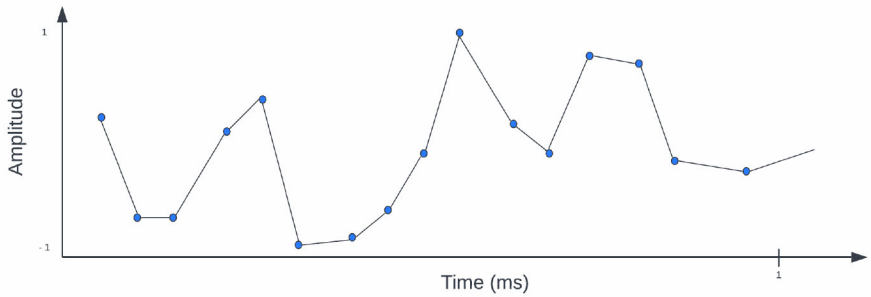


Figure 4.16: 1 millisecond from a 16KHz sample

Each frame is depicted as a matrix, with the number of row elements corresponding to the quantity of sampled values, equivalent to a specific time interval. Considerations of length can either be hardware or model quality concerns. If we want to work with vectors of a specific

length, for example, 16, we can calculate the frame time coverage (Equation 4.4):

$$\begin{aligned} t(s) &= |\vec{v}|/sr \\ t(s) &= 16/16000 \\ t(s) &= 0.001 \end{aligned} \tag{4.4}$$

where t(s) is time in seconds, which is illustrated in Figure 4.16 and sr is the sample rate. Each second of audio contains as many values as the sampling rate. If we, on the other hand, want to define the frame with a specific range of time of 12.5 milliseconds, we can do that by giving us a vector embedding of 200 values for every 12.5 milliseconds (Equation 4.5).

$$\begin{aligned} |\vec{v}| &= sr/t(s) \\ |\vec{v}| &= 16000/0.0125 \\ |\vec{v}| &= 200 \end{aligned} \tag{4.5}$$

In order to create a sliding window throughout the audio file and not miss any vital information, the frame steps must not exceed the defined frame length but, ideally, overlap at shorter intervals.

Before being inserted into the encoder, every frame is subjected to a speech feature embedding process. This entails using 1D convolutional layers (Section 2.4.3), which adeptly transform the speech signals into a series of feature vectors, successfully capturing and preserving essential information from the audio input. Each convolutional layer applies filters that convolve over the input frame, producing intermediate output, often termed feature maps. Using multiple convolutional layers allows the model to learn hierarchical feature representations where the architectural depth enables the identification of more abstract features. Multiple kernel sizes and stride lengths were tested to capture different temporal resolutions in the speech signal. The equation that governs the discrete convolution operation[15] is the sum of vector multiplications when the kernel vector iterates through the input matrix and can be written as follows (Equation 4.6):

$$y[n] = (f * g)[n] = \sum_{-\infty}^{\infty} f[m] \cdot g[n-m] \tag{4.6}$$

Where y[n] is the output of the convolution at the n^{th} position, also referred to as the feature map, f[m] is the input sequence at the m^{th} position, and g[n - m] is the kernel at the $(n-m)^{th}$ position. The process is illustrated in Figure 4.17

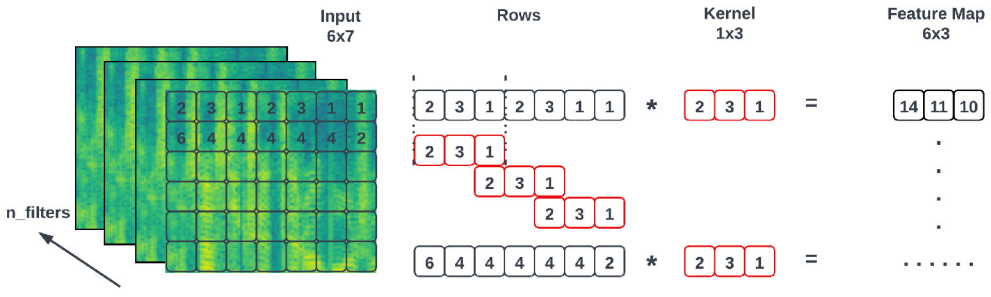


Figure 4.17: 1-D convolution operation

[15] https://en.wikipedia.org/wiki/Convolution

The created feature maps of each frame then undergo a positional embedding to get contextual feature representations of the temporal sequence. The process is illustrated in Figure 4.18.

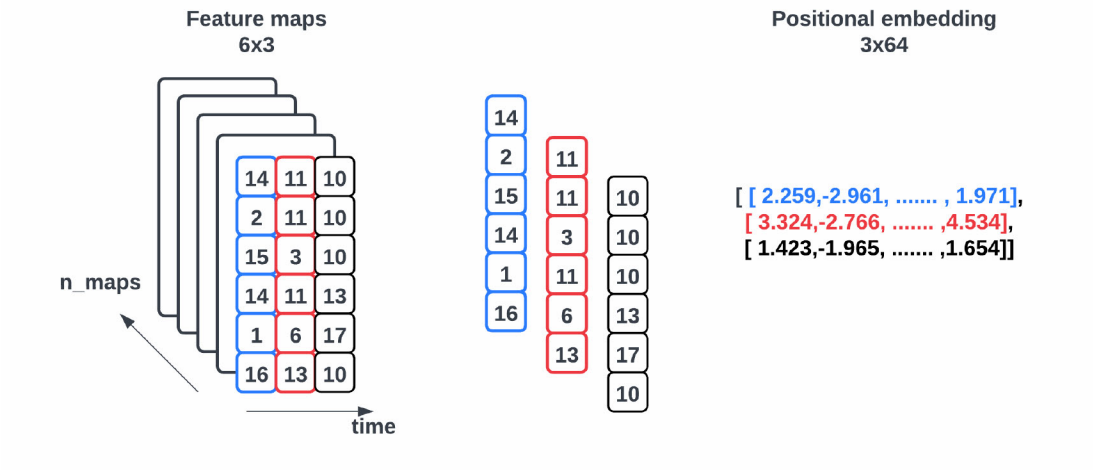


Figure 4.18: Positional embedding vectors

The organized succession of positional feature vectors is conveyed to the Transformer architecture, which provides the initial corpus to the encoder. To be more explicit, these ordered vectors, each encapsulating unique positional information within the sequence, are utilized as the primary input for the encoder, a crucial component of the Transformer system's ability to learn the contextual patterns of the data. This is important because the transformer does not have the recurrent capacities to learn the positional context as LSTM (Section 2.4.1.1) and GRU (Section 2.4.1.2).

The decoder is provided with two inputs. The initial input consists of a sequence of previous predictions ($\hat{y}$), which is then converted into integer tokens utilizing a predefined vocabulary dictionary. Notably, each sequence includes a start token, ensuring that the sequence is not empty even in the absence of prior predictions.

$$\begin{aligned} \text{Output}(\hat{y}) &= \text{<air france ready to takeoff>} \\ \text{Token Embedding} &= [99, 1, 9, 18, 0, 6, 18, 1, 14, 3, 5, 0, 18, 5, 1, 4, 25, 0, 20, 15, 0, 20, 1, 11, 5, 15, ...] \\ \text{Pos. Encoding} &= [[-0.021017, 0.021590, -0.000352], [0.057699, -0.033303, -0.036072]...] \end{aligned}$$

The model employs character-level positional embeddings; however, alternative approaches can instead leverage word, sub-word, or phoneme embeddings. In the example, each integer token corresponding to a character is projected into a dense vector space of fixed dimensionality. These embedded vectors, representing the complex relationships and characteristics of the individual characters, are visually represented in Figure 4.19.

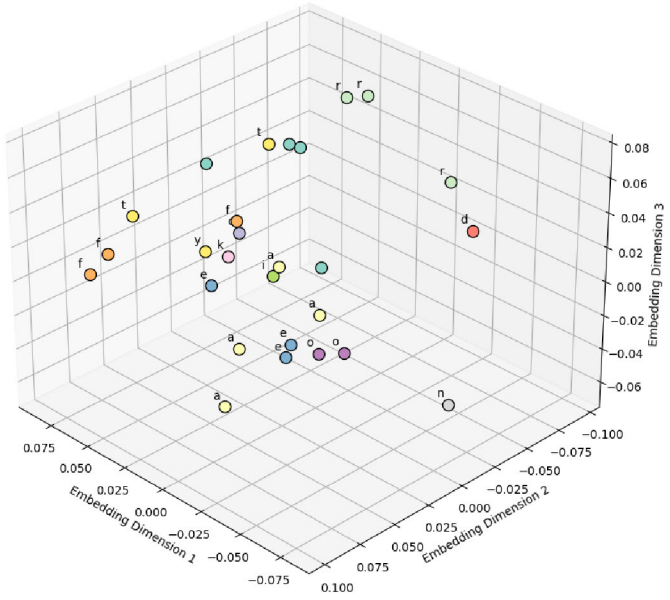


Figure 4.19: 3D plot of the character positional encodings

Additionally, the encoder's output is integrated as an input into the decoder's second multi-head attention layer. This process allows for linking auditory feature tokens with the character tokens predicted earlier.

Both the encoder and decoder architecture consists of multiple stacked layers. Each layer additionally consists of two sub-layers. The first sub-layer is a multi-head self-attention mechanism designed to capture different types of relationships in the data. It allows the model to focus on different parts of the input sequence simultaneously, providing a richer understanding of the context. The mechanism illustrated in Figure 4.20 functions by first partitioning the input data into multiple "heads".

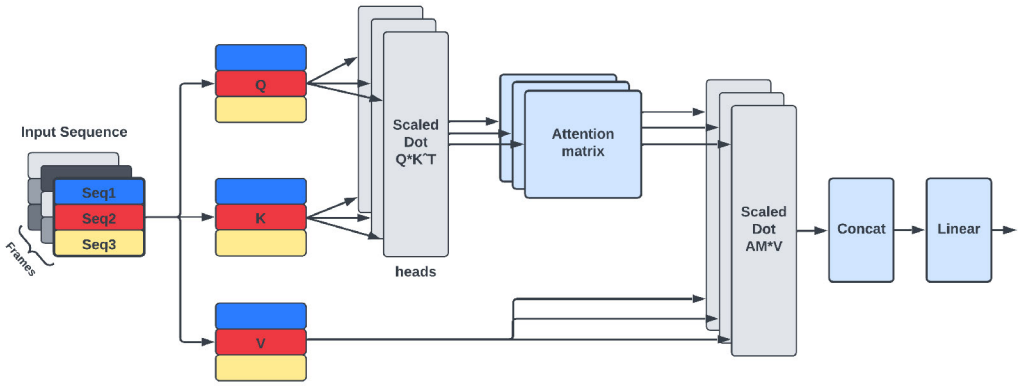


Figure 4.20: Multi-head self-attention

For each head, the input data is linearly transformed into Queries (Q), Keys (K), and Values (V). These transformations are implemented through learned weight matrices, enabling the model to create distinct representations from the same input, thereby facilitating the capture of diverse features or patterns. Queries determine which elements of the input sequence to focus on. The Keys can be seen as an index system helping the model match the right parts of the input sequence to the queries. Once the match between Q and K is determined, V vectors provide the actual information to be used. The better the match, the more influence the corresponding value vector has in the output. The attention process for each head starts by computing the scaled dot product of Q and K^{T}, which determines the compatibility or similarity between Q and K. This value is then scaled down by the square root of the K dimension to prevent large values from dominating the softmax function (Equation 4.7).

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^{\top}}{\sqrt{d_k}}\right)V \tag{4.7}$$

The softmax function applies to these scaled values, converting them into attention scores or weights, which essentially quantify the degree of importance of each value in the sequence and add them to the attention matrix. High scores mean the corresponding tokens are highly relevant to the current processing token. These scores are then used to weigh the corresponding V, producing the output for each head. Finally, the outputs from all heads are concatenated (Equation 4.8) and passed through another linear transformation multiplying it with a learned weight matrix to produce the final output for this sublayer as shown in Figure 4.21.

$$\text{Multi-head}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W_O \tag{4.8}$$

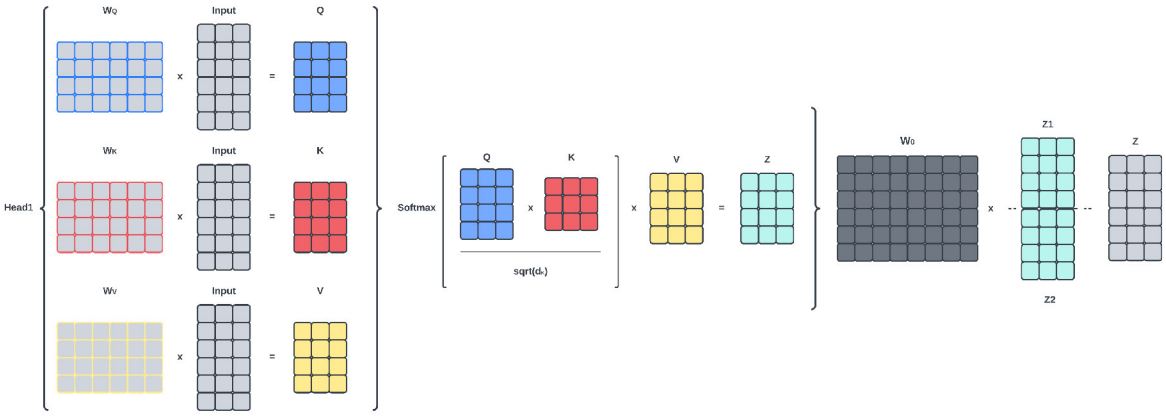


Figure 4.21: Linear transformations throughout a head

The first attention layer of the decoder incorporates a method known as "masking," a sophisticated strategy that deliberately hides specific portions of the input data. Masking is crucial since it guarantees that later stages do not influence the predictions produced at any stage in the sequence. This is a vital precaution that confirms the sequence's predictions depend only on previously known and present information but are unaffected by any details not yet revealed at the prediction point. Essentially, masking maintains the sequential order of the input data. Furthermore, it carefully preserves the time-based relationships between various data points, keeping the natural sequence's timeline intact. Consequently, the masking technique effectively mitigates the potential disturbance caused by future data on present predictions, validating and strengthening the reliability of the sequence generation process within the decoder's architecture.

The secondary sub-layer within the encoder is a fully connected feed-forward network(FFN). While the self-attention mechanism enables understanding the interaction of different parts of the input sequence, the FFN further processes this information, allowing the model to learn more complex and abstract representations. The FFN consists of two fully connected layers using Rectified Linear Unit (ReLU) as the activation function, applying a non-linear operation to the data. The entropy of a single neuron in the FFN dense layers can be written as Equation 4.9:

$$\text{output} = ReLu(\sum(Wi * Xi) + b) \tag{4.9}$$

Where the weighted sum is computed with the dot product of the input(X) and the weight(W), adding the bias(b), ReLU sets all negatively valued neurons to 0. Notably, the FFN is applied independently to each position in the sequence. This allows the model to learn and extract features for each token position separately, enabling the detection of a wide variety of patterns across the sequence. In essence, the FFN aids in processing the information obtained from the self-attention mechanism and helps in learning complex intra-sequence relationships, thereby contributing to the overall effectiveness of the Transformer model. The FNN output is a sequence of the same length as the input. This sequence represents a new, more complex set of features for each token.

Each sublayer incorporates an exclusive characteristic, a residual or bypass connection. This beneficial feature effectively attenuates the gradient vanishing issue during the model's learning process, facilitating the acquisition of intricate representations. Subsequently, the outputs generated from each sublayer, which includes attention mechanisms and feed-forward networks (FFNs), are connected with their respective original inputs, constituting the formation denoted as input(X) + sublayer(input(X)), representative of the residual connection.

An operation known as layer normalization is executed upon establishing this residual connection. This process adjusts and realigns the inputs of the layer to attain a zero mean and unit variance, thus significantly enhancing the stability of the network's training phase. This normalization operation is represented as layernorm(input(X) + sub-layer(input(X)).

The last softmax layer assigns a probability distribution across the classes or tokens such that the total sum of probabilities equals 1 (Equation 4.10). The highest scoring token is the prediction in a given time step.

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{4.10}$$

Where $\vec{z}$ is the input vector, e^{z_i} is the exponential of the i^{th} element in z and e^{z_j} the sum of all elements.

The architectural build of the transformer and its powerful layer compositions and attention capabilities provides a powerful sequence-to-sequence model. To utilize the Transformer architecture at its most basic form and test its capabilities on the ATC domain, we implemented a hyperparameter loop (B.1) conducting continuous training of models at a scheduled timeframe set in Loop 1 (Figure 4.1) or until a human level WER score of is achieved. Human-level scores on the LibriSpeech corpus clean were validated at a WER score of 5.8% on clean data and 12.69% on noisy data by the authors of the Deep Speech 2 paper [4]. We set the threshold to 5.8% WER to alert us if a model's test score surpassed that limit. Furthermore, during the extensive training sessions, all trained models were saved for load or rebuild purposes and safeguarding against data loss. Model parameters and WER scores on validation and test data were continuously written to a CSV file for performance overview and inspection.

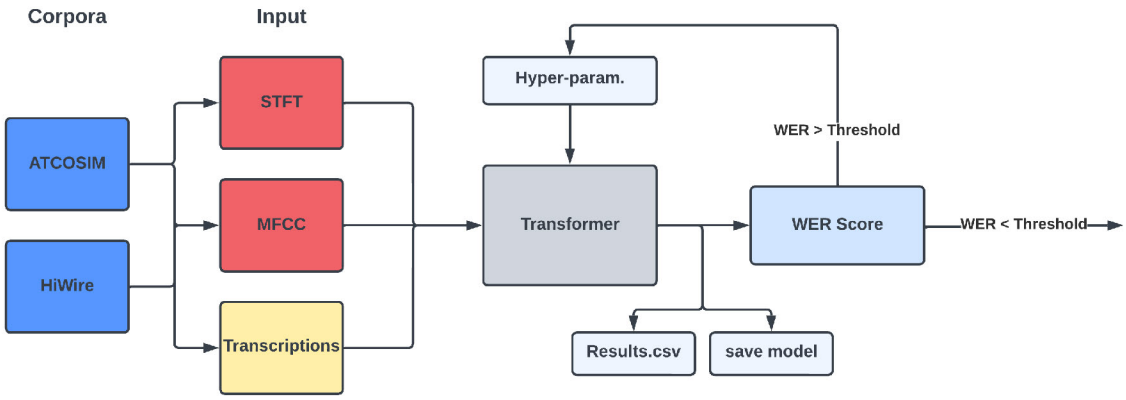


Figure 4.22: Hyper-parameter loop

We can group our parameter focus into four main groups: Input, Convolution, Transformer, and Training. The Input group consists of parameters that deal with the structure of the input data, such as the data source of the input, normalization techniques, and data augmentation methods. The Convolution group refers to parameters within the 1D-convolutional layer applied, which includes the number of filters per layer, kernel size, and stride. The Transformer group revolves around the parameters of the transformer model, which mainly include the number of layers in the encoder and decoder, attention heads, hidden dimensions, and neurons in FFN. Lastly, the Training group encompasses parameters related to the training process, including learning rate, batch size, and number of epochs. Each parameter has been selected to be in the hyperparameter optimization loop due to their impact on the model's performance (Table 4.6).

Table 4.6: Hyperparameters(P)

Group	P1	P2	P3	P4	P5
Input	Corpus	Sub-set	STFT/MFCC	Mel-bins	-
Frame	Length	Step size	-	-	-
Convolution	Kernel size	Step size	-	-	-
Transformer	Enc. layer	Dec. layer	Heads	Head dim.	Neurons (FFN)
Training	Batch size	Epochs	Learning rate	-	-

Each training session uses a "warmup" strategy for adapting the learning rate over time. This approach starts with a minimal learning rate that is increased linearly over 2000 "warmup" steps. After the warmup period, the learning rate decreases in proportion to the inverse square root of the step number. This approach was introduced because it often leads to better training dynamics and model performance. When the weights are randomly initialized at the beginning of training, the gradients can be very large or very small, leading to unstable and inefficient learning. When starting with a small learning rate and gradually increasing it, the model can make minor adjustments initially and avoid getting stuck in local minima.

Throughout the loop, both the optimizer and the loss function remain static. The Adam op-

timizer is used due to its well-known computational benefits and memory efficiency. It is distinct in its adaptive learning rate, which is especially suited for managing large-scale and high-dimensional optimization problems. Meanwhile, the Cross-Categorical Loss function is also employed. This is typically used for multiclass classification tasks, and it offers a reliable and steady way to assess prediction errors by computing the disparity between the model's predictions and the actual transcriptions. Hence, pairing the Adam optimizer and the Cross-Categorical Loss function forms a sound basis for a productive and proficient learning phase during training. The Cross-Categorical Loss function can be described with the Equation 4.11.

$$L = -\sum_{i=1} y_i \log(p_i) \tag{4.11}$$

Where y_i denotes the true label, assigned a value of 1 if true and 0 if false. On the other hand, p_i represents the predicted probabilities associated with each class.

Each unique hyperparameter configuration generated in the loop contributes to a model variant. This iterative exploration of model variants permits us to investigate the potential impacts of these hyperparameters on model performance, guiding us to a more refined understanding of their roles within the ASR task and performance on ATC domain-specific data.

4.6 Wav2vec 2.0

Wav2vec 2.0, developed by Meta AI[16], is an advanced self-supervised speech recognition model. It expands on the foundations of wav2vec, using self-supervised learning techniques to learn useful speech representations directly from raw audio waveforms, thereby removing the necessity for additional tasks during pre-training.

Wav2vec 2.0's architecture, as shown in Figure 4.23, comprises four key components: a feature encoder, a context network, a quantization module, and contrastive loss to connect and regulate the model. The feature encoder and context network are equivalent with the CNN and encoder described in the previous Section 4.5.

Instead of positional embedding used in the original transformer, the model relies on the feature map's relative positional format. Furthermore, a certain percentage of the feature encoder output is masked before passing into the Quantization module and context network. Masking a portion of time steps encourages the model to learn valuable representations that capture masked context based on sequences of unmasked information.

The quantization module act as a bridge between the feature encoder and the contrastive loss. The module consists of a set of codebooks, which can be considered a predefined list or "dictionary" of vectors. The vectors serve as a discrete set of possible representations for the latent feature input. Each feature vector is assigned to a codebook vector where the Gumbel-Softmax allows the module to create a probabilistic, differentiable mapping of all the codebook vectors. The result of the quantization process is a sequence of distributions, which can be interpreted as a sequence of discrete input representations. These assignments are optimized during the training process.

Parallel to the quantization process, the representations from the feature encoder enter the context network. This network employs the encoder model described as part of the transformer in Section 4.5 to capture the temporal context within the sequence of feature representations.

[16] `https://ai.facebook.com`

The contrastive loss function, a critical component of wav2vec 2.0's self-supervised learning strategy, operates on the outputs of the context network and the quantization codebooks. The contrastive loss function is designed to pair the context vectors with the codebook vectors. Here the masking comes into play. The unmasked pairs are mapped and act as the ground truth (positive pairs). The model's task is to map the masked context vector with the "correct" codebook vector. The masked context vector is presented with codebook vectors, where one is the ground truth and the rest is distractors (negative pairs). Through training, the model adjusts its weights based on the ability to identify the correct quantized latent representation. The model will work towards minimizing the distance between the positive pairs and maximizing the distance between negative pairs.

The model's objective function consists of contrastive and diversity loss (Equation 4.12).

$$\mathcal{L} = \mathcal{L}_m + \alpha\mathcal{L}_d \tag{4.12}$$

The contrastive loss function [38] is generalized in Equation 4.13.

$$L(W, Y, \vec{X_1}, \vec{X_2}) = (1 - Y)\frac{1}{2}(D_W)^2 + (Y)\frac{1}{2}\{max(0, m - D_W)\}^2 \tag{4.13}$$

Where W denotes the model parameters, Y represents the binary similarity label between pairs of input samples $\vec{X_1}$ and $\vec{X_2}$. The loss function penalizes the model based on the distances D_W between the input pairs. For similar pairs (Y=1), the squared distance determines the loss. For negative pairs (Y=0), the loss is based on the maximum of zero and the difference between the margin m and the distance. This loss function guides the model to minimize distances for similar pairs.

$\alpha\mathcal{L}_d$ codebook diversity penalty is used if the variation of distractors is small. This discourages the model from focusing on a limited set of codebooks and encourages it to explore a broader variety.

After the self-supervised pre-training phase, wav2vec 2.0 needs to be fine-tuned utilizing labeled data. This process allows the model to effectively leverage the pre-trained generalized weights derived from the pre-training and fine-tune them toward the specific nature of the labeled data. Furthermore, character tokens will be adapted to the sequential patterns learned. The fine-tuning part will be described in Section 4.6.1.

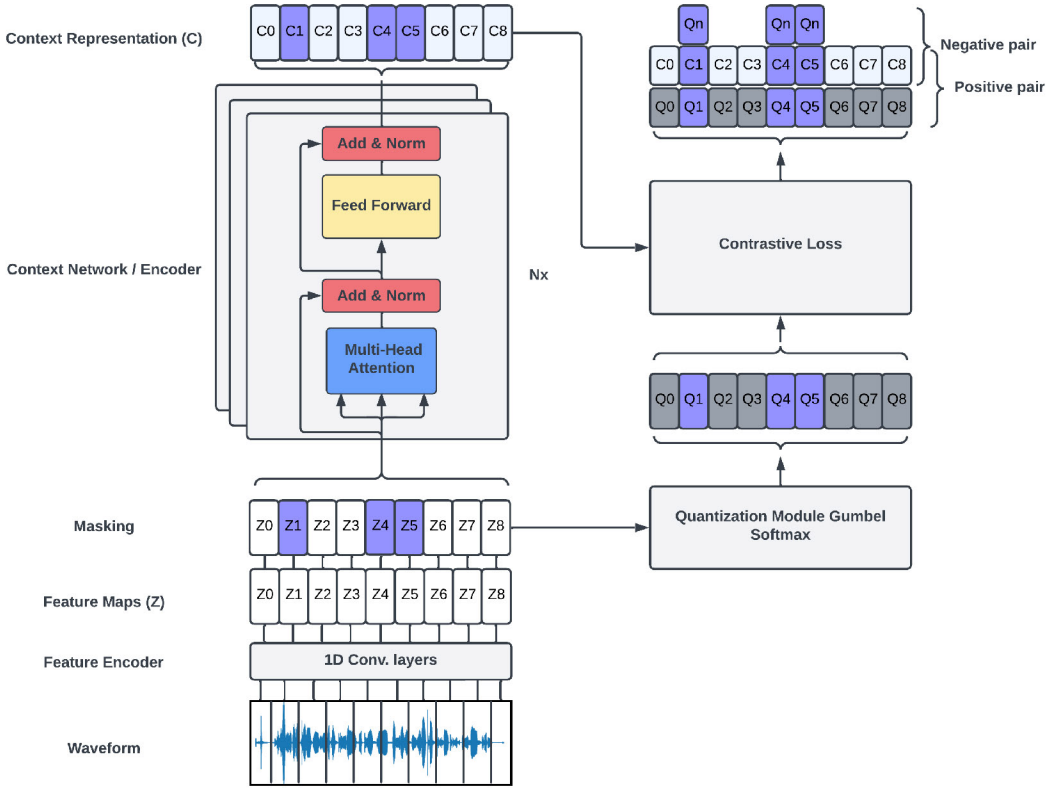


Figure 4.23: Wav2vec 2.0 Architecture

Our initial implementation plan was to utilize the Wav2vec 2.0 framework[17] to create a novel ATC base model, leveraging the benefits of self-supervised learning to capture the underlying patterns and semantics in the ATC domain. Using only a limited annotated corpus, the base model should then be used for downstream airport-specific models adapting to the local tower communications and the operator's linguistic specifications.

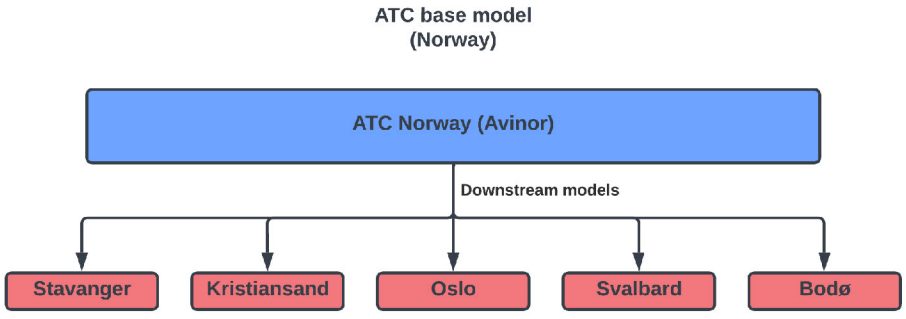


Figure 4.24: ATC base model

The scope of the ATC base is highly dependable on data availability. It can be scaled as needed, provided the magnitude of available audio hours can accommodate substantial size requirements. For perspective, the smallest base model from Meta Research is trained on 53,000 hours of audio, whereas we only have 38.9 hours of ATC data. Even though ATC is highly standardized,

[17] `https://github.com/facebookresearch/fairseq/tree/main/examples/wav2vec`

controllers and pilots must demonstrate the ability to speak and understand English at a certain level. Pronounce and accents can be problematic if a model has not been exposed to similar data. Therefore, it is not plausible that creating a base model will sufficiently cover the ATC domain. Instead, we will utilize Wav2vec 2.0 with different scopes in the following sections.

4.6.1 Hugging Face

Instead of creating an ATC base model downstream, we utilize the secondary step in the Wav2vec 2.0 framework, applying first and second-level fine-tuning on pre-trained models. The implementation was carried out with the Hugging Face framework [1] using the pre-trained models described in Table 4.7.

Table 4.7: Pre-trained base models.

Model	Param.	Pre-trained (hr)	Fine-tuning (hr)	Corpora
Wav2vec2-base	95m	53 000	0	-
Wav2vec2-large-960h-lv60-self	317m	53 000	960	LibriSpeech
Wav2vec2-xls-r-300m	300m	436 000	0	-
wav2vec2-large-robust-ft-swbd-300h	317m	53 000	300	Switchboard

The Wav2vec2-base model was initially pre-trained using 53000 hours of unlabeled data sourced from the Librilight sub-corpus of LibriVox [48] with no fine-tuning. The same base is used in the wav2vec2-large-960h-lv60-self model with an additional level of fine-tuning on 960 hours of data from the LibriSpeech corpus. Next, the Wav2vec2-xls-r-300m model was pre-trained on an impressive 436,000 hours of unlabeled multilingual data obtained from multiple corpora, including CommonVoice [5], VoxPopuli [89], Multilingual LibriSpeech (MLS) [72], VoxLingua107 [87], and the Babel project. Lastly, the Wav2vec2-large-robust-ft-swbd-300h model was pre-trained on a combination of corpora, namely Librilight, CommonVoice, Switchboard [32], and Fisher [21]. It was then fine-tuned using 300 hours of noisy telephone data extracted from the Switchboard corpus.

The models are trained to map large amounts of audio at a base level, clustering the temporal feature patterns within the base corpora. The levels of downstream fine-tuning add specialized capabilities toward the desired ATC domain, as shown in Figure 4.25.

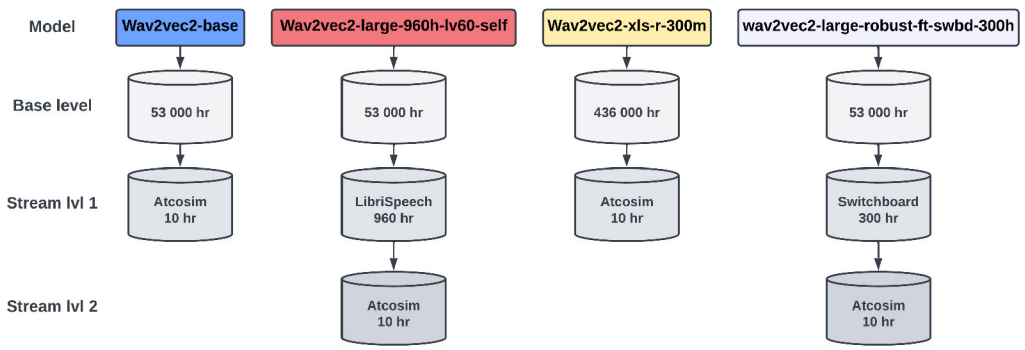


Figure 4.25: Model downstream.

The fine-tuning structure mirrors the pre-training configuration outlined in Section 4.6, with the

exception that the Quantization and contrastive loss modules are replaced by a Connectionist Temporal Classification (CTC) module described in Section 4.4. The weights from the pre-trained base model are imported into the fine-tuning framework, prompting subsequent updates to the convolutional filters, transformer weight matrices, and Feed Forward Networks (FFN). Fine-tuning aims to adapt the pre-trained weights to align with the specific ATC domain and to associate the context vectors with tokens - in this case, characters.

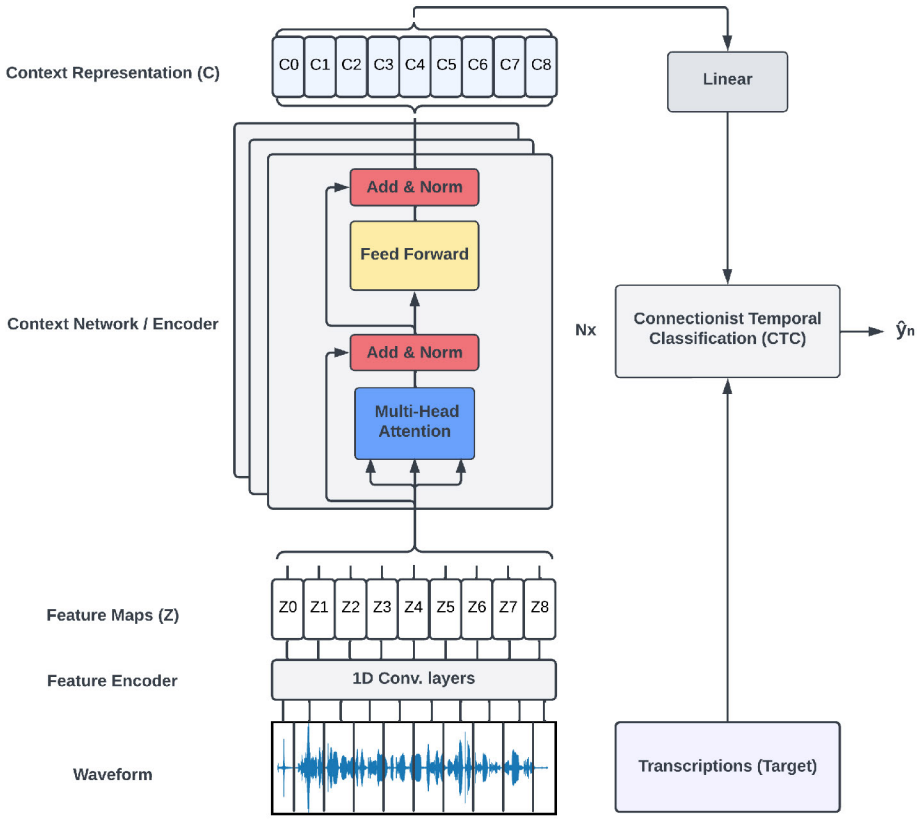


Figure 4.26: Fine-tuning architecture.

Our implementation of wav2vec 2.0 follows a two-step process, divided into a stage before and a stage after establishing the final corpora setup (Section 4.3). Firstly, we initiated the development of a customized version of the ATCOSIM corpus. Our main goal was to enhance the sample size to align with the larger scale observed in other paid corpora. This was accomplished by augmenting three tiers of randomized noise into the original samples, creating three synthetic versions, as shown in Figure 4.27. Consequently, our corpus expanded to encompass 42.4 hours of data, a significant increase from the initial 10.6 hours.

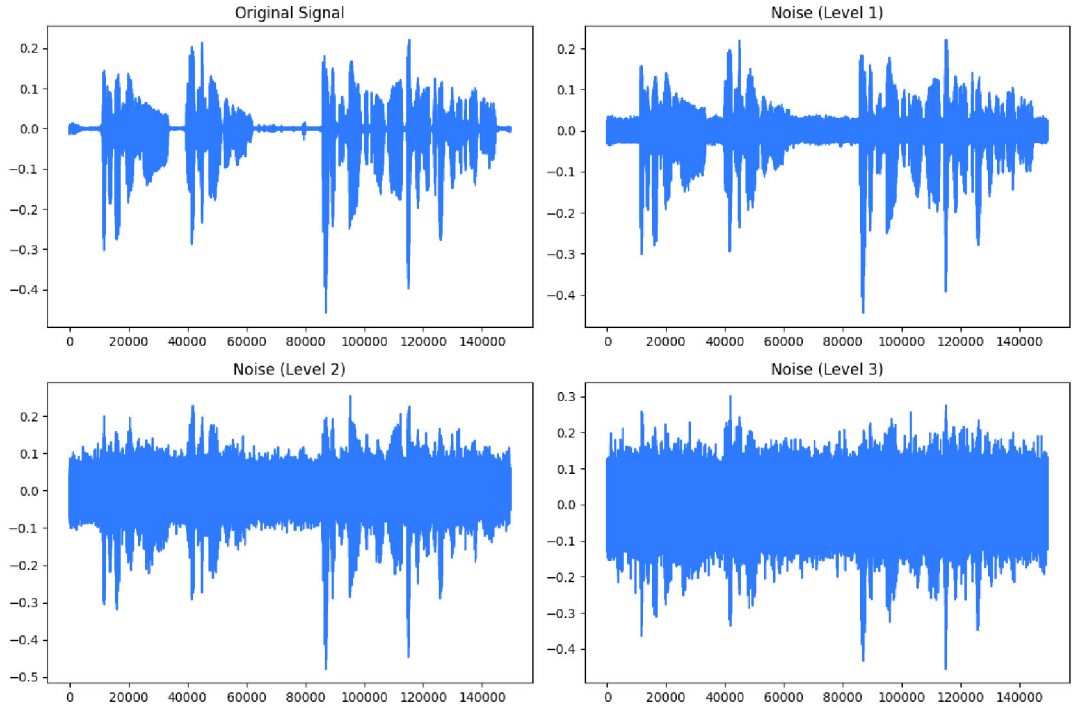


Figure 4.27: Custom corpus noise levels.

We employed a split key 70/20/10 to divide our data into train, validate, and test corpora. Additionally, we conducted tests on the ATCO2 corpus [85] to evaluate the model's predictive capabilities in a non-simulated environment. The ATCO2 corpus contains real ATC data, while our model was trained on simulated ATC data. This allowed us to investigate the model's robustness and ability to make accurate predictions in a cross-reality scenario.

In the second stage of training, we utilized the audio data and corresponding transcriptions as described in Section 4.3. The training process involved multiple cycles of 50 and 100 epochs, employing batch sizes ranging from 8 to 32. Subsequently, we manually tested the best-performing model to explore the potential for further optimization.

4.6.2 Unsupervised

Wav2vec Unsupervised 2.0, also referred to as Wav2vec-U 2.0, introduced in Section 3.4.2 is the most recent ASR initiative by Meta Research's team and is made accessible as open-source through their Fairseq GitHub repository [77]. It is important to note that the project's version v0.12.2, as of May 15, 2023, is still a work in progress. The project's current state can be characterized as experimental, with room for improvement in documentation and code.

The Wav2vec-U 2.0 is an extension of the Wav2vec 2.0 described in Section 4.6 where the fine-tuning 4.6.1 step is replaced with a generative adversarial network (GAN), this in order to train a reliable generator with capabilities to predict sequences of phoneme tokens from context representations and pseudo labels. Figure 4.28 illustrate the overall architectural structure of the unsupervised setup.

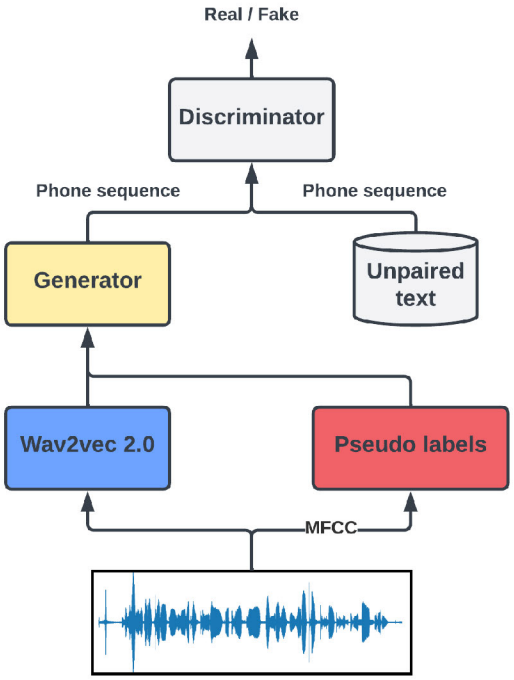


Figure 4.28: Wav2Vec-U 2.0 macro architecture.

The process unfolds in several phases, including preparing speech representations and textual data, GAN input, and training the Generative Adversarial Network (GAN).

Some fundamental steps need to be conducted before the three main phases can be initialized. The first step is to clone a set of GitHub repositories to access the necessary toolkits utilized throughout the process. This applies respectively to the Fairseq framework, rVADFast [86], and kenlm[18] repos.

```
$ git clone https://github.com/facebookresearch/fairseq
$ git clone https://github.com/zhenghuatan/rVADfast
$ git clone https://github.com/kpu/kenlm
```

Listing 1: GitHub repositories

Every repository necessitates installation in accordance with its corresponding documentation. This proved to be a prolonged process due to unclear formulations and code deficiencies in the framework. Furthermore, we operated with limited user permissions on our own system. Consequently, requiring admin approval and subsequent installation on each dependency encountered, leading to a prolonged and time-consuming process.

Once the Fairseq base is established and the environmental variables are set. The implementation of the primary phases can begin. Each phase consists of multiple subprocesses executed in a stepwise fashion. In order to streamline and automate the process, we created an additional module on top of Fairseq, enabling an end-to-end one-step training execution. The module consists of three scripts, one for each phase which can be executed separately or sequentially

[18] `https://kheafield.com/code/kenlm/`

(main) as shown in Figure 4.29.

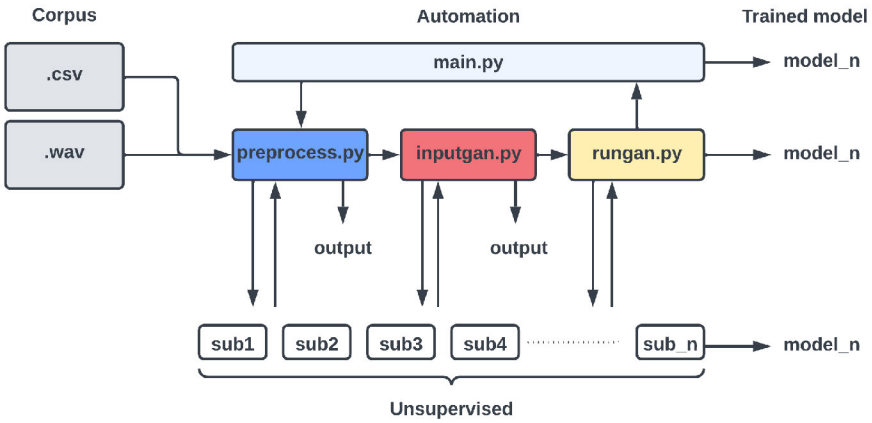


Figure 4.29: Automation of wav2vec unsupervised

The module is designed to take our custom corpus structure (Figure 4.7) as input and, with a one-click execution, produce a trained model as an output. Additionally, the module handles downloads, directories, and file structuring throughout the run-time.

preprocess.py (B.3) covers phase one, where we structure the corpus transcripts into textual input formats. These formats are described as wrd/ltr/phn in the documentation, providing some indications of what the files should contain. Following a series of exploratory trials, the following formats could be utilized.

.wrd = lufthansa three five zero eight
.ltr = l u f t h a n s a | t h r e e | f i v e | z e r o | e i g h t|
.phn = l ʌ f θ æ n s ə θ ɹ iː f aI v z iə ɹ oʊ eI t

Dictionaries for training and validation are generated from the files, creating associations between words and phonemes. In addition, a 'sentence.txt' file, encompassing all unique sentences, is produced. These resources are then used in the following phase.

Additionally, all silence is eliminated from the audio samples. This is accomplished by generating manifest files (.tsv), which associate each audio file with the respective number of samples it contains. Then the rVADFast voice activity detector is applied using short-time Fourier transform (Section 2.3.1) and calculates the spectral flatness (SFT) as in Equation 4.14.

$$\text{SFT(m)} = \frac{exp(\frac{1}{K}\sum_{k=0}^{K-1} ln|X(m,k)|)}{\frac{1}{K}\sum_{k=0}^{K-1}|X(m,k)|} \tag{4.14}$$

where K is the number of frequency bins and X(m,k) signifies the magnitude spectrum of the k^{th} frequency bin in the context of the m^{th} frame. Threshold values decide if voice communication is detected. The detected sampling range in each file is mapped in the .vads files, and a trimmed version of the audio files is created.

The script inputgan.py (B.4) outlines the second phase, where the audio and text data used in the GAN is readied. First, the trimmed audio is processed using frozen pre-trained weights from Wav2vec2-large-960h-lv60-self described in Table 4.7. The contextual vectors outputted from the encoder are stored in a NumPy array serving as an input. In parallel, pseudo labels are created using k-mean clustering of the MFCC representations, attaching each timestep to a centroid. Through a training cycle, the cluster centroid is recalculated to minimize the variance in each cluster. Reducing the summed squared Euclidean distances between the centroid and data points will optimize the precision and shape of each cluster. Important to note that lines of code are missing in the Fairseq clustering script (Listing 2).

```
python $FAIRSEQ_ROOT/examples/hubert/simple_kmeans/learn_kmeans.py \
    $tgt_dir/mfcc $train_split 1 $tgt_dir/mfcc/cls$dim 64
```

Listing 2: Missing code on line 67 in learn_kmeans.py, one of many corrections needed to get the framework running.

Moreover, the collection of unique sentences in the first phase is converted into a phoneme format. This phoneme rendition serves as the "real" data in the GAN architecture. The collection is also used to derive language models utilizing the KenLM toolkit for n-gram models (Section 2.3.2), providing probabilistic mapping of phoneme sequences of n length and Kaldi[19] Finite State Transducers (FST), mapping phoneme-to-phoneme with word split.

$$\text{phn} = \text{l ʌ f } \theta \text{ æ n s ə } \theta \text{ ɹ i: f aI v z iə ɹ oʊ eI t}$$
$$\text{phn-to-phn word} = |\text{l ʌ f } \theta \mid \text{æ n s ə} \mid \theta \text{ ɹ i:} \mid \text{f aI v} \mid \text{z iə ɹ oʊ} \mid \text{eI t}|$$

To the best of our knowledge, the language models are not used directly in the GAN module but utilized when generating from a trained model. It is noteworthy that not only is the recommended PyKaldi tool kit unable to be installed due to dependency conflicts, but the second suggestion also encounters the same issue due to the repo not being maintained. This issue was resolved using a Docker image for the installation, which was extracted (Listing 3).

```
$git clone https://github.com/jjlin/docker-image-extract/
```

Listing 3: Kaldi docker image

Once all inputs have been adequately set up, the rungan.py script (B.5) will initiate the training

[19] https://github.com/pykaldi/pykaldi

process of the GAN. It is crucial to underscore that the original format of the root configuration file corrupts execution and needs to be updated.

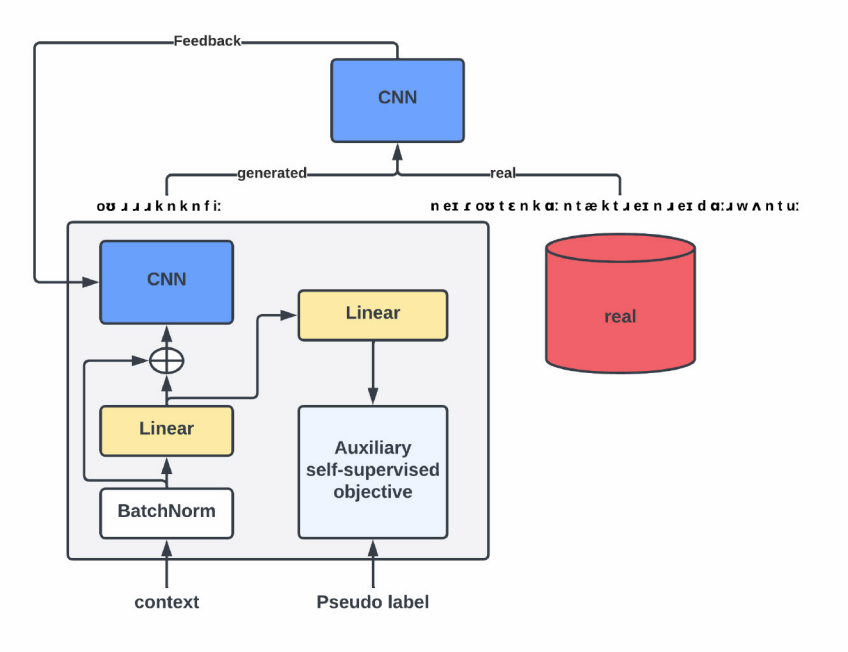


Figure 4.30: GAN architecture

Overall, the GAN comprises two components: the generator and the discriminator. The generator tries to convert the speech signal into a transcript, and the discriminator tries to distinguish between the transcripts produced by the generator and the actual transcriptions. The generator learns to improve its transcriptions to make them more convincing by using feedback from the discriminator, which simultaneously becomes better at distinguishing real from generated transcriptions. This results in a highly competitive process where the generator and the discriminator keep improving over time. This competitive process can be described with Equation 4.15 [56].

$$\min_{G} \max_{C} E_{Y_u}[\log C(Y_n)] + E_X[\log(1 - C(G(X)))] - \lambda \mathcal{L}_{gp} + \gamma \mathcal{L}_{sp} + \eta \mathcal{L}_{pd} + \omega \mathcal{L}_{ss} \tag{4.15}$$

The two first terms describe the objective function of the GAN, where the first term denotes how well the Discriminator C identifies real phoneme sequences Y_u, and the second term denotes how well the discriminator C identifies generated phoneme sequences G(X). The discriminator works towards maximizing the score in both terms, while the generator works to minimize the second term. The third term describes the distributed penalties, imposing regulation in training.

The $\lambda \mathcal{L}_{gp}$ gradient penalty is used to stabilize the training process. It encourages the gradient of the discriminator's output with respect to its input to be close to 1. Doing so prevents extreme variations in gradients, avoiding issues like mode collapse and the vanishing/exploding gradient problem.

$\gamma \mathcal{L}_{sp}$ smoothness penalty is applied to discourage locally inconsistent predictions from the generator.

$\eta \mathcal{L}_{pd}$ Phoneme diversity penalty is applied if a generated sequence lacks phoneme diversity, discouraging it from over-focusing on a limited set of phonemes and encouraging it to explore a wider variety.

New to the Wav2vec-U 2.0 is the $\omega \mathcal{L}_{ss}$ auxiliary self-supervised objective. Testing the generator's

mid-architecture ability to predict a pseudo label at a given timestep in a sequence. The context vectors are diverted through a dense layer with a softmax output with classes reflecting the pseudo labels distributing probabilities that add up to 1. The penalty reflects the model's predictive capacity.

This stabilizes the GAN training performance throughout each iteration and significantly influences the objective function when weights are optimized in the CNN and linear modules.

Ultimately, the stand-alone script, generate.py (B.6), utilize the trained .pt weight file and generates corresponding transcripts from the preprocessed audio.

4.7 Whisper

OpenAI Whisper[20] speech-to-text ASR-system takes a step towards generalized ASR modeling with multilevel capabilities and a zero-shot approach delivered in an easy-to-implement GitHub repository[21] rooted in a paper published December 2022 [75]. The system is based on the well-known encoder-decoder transformer model used and described in Section 4.5, focusing mainly on large-scale weak supervision and a multitasking format during model training cycles.

The repository provides a range of five pre-trained architectural variations, spanning from tiny to large, representing the input embedding, number of layers, and number of heads used in the attention mechanism. Our primary emphasis was on assessing the performance of the small, medium, and large configurations, with the architecture specifications shown in Table 4.8.

Table 4.8: Whisper specifications of small, medium, large architecture

Model	Layers	Width	Heads	Parameters
Small	12	768	12	244M
Medium	24	1024	16	769M
Large	32	1280	20	1550M

At the outset, leveraging the Whisper library for audio transcription seems deceptively straightforward: feed in raw audio files and receive corresponding transcriptions as output. Our exploration commenced with setting up an isolated computational environment, ensuring the incorporation of requisite dependencies. This approach was designed to carry out the experimental processes without undue influence on the root configurations of the computational node 4.2.1.

The next step was to craft an algorithm that could systematically feed the raw audio files from our test corpus into each Whisper model architecture. In response to each audio input, the models generated transcriptions. These output transcriptions were then juxtaposed with the correct transcripts, and this comparison allowed us to compute a WER, providing us with a quantifiable measure of each model's performance (B.7).

Upon initial analysis of the results, it became apparent that a direct comparison of transcriptions was untenable due to differences in the output format. The Whisper models' original output format encompasses a mix of numerical numbers, uppercase, lowercase letters, and special characters. In contrast, our chosen format was much more uniform, strictly utilizing lowercase letters and eschewing the use of special characters (Section 4.3.1.3). To reconcile these disparities and

[20] `https://openai.com`
[21] `https://github.com/openai/whisper`

obtain a meaningful WER, we incorporated a post-processing step to refactor the model outputs, aligning them with the formatting conventions of our corpus. This step was pivotal in ensuring the accuracy of our performance metrics and providing a fair assessment of the models' transcription capabilities.

The outlined functionality of the algorithm is illustrated in Figure 4.31, providing a macro overview of the modules and the connections between them. However, the complexity of the underlying processes can often lead to a disconnect between the user and the system. This disconnect arises due to the sophisticated procedures operating beneath the surface of a few lines of code. Decomposing the Whisper module and the work behind it reveals the complexity and a new approach to the automatic speech recognition (ASR) domain, introducing large-scale weak supervision, which addresses the techniques to attain a corpus of large-scale magnitude with transcriptions. Weak supervision can be described as a hybrid version of supervised and self-supervised learning. The Whisper base corpus consists of audio and paired transcriptions identified and collected from the internet; this is, at a glance, the same structural construction used in supervised learning. However, instead of a costly resource-heavy manual transcription process with strict uniform labeling rules, the weak supervision technique lowers the bar of the transcription quality. This enables an automated screening, scraping, and filtering process to harvest paired web content and scrub the collected data for unwanted characteristics such as ASR created data, duplicates, and misaligned language identification. A preliminary model is trained. This model is tasked to identify the least effective or low-performing samples. These identified samples are then manually inspected and reviewed, facilitating further refinement of the corpus. This approach facilitates the development of extensive corpora akin to what's typically observed in self and unsupervised methodologies.

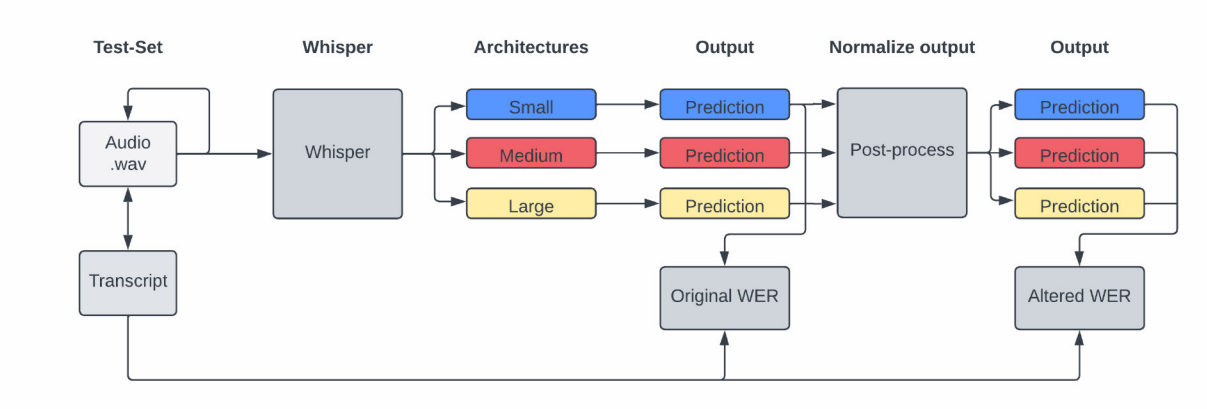


Figure 4.31: Algorithm macro functionality

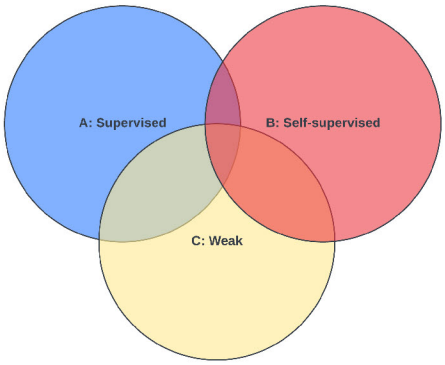


Figure 4.32: Intersecting features of ASR methods; Audio = A ∩ B ∩ C, Transcript = A ∩ C, Low cost scalability = B ∩ C.

The Whisper architectures implemented in our research are pre-trained on 680.000 hours of multilingual and multidomain audio. The audio data are sampled at 16 KHz, segmented into timesteps of 25 milliseconds with a ten milliseconds stride represented as log-magnitude Mel spectrogram. The spectrograms then undergo feature extraction with two one-dimensional Convolutional layers before sinusoidal positional encoding tag relevant sentence positions as a precursor for the encoder's input embedding. The same underlying pre-process is instantiated to the input audio files to utilize the pre-trained transformer system and get a zero-shot prediction.

Whisper demonstrates multilingual and multipurpose capabilities in terms of English audio to English transcriptions, Non-English Audio to English transcriptions, Non-English audio to Non-English transcriptions, and no-speech detection (Figure 4.33). Achieved by a set of unique tokens infused in the base corpus and used throughout the training phase to facilitate a multi-tasking approach, respectively a start-of-transcript-token, a no-speech-token, a transcribe-token, a no-time-stamps-token, and an end-of-transcript-token. Our scope utilizes a part aspect of the system's capabilities by using English ATC samples as described in Section 4.3.1.3. The results produced by the Whisper implementation are significant because of its "general" model approach.

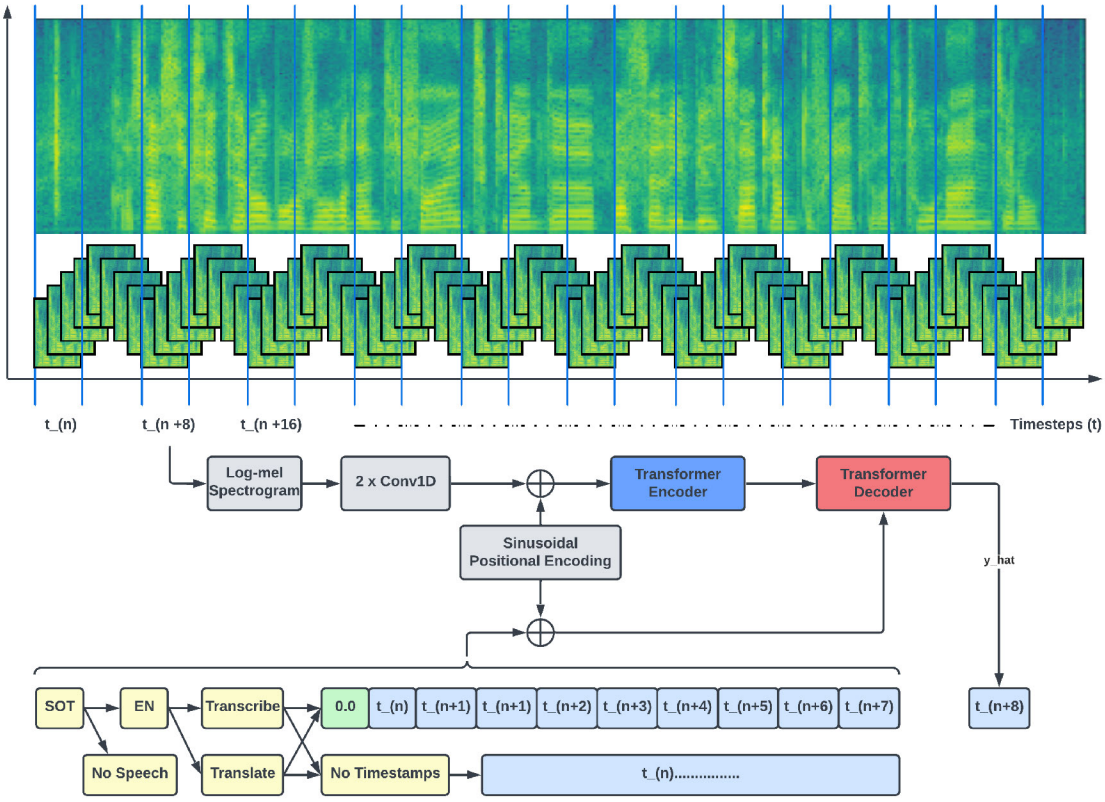


Figure 4.33: Whisper multilingual and multipurpose capabilities

Chapter 5

Results

This chapter presents the results of implementing methods and strategies outlined in Chapter 4, emphasizing the differing word error rate (WER) achieved by various automatic speech recognition (ASR) technologies under more or less identical training conditions. We present concrete examples of model predictions, supplemented by crucial feature extraction data and hyperparameter adjustments during the training process. This side-by-side comparison shows how these technologies perform differently and point out their potential when handling in- and out-of-domain data in some instances.

Furthermore, we document the progression of models over the training period, offering a glimpse into their evolution and refinement. Our findings, presented in various forms for comprehensive understanding, aim to show how different methods perform when presented with identical data. Our primary goal is to demonstrate these varied performances, enhancing the understanding of ASR technologies' capabilities in the ATC domain.

5.1 Deep Speech 2

The optimal configurations for both corpora, limited to the top 5, accompanied by hyperparameter details, are presented in Table 5.1.

The model's performance on ATCOSIM and Hiwire simultaneously exhibits comparable results when trained on individual corpora such as ATCOSIM, Hiwire clean, and Hiwire low noise (LN). However, a notable performance gap emerges when the model is trained on the entire Hiwire set. The introduction of noise in Hiwire significantly affects performance. Nevertheless, it remains within an acceptable range of approximately 20% Word Error Rate (WER), ensuring a reasonable level of comprehensibility in the generated outputs.

Considering the impact of batch size, the model performs slightly better on ATCOSIM when a smaller batch size is utilized. Conversely, with the Hiwire corpus, the opposite trend is observed. As depicted in Figures 5.1, 5.2, 5.3 and 5.4, the training and validation loss exhibit a rapid decrease within the initial 30 epochs, followed by a leveling out and reaching a steady state for both training and validation loss.

Table 5.1: Deep Speech 2, ATCOSIM and Hiwire results

corpus	epochs	batch size	dropout	f_size	f_step	FFT bins	Mel bands	WER (test)
all	100	64	50%	400	80	1024	80	10.1%
-	-	-	60%	-	80	-	-	10.3%
-	-	-	50%	-	160	-	-	11.3%
-	-	-	60%	-	-	-	-	15.2%
ATCOSIM	100	8	50%	400	160	2048	70	8.8%
-	-	-	-	-	-	1024	80	10.9%
-	200	64	-	-	-	-	70	10.9%
-	150	-	-	-	-	512	-	11.0%
-	200	-	-	-	80	1024	-	11.1%
Hiwire all	200	64	50%	400	160	1024	70	5.3%
-	-	-	-	-	80	-	-	5.4%
Hiwire clean	100	64	50%	400	160	1024	70	9.3%
-	200	-	-	-	80	-	-	10.3%
-	100	-	-	-	160	-	-	10.3%
-	200	-	-	-	-	-	-	10.9%
-	150	32	70%	-	-	-	50	11.3%
Hiwire LN	100	50	50%	400	160	1024	80	10.2%
-	200	64	-	-	-	-	70	16.3%
-	-	-	-	-	80	-	-	16.4%
-	100	50	-	-	160	512	80	28.3%
-	200	80	-	-	-	1024	-	38.5%
Hiwire MN	200	64	50%	400	80	1024	70	18.5%
-	-	-	-	-	160	-	-	19.9%
-	100	50	-	-	-	-	80	21.5%
-	-	-	-	-	-	512	-	22.3%
-	200	80	-	-	-	1024	-	29.1%
Hiwire HN	400	50	40%	400	160	2048	80	21.6%
-	-	70	30%	-	-	-	-	22.9%
-	-	-	40%	-	-	-	-	23.1%
-	-	60	-	-	-	-	-	26.3%
-	-	-	30%	-	-	-	-	26.3%

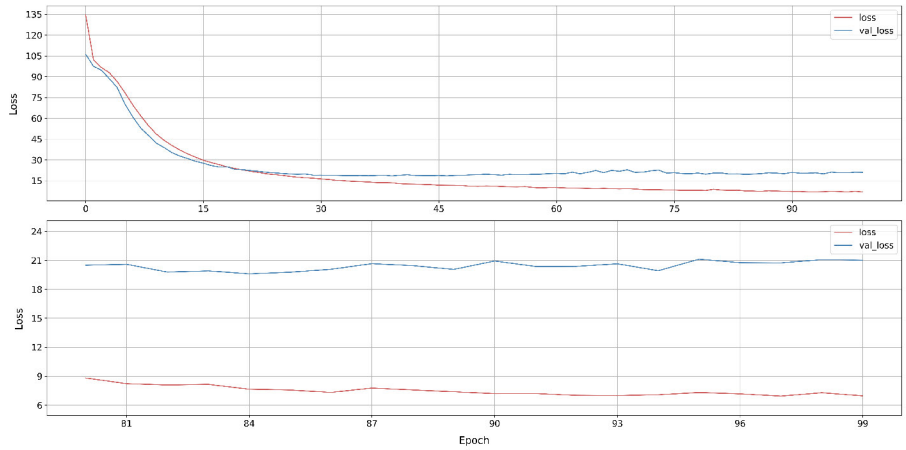


Figure 5.1: Loss function for ATCOSIM and Hiwire combined

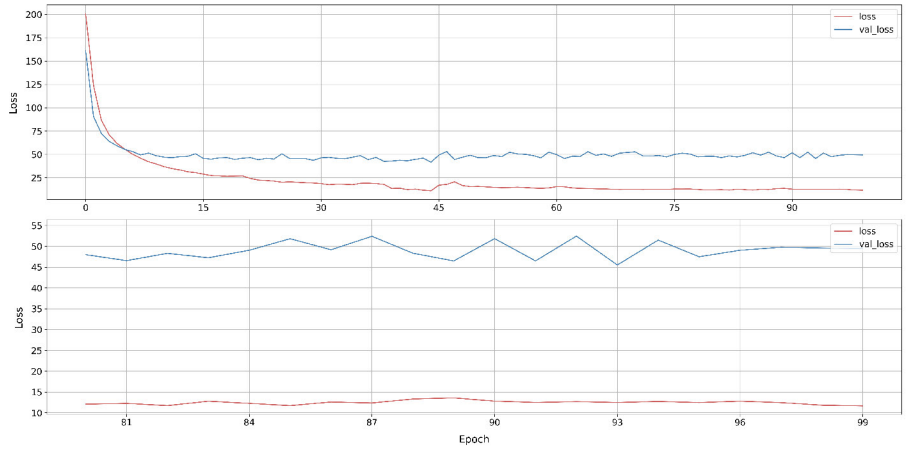


Figure 5.2: Loss function for ATCOSIM only

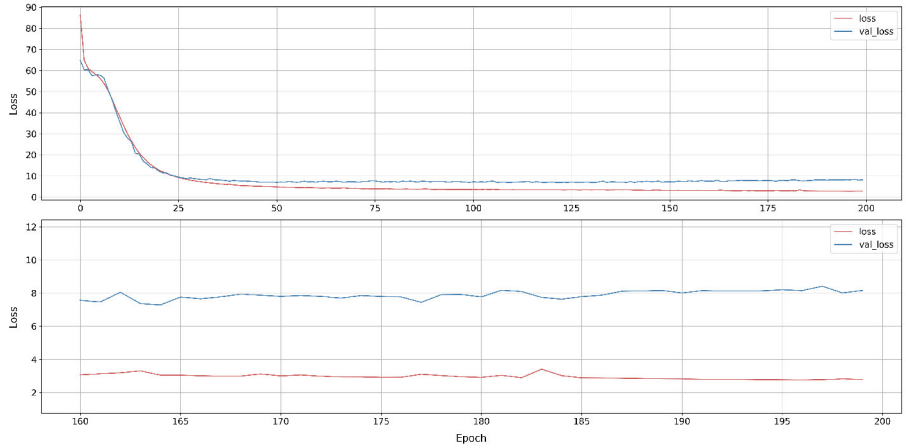


Figure 5.3: Loss function entire Hiwire.

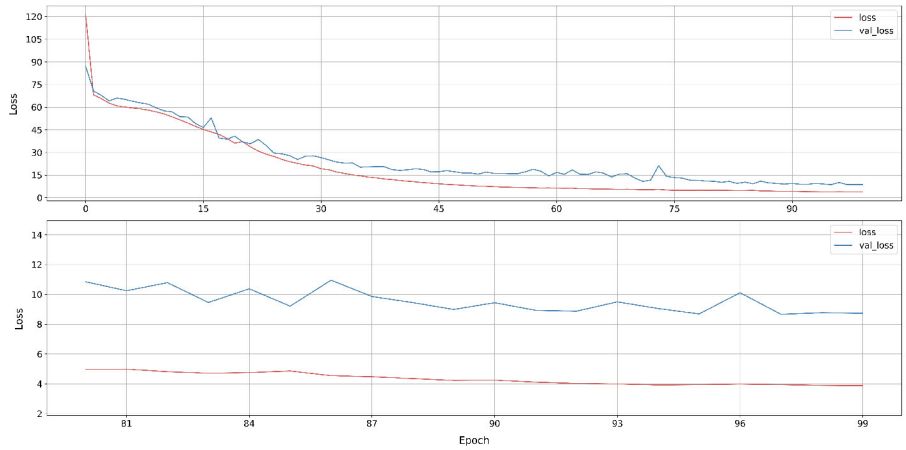


Figure 5.4: Loss function for Hiwire clean

5.1.1 Cross-domain experiments

The Hiwire corpus allows cross-domain WER evaluations utilizing sub-corpora with varying levels of synthetic noise incorporated into the underlying base data. When examining the best performing model (Table 5.2), which was trained on all the sub-corpora simultaneously, the numbers reveal that performance on the high noise subset is significantly worse than the rest.

Table 5.2: Deep Speech 2, best model trained on entire Hiwire corpus subset performance comparison.

subset	total samples	0% WER	Between 0% and 100%	100% WER
HN	342	65.2%	21.6%	12.9%
MN	328	88.7%	6.7%	4.6%
LN	309	91.9%	4.2%	3.9%
clean	336	89.0%	6.9%	4.2%

Table 5.3: Deep Speech 2, individual predictions with WER from the Hiwire corpus.

Subset	Target	Prediction	WER
clean	add	ad	100%
clean	pfd	p3d	100%
clean	maintaining	maintaing	100%
clean	select hf2 two eight three two	select h o to	83.3%
clean	eta delta golf alpha kilo alpha at six three	eta delta golf alpfha kilalfoax sixthree	66.7%
clean	speed one dot three three eight	speed one dt three three eight	16.7%
HN	wilco	list	100%
HN	descent to	message	100%
HN	route	rote	100%
HN	vhf3 one three six dot nine five	potion juliet srit six edecema neine five	71.4%
HN	position eight lima tango lima fox	position ie nineaa ike fox	66.7%
HN	hf2 six five zero nine	hf2 six five zero	20.0%
HN	request descent to five five two	request descent thoe five five two	18.2%

The confusion matrix shown in Figure 5.5 provides some insight into the performance comparison between the best-performing model configuration trained on the entire corpus and the same model trained exclusively on a subset.

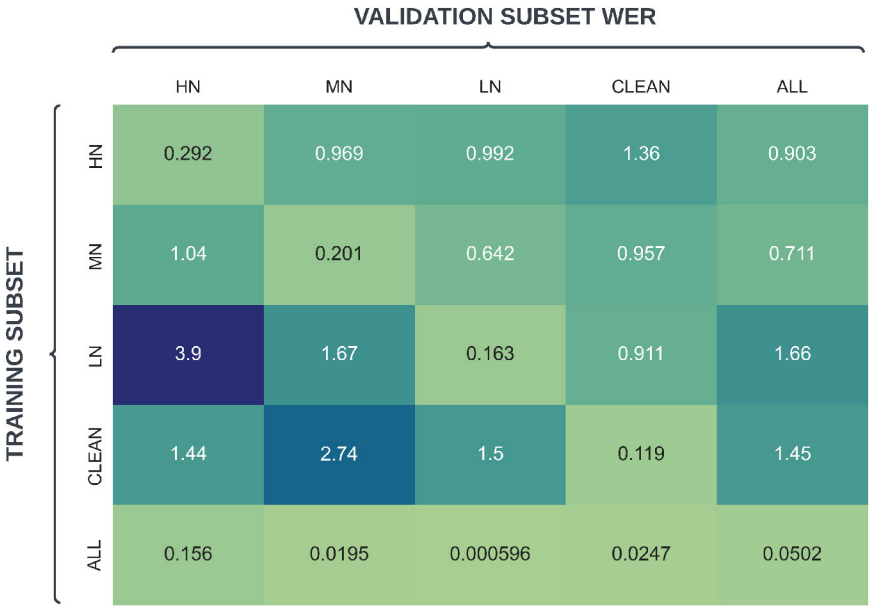


Figure 5.5: Model performance comparison between Hiwire sub-corpora.

In a final post-processing step, the models' predictions undergo a basic correction function using the Levenshtein distance as described in Section 4.4.1. This approach yielded a decrease in total WER of 2.4 percentage points on ATCOSIM and 1.4 percentage points on Hiwire, examples of which are listed in Table 5.4.

Table 5.4: Deep Speech 2 corrected output examples.

Target	Prediction	WER	Corrected	WER
alitalia one one seven guten tag identified	alitalia one one seven guten taeg identified	14.3%	alitalia one one seven guten tag identified	0.0%
sabena nine three seven two calling	sabena cninbe three seven two calling	16.7%	sabena nine three seven two calling	0.0%
position charlie zero five papa nine	position charlie zero five pava one	33.3%	position charlie zero five papa one	16.7%

5.2 Transformer

The results produced by the Transformer are based on the looped hyperparameter cycles described in Section 4.5. With a projected timespan stretching throughout loop one scheduled in Figure 4.1, a set of 1250 individual models was trained. The main goal was to observe model performances on the ATC domain and get familiar with the impact of the parameters on the specific task. The set threshold of 5.8% WER did not subside the human-level scores achieved on the LibriSpeech clean corpus. It is pertinent to note that the models' performance metrics did not dictate the end of the experiment cycle but rather by the predetermined time frame as mentioned in Section 4.1.

Table 5.8 presents the top five best-performing models in each sub-corpus with the STFT input. We can observe that the differences in sub-corpus complexity and noise levels constitute the most significant differences in prediction capability. It is also apparent that the top models have similar hyperparameter settings with subtle differences. The vast amount of models trained has created hyperparameter clusters, but the difference between the best and worst models is significant.

We see that small batch sizes perform the best on the clean data, while the opposite is observed on the noisy and aggregated data. We also note that the best-performing model's transformer architectures are generally more shallow regarding the number of layers and attention heads employed in the encoder and decoder.

Table 5.5: Transformer, STFT features

corpus	batch_size	batch_size_val	n_hid	n_head	n_feed_forward	n_layers_enc	n_layers_dec	WER (test)
ATCOSIM	4	2	200	2	400	4	1	30.0%
-	8	4	-	-	-	-	-	31.0%
-	8	8	-	-	-	-	-	31.0%
-	16	16	-	-	-	-	-	32.0%
-	16	16	-	-	-	-	-	33.0%
Hiwire clean	4	4	200	2	400	4	1	9.7%
-	-	2	300	-	300	-	2	10.6%
-	-	-	200	4	-	-	1	10.8%
-	-	-	100	2	500	2	1	11.3%
-	-	4	200	-	300	-	2	11.3%
Hiwire HN	32	32	200	2	400	4	1	64.0%
-	16	16	-	-	-	-	-	65.0%
-	-	-	-	-	-	-	-	66.0%
-	-	8	-	-	-	-	-	70.0%
-	16	16	-	-	-	-	-	76.0%
All	8	4	200	2	400	4	1	50.0%
-	32	16	-	8	600	8	8	51.2%
-	16	8	-	2	400	4	1	53.0%
-	32	16	-	8	600	8	8	54.8%
-	-	-	400	-	-	-	-	55.7%

When observing selected samples from predictions made by the best-performing models in each sub-corpora, we see that the Transformer excels in producing sentences without misspellings (Table 5.6). However, the errors that occur show that the model is prone to substitutions which significantly affect the WER.

Table 5.6: Transformer, STFT output samples.

Target	Prediction	WER
ATCOSIM		
hamburg air two five zero zero good morning radar contact direct st prex	hapag lloyd two five zero good morning radar contact prex	38.0%
hapag lloyd ah six five three good morning climb to flight level two seven zero	hapag lloyd ah seven seven six five good morning climb to flight level two seven zero	20.0%
alitalia four one six radar contact climb flight level three zero zero final level	alitalia four one six radar contact climb flight level three zero zero	21.0%
aerovic one zero six one contact milano one three four decimal five two	air one zero six one contact zurich one three four decimal four	31.0%
topswiss four five five two climb to flight level three two zero	line five two four five two climb to flight level three two zero	33.0%
adria one six four two good afternoon squawk two seven one zero	hapag lloyd one six four good afternoon squawk two seven one zero	25.0%
delta mike echo fly direct to karlsruhe climb flight level two seven zero now	delta mike echo bravo echo climb flight level two seven zero now reading now	43.0%
jet set four four two on heading one seven zero descend to flight level two nine zero	jet set four four two one eight climb to flight level two nine zero	29.0%
Hiwire clean		
position bravo zulu whiskey lima one	position kilo four whiskey hotel seven	66.67%
request direct to yankee echo hotel due to weather	request direct to yankee due to weather	22.22%
request speed four decimal zero due to performance	request speed four point zero due to performance	12.5%
eta alpha november india golf victor at two three hours four three minutes	eta one six hotel mike foxtrot one three hours four one minutes	61.54%
we cannot accept plus nine nine nine	we cannot accept plus nine nine	14.29%
position kilo november lima uniform hotel	position one alpha november one charlie	83.33%
level one eight three seven seven	level one eight five seven zero	33.33%
fd engage	fd disengage	50.0%
Hiwire HN		
request direct to oscar four victor alpha zero due to aircraft performance	request direct to echo foxtrot oscar due to aircraft performance	42.0%
we can accept plus six five zero at six zero three minutes	we can accept nine six seven at three zero minutes	42.0%
select hf2 three four nine six	select hf2 three six five eight	50.0%
weather radar off	weather radar	67.0%
request speed zero dot nine one due to performance	request speed zero dot six six nine due to performance	33.0%
request descent to five five two	request descent to one	50.0%
switch frequency	change frequency	50.0%
position hotel quebec golf mike golf	position yankee three two yankee november	83.0%

In Table 5.7, we also see the clustering effect, with many parameter similarities. The main difference from the STFT is the number of heads and layers in the encoder and decoder. When handling a less complex input representation, the transformer's architectural depth seems more complex.

Table 5.7: Transformer, MFCC features

corpus	batch_size	batch_size_val	mel_bins	n_hid	n_head	n_feed_forward	n_layers_enc	n_layers_dec	WER (test)
ATCOSIM	4	2	40	400	8	600	8	8	28.4%
-	-	-	-	-	-	-	-	-	29.6%
-	-	-	-	-	-	-	-	32	29.7%
-	-	-	-	-	-	-	-	16	29.9%
-	-	-	-	-	-	-	-	8	30.0%
Hiwire clean	4	2	80	400	8	600	8	8	6.8%
-	-	-	40	200	4	300	2	2	7.8%
-	-	-	80	-	-	-	-	-	8.4%
-	-	-	40	400	8	600	8	8	8.8%
-	-	-	-	-	-	-	-	-	8.8%
Hiwire LN	4	2	80	600	8	600	16	16	14.9%
-	-	-	40	-	-	800	8	8	15.3%
-	-	-	-	-	-	-	-	16	15.3%
-	-	-	80	400	-	600	16	8	15.3%
-	-	-	-	-	16	800	-	32	15.3%
Hiwire HN	4	2	40	400	8	600	8	8	52.0%
-	-	-	80	-	-	-	-	-	54.3%
-	8	4	-	600	-	800	-	-	58.4%
-	-	-	-	-	16	-	16	32	58.7%
All	64	32	120	400	8	600	8	8	123.7%
-	128	64	80	-	-	-	-	-	131.9%
-	64	32	40	-	-	-	-	-	134.6%
-	-	-	80	-	-	-	-	-	145.9%
-	8	4	-	-	-	-	-	-	152.7%

The type of errors observed in Table 5.8 shows the same particular characteristics as observed in Table 5.6.

Table 5.8: Transformer, MFCC output samples.

Target	Prediction	WER
ATCOSIM		
lufthansa three five six six if cleared higher what would be rate of climb	lufthansa three five six six three climb ilevel three climb ilevel three bye	64.29%
four two zero nine i told you to contact rhein one three two decimal four	foxtrot zero nine contact rhein one three two decimal four	40.0%
swissair nine three five two rhein radar identified proceed direct to gotil	swissair nine three five two rhein radar identified climb	33.33%
air berlin five two six one geneva one three three decimal one five bye bye	berlin five two six one two contact geneva one three decimal one five bye bye	33.33%
transwede three five seven one turn right one zero degrees due traffic	transwede three five seven one turn right one zurich east two three two three two due trafff	66.67%
georgia zero three seven contact rhein one two seven decimal three seven	german level three zero seven contact rhein one two seven decimal three seven	25.0%
air malta zero zero four for separation turn right by ten degrees	air malta zero zero for separ separar ah separation ten degrees	41.67%
austrian two two contact milan one three four decimal five two bye	oscar two three four contact milan one three four decimal five two bye	25.0%
Hiwire clean		
select vhf2 one two five	select vhf2 one five two five	20.0%
hf1 three zero seven eight	hf1 three zero seven eight zero	40.0%
request climb to eight four five due to weather	request climb to two four five due to weather	11.11%
eta seven romeo victor india uniform at one one hours seven	eta romeo papa four seven xray at one seven two	72.73%
select vhf2 one five two decimal zero	select hf2 five zero one two	71.43%
level one eight three seven seven	level one two four dot three seven	66.67%
preferred level five two zero	preferred level five zero two	40.0%
preferred level five four three	preferred level four one four	60.0%
Hiwire LN		
position tango victor hotel golf zulu	position tango victor hotel golf zero	16.67%
request climb to eight four five due to weather	request climb to mike at five four weather	44.44%
request direct to lima tango oscar six zulu	request direct to lima tango oscar six zero due to weather	50.0%
we cannot accept two zero zero five nine	we cannot accept two zero zero five nine one	25.0%
position hotel quebec golf mike golf	position quebec golf nine two	50.0%
preferred level four one eight	preferred level seven four zero	60.0%
request speed zero dot nine one due to performance	request level seven two minutes	88.89%
ils off	ils on	50.0%
Hiwire HN		
preferred level five four three	preferred level minus seven	60.0%
select hf1 five four three four	select hf1 five zero nine two	50.0%
select vhf2 one five two decimal zero	select hf2 two zero seven eight five	85.71%
request climb to plus seven three six due to aircraft performance	request descent to plus seven five six due to aircraft performance	18.18%
request climb to eight four five due to weather	request direct to whiskey oscar echo foxtrot delta due to weather	66.67%
position tango victor hotel golf zulu	position one alpha november one charlie	83.33%
request direct to lima tango oscar six zulu	request descent to plus seven five due to weather	87.5%
maintaining zero zero six one nine	maintaining seven five six	83.33%

5.3 Hugging Face

The results achieved with the wav2vec 2.0 fine-tuning can be separated into two parts, before and after the final corpora setup as described 4.6.1. Our early perception of a generalized ATC model characterizes the first setup, which can cross from the simulated to real-world ATC domain. In Table 5.9, the results from the augmented data are presented. The data is partitioned in a clean version with three noise levels, and ATOCOSIM_BIG encapsulates all versions. Test1 WER scores are conducted on samples from the corpus, and Test2 WER are samples from real ATC.

Every model performs well on unseen data from its corpus pool, but there is a clear difference when the models are exposed to real ATC communications. The models 1-5, which were fine-tuned on ATCOSIM_BIG, substantially outperformed the top-performing sub-corpus of ATCOSIM_BIG, exhibiting a considerable 34.59 percentage point difference in WER.

Table 5.9: Cross-domain results, pretrained models on ATCO2-ASR-ALL.

Index	Model	Corpus	Epochs	Batch size	Test1 WER	Test2 WER
1	wav2vec2-large-a-ATCOSIM_big-E50-B64_base-ATCOSIM_big-E50-B8_base	ATCOSIM_BIG	50	8	4.30%	29.57%
2	wav2vec2-large-a-ATCOSIM_big-E50-B64_base-ATCOSIM_big-E50-B16_base	-	-	16	4.64%	30.02%
3	wav2vec2-large-a-ATCOSIM_big-E50-B8_base	-	-	8	5.21%	30.92%
4	wav2vec2-large-a-ATCOSIM_big-E50-B16_base	-	-	16	4.92%	31.39%
5	wav2vec2-large-a-ATCOSIM_big-E50-B64_base	-	-	64	6.66%	32.27%
6	wav2vec2-large-a (baaddfaa495711edbd965723004f39d0)	ATCOSIM_clean	-	16	1.13%	64.16%
7	wav2vec2-large-a (52b9e0e0495911edb38d53b30a9f3c33)	-	-	8	1.12%	65.35%
8	wav2vec2-large-a-baaddfaa495711edbd965723004f39d0 (958c26784a2411ed80732959f50e2524)	ATCOSIM_noise3	-	64	17.64%	71.01%
9	wav2vec2-base-960h-clean-e50-b64	ATCOSIM_noise1	50	-	2.30%	72.60%
10	wav2vec2-base-960h-NOISE1-e100-b64	ATCOSIM_noise2	100	-	4.30%	76.30%
11	wav2vec2-base-960h	-	-	-	1.70%	76.60%

The secondary setup was predominantly concentrated on the ATOCOSIM corpus. All the pipelines performed exceptionally well, with near-perfect predictions in most cases. Table 5.10 and Figure 5.6 reveal minuscule differences in the different models. The results align with the results produced in Table 5.9 and underscore the capabilities of the downstream method.

Table 5.10: Hugging Face pipelines best performance.

Model	Batch size	Train loss	Val loss	Test WER
wav2vec2-large-960h-lv60-self	8	0.018	0.045	0.95%
wav2vec2-large-robust-ft-swbd-300h	-	0.017	0.042	1.01%
wav2vec2-large	-	0.006	0.054	1.04%
wav2vec2-large	16	0.036	0.062	1.09%
wav2vec2-xls-r-300m	8	0.011	0.055	1.3%
wav2vec2-base	16	0.016	0.113	1.46%

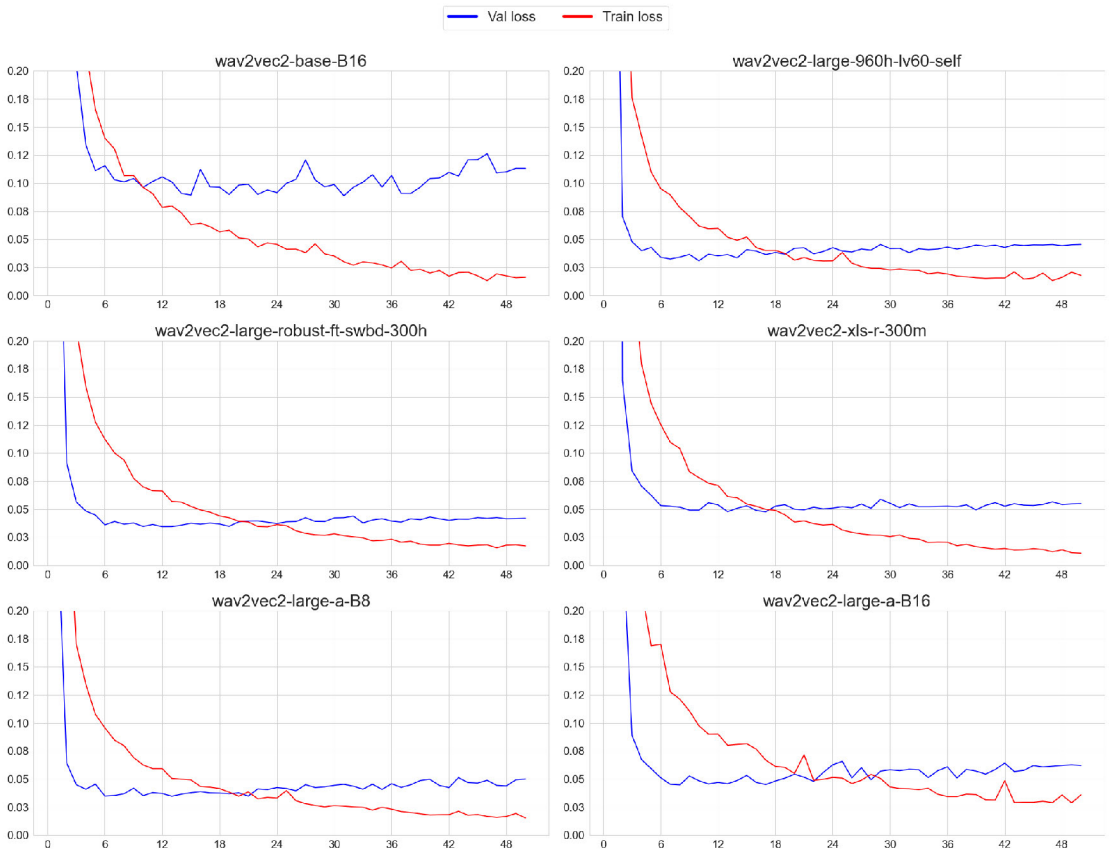


Figure 5.6: Hugging Face pipelines graphs.

Probing further into the individual WER scores for each sample in the test sub-corpus reveals that most samples yield a WER of 0%, a perfect prediction. As seen in Table 5.11, the mean WER for the samples which were not perfectly predicted remains low.

Table 5.11: Hugging Face pipelines WER details (* indicate that samples with character length below 10 were removed from test sub-corpus).

Model	0% WER	100% WER	Between 0 and 100%	0-100% mean
facebook-wav2vec2-large-960h-lv60-self *	858	0	75	12.10%
facebook-wav2vec2-large-robust-ft-swbd-300h *	852	0	81	11.02%
yongjian-wav2vec2-large-a-B8 *	848	0	85	12.65%
yongjian-wav2vec2-large-a-B16	868	2	86	16.04%
facebook-wav2vec2-xls-r-300m *	838	0	95	14.52%
facebook-wav2vec2-base	847	2	107	13.72%

The nature of errors in Table 5.12 is also observed in Section 5.2 regarding the Transformer predictions.

Table 5.12: Hugging Face output samples.

Target	Output	WER
facebook-wav2vec2-large-960h-lv60-self		
ah metavec one zero six one turn now right heading two one zero sorry	and medavic one zero six one h turn now right heading two one zero sorry	21.43%
thats correct stand that routing	thats correct stand at routing	20.0%
airfrans one five zero nine yeah stand by	airfrans one five zero nine yeahe stand by	12.5%
okay were finished thank you have a nice weekend	okay we finished thank you have a nice week and	33.33%
thank you tschuss	thank you schuss	33.33%
swissair four eight eight checked in	swissair four eight eight checkeding	33.33%
roger youre identified	roger identified	33.33%
facebook-wav2vec2-large-robust-ft-swbd-300h		
okay were finished thank you have a nice weekend	okay we finished thank you have a nice weckand	22.22%
ah expect higher later on after crossing traffic above	ah expect higher later on i have crossing traffic above	22.22%
airtas one guten tag identified report your heading	airtors one guten tag identified report your heading	12.5%
ah metavec one zero six one turn now right heading two one zero sorry	and medarovec one zero six one ah turn now right heading two one zero sorry	21.43%
roger youre identified	roger identified	33.33%
hoi k l m three four six identified	hoil k l m three four six identified	12.5%
fox sierra india climb to level three seven zero	ah fox sierra india climb to level three seven zero	11.11%
yongjian-wav2vec2-large-a-B8		
air adria one six two seven is identified	yeah adria one six two seven is identified	12.5%
belgian airforce two three four own navigation direct to epinal	rogern airforce three three four own navigation direct to epinal	20.0%
ah metavec one zero six one turn now right heading two one zero sorry	anh merarvec one zero six one turn now right heading two one zero sorry	14.29%
okay were finished thank you have a nice weekend	okay we finish thank you have a nice wd again	44.44%
ah expect higher later on after crossing traffic above	ah expect higher later on i have crossing traffic above	22.22%
yeah you too	yeah you too b	33.33%
alright one three three one five please	a rih one three three one five please	28.57%
yongjian-wav2vec2-large-a-B16		
climb to flight level three eight zero	ah climb to flight level three eight zero	14.29%
and at level thr two nine zero	and at level thro two nine zero	14.29%
ah okay then we finish	okay then were finished	60.0%
air inter sierra oscar set course direct to kamas	air intera sierra oscar set course direct to kamas	11.11%
speedbird one two four good morning radar contact	speedbird ah one two four good morning radar contact	12.5%
ohh la la whats going on behind	olla la whats going on behind	28.57%
and nine three six zero contact rhein one two seven three seven tschu	ad nana three six zero contact rhein one two seven three seven tschu	15.38%
facebook-wav2vec2-xls-r-300m		
georgia air zero three seven is in radar contact	georgia zero three seven is in radar contact	11.11%
alitalia one three five what is your mach number	alitalia one three five what is you mach number	11.11%
thats correct stand that routing	thats correct and routing	40.0%
yeah should be with ah one two eight one five i think	yea should be withe ah one two eight one five ai think	25.0%
one ah three three four please	one three three four please	16.67%
alright one three three one five please	aright one three three one five please	14.29%
merair six nine five two is identified ill call you back for higher soon	merair six nine five two is identified ill call yu back for higher seveln	14.29%
facebook-wav2vec2-base		
japanair four one nine zurich	japan air four one nine zurich	40.0%
airfrans three two six one no further speed restrictions	airfrans three two six one now further speed restrictions	11.11%
p s a five two six hotel contact rhein one three two decimal four	be ls ar five two six hotel contact rhein one three two decimal four	21.43%
ah what is your full callsign	ah what is your four callsign	16.67%
air adria one six two seven is identified	eah adria one six two seven is identified	12.5%
yeah follow flight planned route radar contact	yeah furlo flight planned route ahga radar contact	28.57%
gulf air zero three two ah proceed direct fusse	gulf air zero three thre ahd proceed direct fusse	22.22%

5.4 wav2vec-U 2.0

The results produced with the unsupervised method are the results of an extensive implementation phase of the novel generative approach to the ASR domain. Figure 5.7 show the results of one of the first successful training runs after finally getting the framework operational.

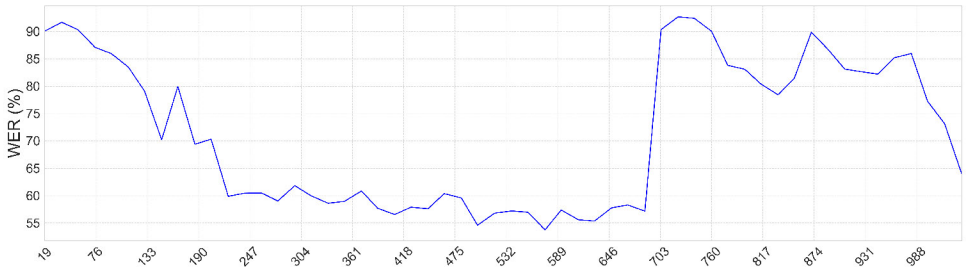


Figure 5.7: Wav2Vec-U 2.0 training details, 53% WER model.

The wav2vec-U framework utilizes phoneme token representations, posing considerable difficulty in generating comparative and interpretative outputs. The alignment of the source text was a primary source of confusion, with initial runs yielding outputs that were difficult to decipher.

The samples presented in Table 5.13 emerged from numerous iterations in training. Here, rather than separating phonemes by spaces, they were joined together, consistent with the model output regularly displayed on the terminal log to facilitate tracking of the WER progression.

Table 5.13: wav2vec-U 2.0 phoneme predictions, early stage.

model	output
-	cross air six eight zero bonjour climb to flight level three zero zero
target	kɹɔsɛɹsɪkseɪtziəɹoʊbɔ:nʒʊɹklaɪmtəflaɪtlɛvəlθɹi:ziəɹoʊziəɹoʊ
libri960	ɑ: ʃɑ:kɔ:sɛɛɹsɪkseɪtziəɹoʊmɑ:zɹoʊʃɑ:ʃklaɪmbaɪtlɛvəlθɹi: ziəɹoʊziəɹoʊmɑ:
xlsr_53_56	θi:pkɑ:o:ɹsɛɹsɪkseɪtziəɹoʊbɑ:dʒəˑðəˑklaɪmklaɪmlɛvəlθɹi:ziəɹoʊiəɹni:
-	delta mike echo turn right to trasadingen
target	dɛltəmaɪkɛkoʊtɜ:naɪttəɪæsɪdɪndʒən
libri960	ɑ:dɛltəmaɪækɛkoʊtɜ:ɹaɪtəɪæsɪdɪmɑ:
xlsr_53_56	æi:dɛltəmaɪkɛkoʊtɜ:ɹaɪttəɪæsɪdɪni:
-	aero lloyd five three three is identified
target	ɛɹoʊlɔɪdfaɪvθɹi:θɹi:ɪzaɪdɛntɪfaɪd
libri960	ɑ:eɪɛɹɹvaɪfaɪvθɹi:θɹi:ɪzaɪdɛntɪfaɪd
xlsr_53_56	mɛoʊɔɪdfaɪvθɹi:θɹi:ɪzaɪdɛntɪfaɪni:
-	affirm swissair nine three six zero contact rhein radar one two orange seven decimal three seven
target	ɐfɜ:mswɪsɛɹnaɪnθɹi:sɪksziəɹoʊkɑ:ntækʈɛɪnɹeɪdɑ:ɹwʌntu:sɛvəndɛsɪməlθɹi:sɛvən
libri960	ɑ:ɛɹfɜ: swɪsɛɹnaɪnθɹi:sɪksʃɪzɹoʊkɑ:ntækʈɛɪnɹeɪdɑ:ɹwʌntu:sɛvəndɛsɪəlθɹi:sɛvmnɑ:
xlsr_53_56	ɛɹfɜ: msswɪsɛɹnaɪnθɹi:ssɪkssziəziəɹoʊðəˑkɑ:ntæktɛɪneɪdɑ:sɹwʌntu:sɛvndɛsɪməlθɹi:sɛvəni:

Figure 5.8 show the performance of the best model, yielding a final WER of 23.7% on ATCOSIM. The variance in the two final results was traced back to misalignment between the transcript and audio file in the test sub-corpora (referred to as validation files in the framework). Upon discovering this misalignment, the WER dropped down to 23.7%.

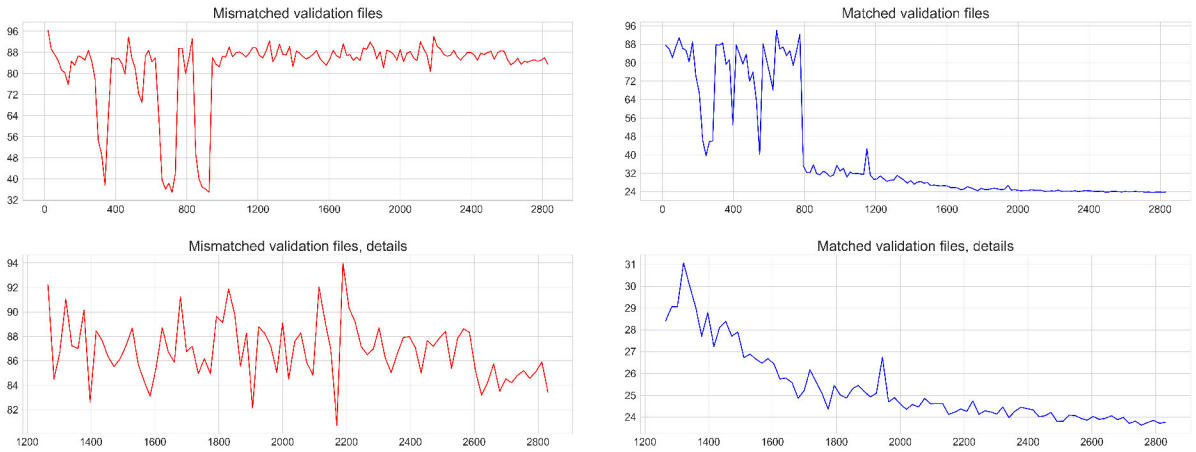


Figure 5.8: Wav2Vec-U 2.0 training details, 23.7% WER model.

The final prediction format is presented as a sequence of phonemes mapped against the ground truth (Table 5.14). Identifying the error patterns before the post-processing is challenging at first glanze yet there is an overall similarity present

Table 5.14: wav2vec-U 2.0 phoneme predictions, from the model achieving a WER/UER of 23.7%

alitalia four one three can you accept flight level three one zero as cruising level today	
target	æ l ɪ t eɪ l iə f oːɹ w ʌ n θ ɹ iː k æ n j uː ʁ k s ɛ p t f l aɪ t l ɛ v əl θ ɹ iː w ʌ n z iə ɹ oʊ æ z k ɹ uː z ɪ ŋ l ɛ v əl t ə d eɪ
prediction	æ l ɪ t eɪ l iə f oːɹ w ʌ n θ ɹ iː k j n j uː k ɛ k f l aɪ ə l ɛ v əl θ ɹ iː w ʌ n z iə ɹ oʊ f ɛ z ɪ d ɪ l ɛ v əl t ə d ɪ b aɪ
hamburg air two five four six climb to flight level three zero zero	
target	h æ m b ɜː g ɛɹ t uː f aɪ v f oːɹ s ɪ k s k l aɪ m t ə f l aɪ t l ɛ v əl θ ɹ iː z iə ɹ oʊ z iə ɹ oʊ
prediction	m b ɜː g ɛɹ t uː f aɪ v f oːɹ s ɪ k s k l aɪ m f l aɪ t l ɛ v əl θ ɹ iː z iə ɹ oʊ z iə ɹ oʊ
georgia air three correction zero three seven rhein identified	
target	dʒ ɔːɹ dʒ ə ɛɹ θ ɹ iː k ɚ ɹ ɛ k ʃ ə n z iə ɹ oʊ θ ɹ iː s ɛ v ə n ɹ eɪ n aɪ d ɛ n t ɪ f aɪ d
prediction	t ɑːɹ æ ɑː iː i ə eɪ ɛ θ ɹ iː k ɚ ɹ ɛ k ʃ ə z iə ɹ oʊ θ ɹ iː s ɛ v ə n ɹ eɪ n aɪ ɛ n t uː
delta echo charlie continue descent to flight level three five zero	
target	d ɛ l t ə ɛ k oʊ tʃ ɑːɹ l i k ə n t ɪ n j uː d ɪ s ɛ n t t ə f l aɪ t l ɛ v əl θ ɹ iː f aɪ v z iə ɹ oʊ
prediction	l ə k oʊ tʃ ɑːɹ l i k ə n t ɪ n j ɪ ə s ɛ n t ɑː l v θ ɹ iː f aɪ v z iə ɹ
merair six nine five two climb to flight level two nine zero	
target	m ɛ ɹ ɛɹ s ɪ k s n aɪ n f aɪ v t uː k l aɪ m t ə f l aɪ t l ɛ v əl t uː n aɪ n z iə ɹ oʊ
prediction	ɛɹ ɹ s ɪ k t n f aɪ v t uː k l aɪ m t uː
french air force five five one zero report your rate of descent	
target	f ɹ ɛ n tʃ ɛɹ f oːɹ s f aɪ v f aɪ v w ʌ n z iə ɹ oʊ ɹ ɪ p oːɹ t j ʊɹ ɹ eɪ t ʌ v d ɪ s ɛ n t
prediction	ɹ æ n d f oːɹ s f aɪ v f aɪ v w ʌ n z iə ɹ oʊ ɪ p oːɹ t j ʊɹ ɹ eɪ t v t ɪ s ɛ n d t
good afternoon lufthansa triple five zero squawk two seven five seven	
target	g ʊ d æ f t ɚ n uː n l ʌ f θ æ n s ə t ɹ ɪ p əl f aɪ v z iə ɹ oʊ s k w ɔː k t uː s ɛ v ə n f aɪ v s ɛ v ə n
prediction	g ʊ d æ f t ɚ n uː n l ʌ θ æ n s ə t ɹ ɪ p əl v aɪ v z iə ɹ oʊ s k w ɔː k t uː s ɛ v ə n f aɪ v s ɛ v ə n
britannia three two seven alfa contact rhein radar one three two decimal four	
target	b ɹ ɪ t æ n iə θ ɹ iː t uː s ɛ v ə n æ l f ə k ɑː n t æ k t ɹ eɪ n ɹ eɪ d ɑːɹ w ʌ n θ ɹ iː t uː d ɛ s ɪ m əl f oːɹ
prediction	eɪ ɜː θ ɹ iː t uː s ɛ v ə n æ l f ə k ɑː n t æ k ɹ eɪ n ɹ eɪ d ɑːɹ w ʌ n θ ɹ iː t uː oʊ ɛ ɹ ɪ v w ɑːɹ ə
tag lufthansa three five zero eight identified climb flight level two five zero	
target	t æ g l ʌ f θ æ n s ə θ ɹ iː f aɪ v z iə ɹ oʊ eɪ t aɪ d ɛ n t ɪ f aɪ d k l aɪ m f l aɪ t l ɛ v əl t uː f aɪ v z iə ɹ oʊ
prediction	t æ g l ʌ f θ æ n ə θ ɹ iː f aɪ v z iə ɹ oʊ eɪ t ɛ d ɛ n t ɪ f aɪ k l aɪ m f l aɪ t l ɛ v əl t uː f aɪ v z iə ɹ oʊ
lufthansa four six five two climb to flight level three four zero	
target	l ʌ f θ æ n s ə f oːɹ s ɪ k s f aɪ v t uː k l aɪ m t ə f l aɪ t l ɛ v əl θ ɹ iː f oːɹ z iə ɹ oʊ
prediction	ə æ n s ə f oːɹ s ɪ k s f aɪ v t uː k l aɪ m f l aɪ t l ɛ v əl θ ɹ iː f oːɹ z iə ɹ oʊ
airfrans zero zero nine november good afternoon squawk two seven four four	
target	ɛɹ f ɹ ə n z z iə ɹ oʊ z iə ɹ oʊ n aɪ n n oʊ v ɛ m b ɚ g ʊ d æ f t ɚ n uː n s k w ɔː k t uː s ɛ v ə n f oːɹ f oːɹ
prediction	ɛɹ f ɹ ə n z iə ɹ oʊ z iə ɹ oʊ n aɪ n oʊ v ɛ b ɚ g ʊ d æ f t ɚ n uː n s k ɔː k t uː s ɛ v ə n f oːɹ f oːɹ
roger aerovic one zero six one for separation turn right heading one nine five	
target	ɹ ɑː dʒ ɚ ɹ ɛ ɹ oʊ v ɪ k w ʌ n z iə ɹ oʊ s ɪ k s w ʌ n f oːɹ s ɛ p ɚ ɹ eɪ ʃ ə n t ɜː n ɹ aɪ t h ɛ d ɪ ŋ w ʌ n n aɪ n f aɪ v
prediction	ɹ ɑː dʒ ɚ ɛ ɹ oʊ v ɪ k w ʌ n z iə ɹ oʊ s ɪ k s k z w ʌ n n f oːɹ s ə v p ɹ eɪ ʃ ə n t ɜː n ɹ aɪ t h ɛ n ɪ ŋ

5.5 Whisper

The results presented in Table 5.15 are directly comparable with the other methods utilized in our research. It is discernible that architecture complexity correlate with increased performance and WER score. However, it is also noticeable that the momentum of improvement begins to diminish with increased parameters. This is evident by a substantial improvement of 14.8 percentage points when transitioning from a small to medium architecture but a significantly smaller gain of only 1.7 percentage points from medium to large.

Table 5.15: Whisper performance comparison.

Model	Original WER	Corrected WER
small	91.1%	46.0%
medium	84.2%	31.2%
large	82.9%	29.5%

The original Whisper output was slightly corrected to create a comparable transcript to better match the other output formats, as shown in Table 5.16. Again we see the same type of errors as mentioned in Section 5.2 and 5.3.

Table 5.16: Whisper (small) output samples.

Original prediction	Corrected prediction	WER
swissair six five five two contact milan one three four five two good bye		
Swiss 6552, contact Milan 13452, goodbye.	swiss six five five two contact milan one three four five two goodbye	21%
lufthansa four five seven five good morning radar contact continue climb flight level three two zero willisau trasadingen nelli		
Lufthansa 4575, good morning radar contact, country or client flight L320, release our trials at Nellie.	lufthansa four five seven five good morning radar contact country or client flight l three two zero release our trials at nellie	163%
tunair four eight zero two is identified youre cleared direct to fox fox mike		
Turn F4802 as identified, Jekyllert, Directive Fox, Fox, Mike.	turn f four eight zero two as identified jekyllert directive fox fox mike	50%
monarch two four four bonjour squawk five seven three seven		
Monarch 244, Mojo Squawk 5737	monarch two four four mojo squawk five seven three seven	10%
p s a five two six hotel contact rhein one three two decimal four		
BASTA 526 VATEL, Kontakt Rhein-Wandri-2-Desmal-4	basta five two six vatel kontakt rhein-wandri- two-desmal- four	71%

Table 5.17: Whisper (large) output samples.

Original prediction	Corrected prediction	WER
swissair six five five two contact milan one three four five two good bye		
Swissair 6552 contact Milan 13452, goodbye.	swissair six five five two contact milan one three four five two goodbye	14%
lufthansa four five seven five good morning radar contact continue climb flight level three two zero willisau trasadingen nelli		
Lufthansa 4575, good morning, radar contact, congeal climb flight level 320, release our trials at Inleli.	lufthansa four five seven five good morning radar contact congeal climb flight level three two zero release our trials at inleli	142%
tunair four eight zero two is identified youre cleared direct to fox fox mike		
Tower 4802 is identified, you're clear to direct to FOX, FOX Mike	tower four eight zero two is identified youre clear to direct to fox fox mike	21%
monarch two four four bonjour squawk five seven three seven		
Monarch 244, Możeszkow 5 7 3 7	monarch two four four możeszkow five seven three seven	20%
p s a five two six hotel contact rhein one three two decimal four		
Belfast 526H, contact R132.4	belfast five two sixh contact r one three two four	50%

Chapter 6

Discussion

The main focus of this research endeavor has been to investigate and assess various methods for ASR tailored explicitly for ATC audio. The ultimate goal is to equip Kongsberg Defence & Aerospace (KDA) with comprehensive insights to facilitate informed decision-making for its ongoing development of remote towers systems (RTS). To accomplish this, we have conducted experiments on publicly available corpora containing ATC audio recordings in English.

By leveraging the methodologies outlined in Chapter 4 and analyzing the outcomes presented in Chapter 5, we will attempt to answer the research questions proposed in Section 1.2:

RQ 1 What are the current State-of-the-Art technologies for ATC ASR, and how can we effectively implement these?

Drawing upon the observations and trends discussed in Sections 3.5 and 3.6, we can confidently assert that wav2vec 2.0 is the prevailing state-of-the-art technology for ASR, transcending domain-specific boundaries. Our experiments have achieved exceptional results, attaining a WER below 1% on ATCOSIM by employing the general fine-tuning approach to develop a downstream model customized for this particular domain.

These findings substantiate the efficacy and prominence of wav2vec 2.0 as a cutting-edge solution for ATC-specific ASR applications. It showcases the model's ability to deliver exceptional accuracy and robust performance, surpassing previous methodologies in the field. By leveraging the strengths of fine-tuning techniques and domain-specific adaptations, the proposed approach ensures optimal ASR performance, increasing the efficiency and effectiveness of ATC systems.

Having pinpointed what we regard as the most sophisticated and relevant technology for ASR in ATC, we turn to the second research question:

RQ 2 Which solutions meet the requirements for implementing ASR in KDA RTS?

We also encounter a significant safety concern against the backdrop of impressive results obtained through ready-to-use frameworks such as the Hugging Face interface and pipelines, which use pre-trained models from Meta Ai's GitHub repository and the Hugging Face website itself. Given the large scale of these frameworks, it necessitates a high level of expertise to navigate the source code concerning security. Thus, to address the second research question, we will to the best of our abilities, present the concerns that arose during our research.

6.1 Deep Speech 2

Despite its age, the Deep Speech 2 architecture still performs satisfactorily in certain scenarios. The original paper described in Section 3.2.1 reports a WER of 5.15% on the LibriSpeech test-

clean set, which is on par with the results of 5.4% we observed when training on the entire Hiwire test sub-corpus and 3.65 percentage points less than what we observed on the ATCOSIM test sub-corpus at 8.8%.

However, the Deep Speech 2 model exhibits limitations when faced with out-of-domain audio and noisy data, even within its specialized domain. Notably, when exposed to the high noise sub-corpus of Hiwire, the performance deteriorates significantly, resulting in a WER of 21.3%. This decline in performance highlights the model's vulnerability to noise and raises concerns regarding its effectiveness in handling out-of-domain data. While overfitting is generally not desirable, in some instances, such as specialized domains like ATC, it may be a consideration worth exploring, as robustness may be of lesser concern when dealing with out-of-domain data.

Furthermore, cross-domain evaluation of the models reveals that training on the entire corpus outperforms training on individual sub-corpora. The corpus size plays a crucial role, as models trained on individual sub-corpora achieve WERs ranging from 29.2%

Regarding the second research question (RQ2), the Deep Speech 2 model initially appears to be a suitable choice due to its simplicity and straightforward implementation. Its design, relying on the TensorFlow framework, allows for the straightforward integration of an ASR module into a system, where it can process audio inputs and generate transcripts with minimal additional processing. However, the reliance on a third-party framework necessitates a thorough vetting process to address potential safety concerns, as outlined in RQ2.

Considering the unique characteristics of ATC communications, the limited availability of training data, and the substantial requirements of supervised learning approaches that demand large amounts of annotated data, the use of Deep Speech 2 in KDAs RTS does not appear feasible, both when considering costs to transcribe the amount of training data required manually, and considering the existence of more advanced ASR solutions available today.

Deep Speech 2 holds historical significance as one of the first end-to-end models available for implementation (provided the corpus is correctly annotated). It is fair to say it served as a stepping stone in the evolution of such systems. However, considering the technological leaps made since this architecture was considered state-of-the-art, its primary purpose now seems to be an introduction to the end-to-end ASR systems powering today's digital assistants.

6.2 Transformer

Like Deep Speech 2 (DS2), the transformer is not a new technology, as its core principles were described in 2017. It was initially intended to solve textual challenges through sequence-to-sequence learning but has proven to handle most types of sequence-to-sequence tasks with great success.

The results we observed in Chapter 5 show some overall trends. ATCOSIM and Hiwire clean are comparable in noise levels as they are both simulated data recorded in controlled environments. However, they differ in sentence and word complexity, which creates a significant difference in the prediction ability of the transformer. The same pattern applies when we look at the noise levels on the Hiwire results, indicating increased noise correlation with decreasing WER. It also observed that using MFCC as input improves performance on the corpora-specific models, with minor improvements on the clean models, respectively 1.6% WER on ATCOSIM and 2.2% WER on Hiwire clean to a 12% WER improvement on Hiwire HN, indicating significant impact on noisy data.

However, when observing models trained on both corpora, they encounter difficulties in establishing a baseline as the data becomes more diverse. Mainly, MFCCs appear susceptible to this effect, exhibiting a substantial decrease in predictive proficiency compared to the STFT

input. This results in a stark 73.7% WER disparity on unseen data when juxtaposed with the STFT.

Notably, the transformer's overall performance on the two corpora is worse than that of the results observed with Deep Speech 2. However, the transformer is far better at forming coherent words within a few training epochs, as shown in Table 6.1.

Table 6.1: Transformer early prediction development.

Epoch	Target	Prediction
1	range one hundred sixty	request direct to sixto fo fo atherat fo fo thr
-	lufthansa four two one nine is identified	alitalia thre zero one o t one tone tonee threee two threre
-	delta fox sierra geneva one three three one five bye	alitalia three two five o t onent o onee threee threee three thro
5	aero lloyd five three three continue left turn heading of zero two zero	aero lloyd five two two two all three two zero zero aint

It must be noted that DS2 uses a CTC loss function, while the transformer uses a cross-categorical loss function, which may have impacted the results we observed. The complexity of the corpora seems to correlate as they both perform better on Hiwire than ATCOSIM.

Considering the research question (RQ2), the transformer architecture used in our experiments is built with the TensorFlow framework, and as such, the same security concerns apply here as with DS2. While more complex in architecture and implementation than DS2, the transformer as a standalone module could also be implemented without additional adaption. It is also possible to further research supervised down streaming, but as with DS2, this will require a substantial amount of quality checked labeled data and highly specific for the domain.

6.3 wav2vec 2.0

As mentioned in Section 4.6, the initial premise for this thesis was the delivery of a substantial package of unlabelled audio from one of KDAs partners. After identifying wav2vec 2.0 as the most promising solution in terms of access and usability to solve KDAs problem as mentioned in Chapter 1, our initial approach was to utilize this framework to create our own, novel ATC base model using this unlabelled data for a downstream task within the same domain, and perform experiments with various amounts of labeled data.

Due to unforeseen circumstances regarding legal concerns, the partner could not deliver the audio data. This was not ideal and changed the nature of our research going forward. Our options at the time were to start building a custom corpus containing ATC audio scraped from sources like YouTube and various aviation enthusiasts' websites, with all the potential problems and difficulties this would entail both legally and technically. This option was quickly scrapped, and instead, a compromise was made utilizing the aforementioned public corpora, which would ensure our experiments could be measured against previous research and, above all, be reproducible by others.

6.3.1 Hugging Face

Initially, the plan was to train pre-trained models using the Hugging Face framework on both corpora. However, when reading the fine print in the Hiwire EULA, this did not align with the Hugging Face terms of service [2]:

- If you decide to create an Account for your Organization, you represent that you have the authority to act on behalf of your Organization and bind your Organization to these Terms.
- You own the Content you create! We will not sell your Content, nor will we use it in any other way as permitted under these Terms. *However*, by posting your Content or otherwise making it available on our Website, you must be aware that:
  - You hereby grant us a worldwide, royalty-free, and non-exclusive license to use, display, publish, reproduce, distribute, and make derivative works of such Content to provide Services and as otherwise permitted under these Terms and our Privacy Policy; and,
  - Your Content will be viewed by others, and therefore:
    * If you decide to set your Repository private, we will use reasonable and appropriate measures designed to keep your Content confidential and protected from any unauthorized access or disclosure. However, we may access or share your private information pursuant to the terms set forth in our Privacy Policy.

As it is better to err on the side of caution, the experiments were conducted solely on the ATCOSIM corpus, which is open-source and free to use.

As evident from the results presented in Section 5.3, the performance of pre-trained wav2vec models is superior to both DS2 and basic transformer architectures when tested on unseen data. Based on these observations, we are confident that the state-of-the-art solution regarding RQ1 is anchored in downstream models produced by fine-tuning much larger pre-trained models like the wav2vec2-large-960h-lv60-self.

Unlike DS2 and Transformer, wav2vec 2.0 uses the raw wave plot as input to the feature extraction layer, eliminating the need to pre-process the audio. When compared with the basic transformer's performance (or lack thereof) when using MFCCs as a basis for feature extraction on the combined ATCOSIM and Hiwire corpora, it is possible that MFCC as features make the model handle diversity such as noise levels and sentence complexity in a less than optimal way.

Wav2vec 2.0 produces near-perfect results with the best model predicting with sub-one percent precision. The results also demonstrate increased robustness when the data is augmented and tested on real ATC, but also show reduced capabilities on unseen data from the same source as the fine-tuning pool. The employed cycles of 50 epochs showcase the framework's fast adaptation of the domain-specific language. Furthermore, the few errors produced exhibit the same characteristics described in the transformer results.

As for the security aspect of using a third-party framework, Hugging Face does not meet the requirements described in RQ2, but the results are firmly in favor of full implementation and further testing on actual ATC data utilizing the underlying wav2vec 2.0 framework created and managed by Meta AI.

6.4 wav2vec-U 2.0

At first glance, this framework and its presentation seemed to provide a feasible solution to the problem facing KDA and its partners: having a substantial amount of unlabeled audio. The core concept of wav2vec-U 2.0 (w2v-U) is the unsupervised approach to solving the common problem in ASR related to domains with little to no transcribed audio. With the limitations and challenges associated with ASR in the ATC domain, as described in 3.6, this framework

could potentially help lower the cost of implementing an ASR system as described in Chapter 1 with no labeled data necessary. However, development of the framework appears to be on hiatus, with its latest commit tracing back to June 15th, 2022, as evidenced by this comment at the end of the readme file:

> *** Note: these instructions are a work in progress and will be updated over the next few days

With our own experience working with this framework in mind, the sheer complexity and severe lack of documentation might explain why mainstream adaption for the time being is lacking, given the apparent benefits the framework promises. From an academic point of view, there seems to be some traction in adopting the concept (Section 3.5), with nine publications in 2022, followed by fourteen so far in 2023, according to Google Scholar.

The enthusiasm for potential generative ASR modeling soon turned into frustration as the cascade of challenges started to mount, with what appeared to be two steps back for every progress made. In Section 4.6.2, we describe the step-by-step processes. However, we have omitted the numerous error messages, dependency requirements, code deficiencies, and lack of documentation for each data processing step, challenges we nicknamed "The Framework Fairy." During training runs using the wav2vec-U framework, the use of GPU memory would frequently approach the limit, and on some occasions, the GPU would stall, throwing system-wide error messages like the one in Listing 4. This would cause our processes running on the GPU to terminate, and other users' processes would also be dropped (as we experienced first-hand).

```
kernel:[259395.394258] watchdog: BUG: soft lockup - CPU#159 stuck for
↪  22s! [fairseq-hydra-t:2262736]
```

Listing 4: System wide error message.

Even though both authors were using the same server to configure the framework per the instructions, the path we undertook was quite different toward the end goal. In order to streamline this process, we created a layer on top of the framework, listed in Section 4.6.2, automating the preparation of both text and audio data, preprocessing the input to the GAN and initializing the training cycles in a one-click setup.

Each training cycle with the hardware described in Section 4.2.1 would typically last 57 hours, not including the steps involved in preparing the data. Given the relatively small size of the ATCOSIM corpus, the total amount of GPU RAM used would be around 30GB, give or take. In the scheduled time frame, the time allocated for researching the unsupervised approach was set to eight weeks starting in February; due to the abovementioned challenges, the actual training did not commence until early April, lasting approximately three weeks. During this period, we would, at any given time, have four different models training simultaneously, with some downtime due to other users needing access to the GPUs.

The results presented in Section 5.4 are from the best-performing models from each of the approaches that ultimately led to the framework running on our system. They are the culmination of countless attempts to get the WER down to an acceptable level, with a trial-and-error approach as the only viable option, given the lack of documentation and support in the GitHub repo. Compared with the basic Transformer model at 28.4% using the supervised approach, the

unsupervised result of 23.7% shows potential for further research and development.

Compared with downstream models the w2v-U framework cannot compete and, as such is not currently state of the art, even though the underlying technology is the same. Second, the fact that this framework is challenging even to get working means it is not ready for implementation, neither streaming nor offline. Regarding security, multiple potential risks are associated with a highly experimental framework like this, with many dependencies, some more obscure than others.

6.5 Whisper

OpenAI Whisper, described in Section 4.7, builds on the transformer architecture with a new approach to data collection and labeling called weak supervision, resulting in a 680.000 hours base for the pre-trained models. The results are based on three different architectures spanning from small to large, describing the number of parameters in the architecture.

The top-performing model registers a WER of 29.5%. When juxtaposed with the best-performing Transformer model that employs the MFCC input, as shown in Table 5.7, the Transformer exhibits a WER that is 1.1 percentage points lower. This is particularly intriguing when considering the data platform used and the architectural depth of each model. With the resources spent on achieving these comparable results, Whisper and the basic Transformer are at opposite ends of the spectrum. Whisper demonstrates a general ASR model's robustness when presented with presumed unseen data. We say presumed because OpenAI has not disclosed what the 680.000 hours of training data consist of in detail. It is fair to assume that publicly available corpora, like ATCOSIM, are included in this massive pool of audio data. Therefore, it may have seen the data previously, yet it performs on the same level as a basic Transformer trained with less than 10 hours of labeled domain-specific data.

Regarding RQ1, Whisper can not be considered state-of-the-art with the results presented here. It is somewhat robust, but this could also be explained by the mysterious corpus they utilize, the contents of which remain unknown as of the submission date of this thesis paper. However, of all the methods tested, it is by far the simplest to implement, though at a considerable cost regarding time, as it would take upwards of one hour to predict the 954 samples in the test split. Implementation would result in a running cost currently set at 0.006$ per minute of transcribed audio and does not support streaming.

As for the security concerns in RQ2, using a black box API like Whisper does not conform with the strict requirements listed in Section 1.2. Given that the other methods produce almost instant predictions leads us to suspect that OpenAI is doing more behind the scenes than they let on, but this would be pure speculation on our part. In any event, the many open questions regarding OpenAIs practice make Whisper an unsuited candidate for further consideration in the remote towers systems (RTS) project, but rather as a benchmark of the abilities of one of the leading AI companies in the world.

6.6 Hardware resources needed

To get a sense of scale when discussing the hardware resources needed to create pre-trained models like the ones described in Section 5.3, let us take a closer look at what Meta AI has built over the last decade as described by themselves [1] [2].

Since establishing the Facebook AI Research lab in 2013, Meta has made notable advancements in AI, particularly in self-supervised learning and transformers. This focus has driven the de-

[1] https://ai.facebook.com/blog/ai-rsc/

[2] https://ai.facebook.com/blog/supercomputer-meta-research-supercluster-2023/

velopment of high-performance computing infrastructures necessary for training large models. The first iteration, crafted in 2017, was a single cluster with 22,000 NVIDIA V100 Tensor Core GPUs, capable of 35,000 training jobs a day.

A strategic decision in 2020 set the course for a technological overhaul. A new computing infrastructure was designed from scratch to exploit the latest GPU and network fabric technologies. This enabled the handling of massive corpora up to an exabyte in size and training models with over a trillion parameters. The resulting system, the Research SuperCluster (RSC), comprised 760 NVIDIA DGX A100 systems, equating to 6,080 GPUs. This represented a substantial upgrade from the V100 GPUs used previously. A high-performance NVIDIA Quantum 1600 Gb/s InfiniBand fabric facilitated rapid inter-GPU communication. The system also incorporated a storage tier of 175 petabytes of Pure Storage FlashArray, 46 petabytes of cache storage in Penguin Computing Altus systems, and 10 petabytes of Pure Storage FlashBlade.

By May 2023, Meta completed the second-phase buildout of RSC, significantly enhancing its capabilities and positioning it among the fastest AI supercomputers globally. This expansion integrated 2,000 NVIDIA DGX A100 systems or 16,000 NVIDIA A100 Tensor Core GPUs, all linked via a 16 Tb/s NVIDIA Quantum InfiniBand fabric network. The computing power reached nearly 5 exaflops, meaning quintillions of calculations per second. In parallel with these upgrades, the storage solution named AirStore was refined. Developed in collaboration with Penguin Computing and Pure Storage, AirStore featured 80 petabytes of cache and over half an exabyte of bulk storage, delivering up to 16TB/s of throughput. This secure, scalable solution was vital to managing the vast data inflow, ensuring sustained training for projects running on thousands of GPUs over weeks, an essential factor in the success of RSC.

In comparison, the hardware resources available at HiØ, as described in Section 13, consists of two A100 80GB GPUs. Each of these currently retails at 220,000NOK / 20,500$ inc. VAT. Of course, the RSC is not solely for ASR research and is designed to meet the demands of one of the biggest tech companies in the world. However, the main point here is that for Kongsberg Defence & Aerospace (KDA), the scale of resources needed correlates with implementation methods and the corpora size. Suppose a novel supervised or self-supervised ATC base model is desired. In that case, large amounts of audio data must be processed and require substantial hardware capabilities compared to a downstream model built from pre-trained wav2vec 2.0 weights.

6.7 Data

One of the most vital parts of machine learning is access to relevant, quality data. Without it, even the most advanced algorithms cannot produce reliable results, and though this can be mitigated through various tricks and techniques, like data augmentation and synthetic data, for a sub-field like ASR, the access to relevant data is paramount. For instance, in object detection in the field of computer vision, labeling a custom corpus of images in the thousands can be conducted in a matter of hours with relative ease using purpose-built tools like Supervise.ly[3] or outsourcing to third party companies. Audio differentiates from other data types due to the time-consuming procedure of obtaining accurate transcripts matched to their source. As an example, keeping within the air traffic domain, consider the process needed to label 100 images of airplanes with a simple box around the object for identification versus transcribing 1 hour of audio which will require repeated start and stop in playback in order to correctly transcribe the content, even for high quality and clean audio. It will naturally be even more time-consuming when handling noisy data.

If we assume that transcribing 1 hour of data can be done in 1.5 hours by a professional, then we can ruffly estimate the cost attached to transcribing 1000 hours of audio. A full-time position

[3] `https://supervisely.com`

in Norway is set to 1750 hours of effective work, and the average salary is 637,000 NOK as of 2023[4]. This gives us a rough estimate of $\frac{(637{,}000)}{1750} * (1000 * 1.5) = 546,000 NOK$.

The state-of-the-art methods described in Chapter 4 focus on their performance on common, large-scale corpora and the race towards reaching sub 1% WER, but also on reusing large, pre-trained models for domain-shifted ASR which can potentially help alleviate the costly and tiresome process of manually labeling audio data.

However, it also appears that ASR has reached a pivotal point in terms of complexity 4.6.2 and data/hardware resources needed for further advancement, centralizing the development to a few tech giants. This can be problematic regarding RQ2, in which we are rapidly approaching a situation where implementation of ASR is entirely dependent on third-party actors.

6.8 Future work

After a thorough exploration, the recommended strategy for Kongsberg Defence & Aerospace (KDA) involves leveraging the substantial progress achieved by Meta AI in robust ASR model development. This recommendation is drawn from our experiments on ATC-specific audio and widespread academic agreement, which point to wav2vec 2.0's unparalleled downstream capabilities in areas with limited resources like ATC.

However, there could be challenges when using large pre-trained models as a starting point and adopting a fine-tuning downstream approach. These could include security risks and the need to keep up with rapidly advancing AI technology. It is important to note that all technologies used in this research are open-source. This brings several advantages, such as reduced development costs, faster time-to-market, and ongoing community-driven enhancements and debugging, which may improve robustness and security. However, open-source also comes with some drawbacks. Remote towers systems (RTS), as a commercial product, might face potential licensing issues and intellectual property complexities due to the use of open-source technology. Changes to the frameworks could demand substantial maintenance, support, and compatibility resources. The source code is publicly available, so it could expose security vulnerabilities if not properly managed.

Regardless of the strategy KDA chooses, high-quality data is a non-negotiable requirement for ASR development. Both self-supervised and unsupervised approaches require substantial ATC audio from KDAs partners. While unsupervised learning has shown potential, it is still in its early stages, and a fully functional end-to-end system that inputs an unlabelled audio signal and outputs an accurate text transcript rather than phonemes is unlikely anytime soon. The self-supervised approach similarly requires transcribed data for fine-tuning, and as for traditional architectures using supervised learning, transcribed data is integral.

As usual, the problem and solution are the amount of data available. Large amounts of data and some human involvement in the transcription process are inevitable. However, considering the high accuracy achieved using wav2vec 2.0 pre-trained models with limited data for fine-tuning, some parts of this process could be automated, effectively creating an annotation tool. If used in an offline sandbox environment, this would mitigate some of the concerns related to RQ2.

Expanding on the concept first presented in Section 4.6, this would involve using transcripts produced by wav2vec pre-trained models from actual ATC audio, which undergoes a quality control through human supervision, where errors found in the ASR transcripts are corrected and then added to a main database as verified samples. This process is repeated with continuous fine-tuning of the model, and as the number of verified samples increases, the model becomes more accurate, ultimately eliminating human supervision and becoming autonomous.

[4] `https://www.ssb.no`

Considering the straight-forward design of both Deep Speech 2 and the transformer, and the fact that these architectures can be built and controlled almost from scratch with either the PyTorch or the TensorFlow back-end, we would also recommend KDA to keep the option open for parallel development of supervised learning based models with potential downstream capabilities utilizing the main verified corpus (Figure 6.1).

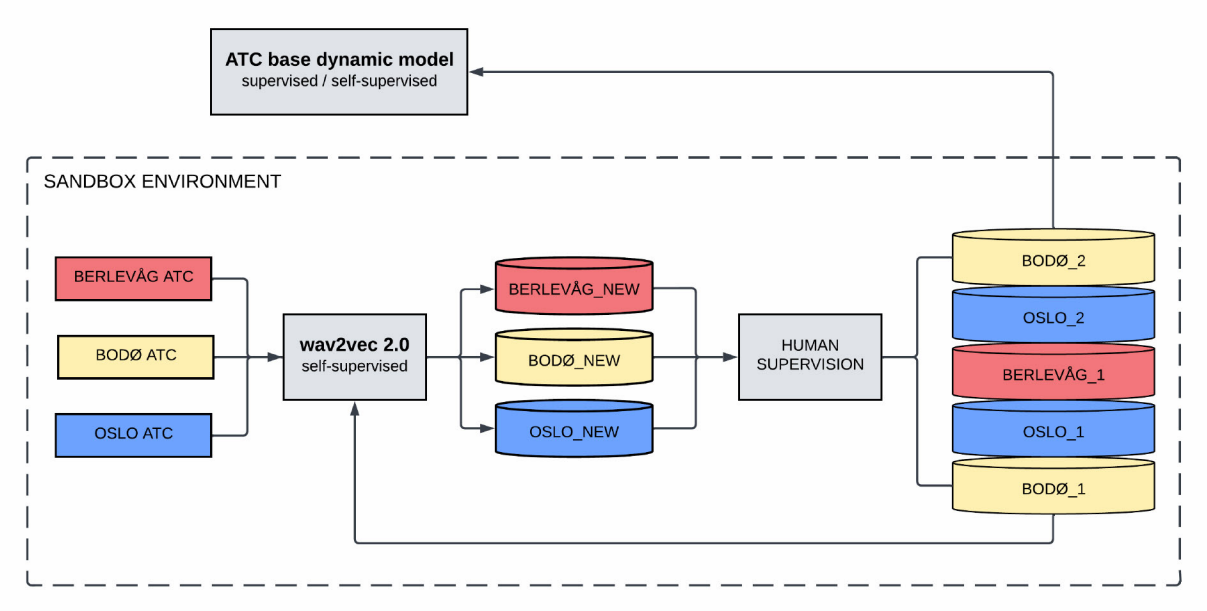


Figure 6.1: Proposed ASR setup.

Even if a robust, all-knowing model may not be possible with the current tech, the fine-tuning of large models with very little specific audio in connection with smaller airports seems like the most viable path forward, similar to how the wav2vec 2.0 pre-trained models described in Section 4.6.1 were created. These fine-tuning steps could be down to the individual ATC operator level but will, of course, require considerable research and development to distill into the streamlined, easy-to-use process we envision.

Chapter 7

Conclusion

The initial motivation of this thesis was to investigate the possibility of implementing automatic speech recognition (ASR) within the air traffic control (ATC) domain, specifically for the remote towers systems (RTS) made by Kongsberg Defence & Aerospace (KDA). KDA has expressed interest in enhancing RTS capabilities using real-time transcriptions of pilot-controller audio communication, augmented with visual cues. This integration of ASR and natural language processing (NLP) intends to lessen the controller's cognitive load, boost situational awareness, improve safety, and efficiently handle increased air traffic across multiple airports.

At the start of this project, the audio data needed for conducting research and experiments were expected to be delivered by one of KDAs partners by mid-November. Due to unforeseen legal circumstances, the data was not possible to obtain. As a result, we had to use the open-source publicly available ATC corpus ATCOSIM for most of our research, along with the Hiwire corpus available for purchase for academic and commercial use. The corpora contain simulated ATC audio, with Hiwire having augmented levels of noise infusion divided into four sub-corpora, giving us a total of $\approx$ 40 hours of transcribed audio to conduct our research.

Two research questions are discussed in this thesis. First, we identify the current state-of-the-art technologies within the field of ASR today and provide a possible solution for implementation in KDAs RTS. The second question investigates the potential security implications that may arise when utilizing the technologies identified in the first research question. The advantages and potential drawbacks of implementing these technologies are critically assessed, providing a balanced view of their practical application.

In our work, we have explored various methods to solve the challenge described in research question one, with limited data, and have identified downstreaming from large corpus pre-trained models as the best candidate. We also utilize a novel, generative framework called wav2vec-Unsupervised 2.0 for low-resource domains in which unlabeled audio can be transcribed using a GAN network with unassociated textual data, the only requirement being that both audio and text must be in the same language. Furthermore, we present a solution for automating the unsupervised training process, reducing implementation time significantly, and providing a (by comparison) far more streamlined process of instantiating unsupervised training in the ASR domain.

Even though the results we achieved are more than acceptable regarding WER, one must remember that these results are from simulated data only and that real-world ATC will be far more diverse and challenging. Nevertheless, we believe the research presented in this thesis can serve as a general guide for which direction Kongsberg Defence & Aerospace (KDA) should take in their future endeavors.

In closing, our main findings from this thesis have been accepted by the 26^{th} International

Conference of Text, Speech, and Dialogue (TSD2023)[1] and will be published by Springer in their *Lecture Notes in Artificial Intelligence* series[2]. The conference paper is listed in Appendix A.

[1] https://www.kiv.zcu.cz/tsd2023/index.php
[2] https://www.springer.com/gp/computer-science/lncs

Bibliography

[1] Huggingface platform. `https://huggingface.co`. Accessed: 2021-11-30.

[2] Huggingface terms of service. `https://huggingface.co/terms-of-service`. Accessed: 2023-05-30.

[3] Ibm shoebox. `https://www.ibm.com/ibm/history/exhibits/specialprod1/specialprod1_7.html`.

[4] Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, et al. Deep speech 2: End-to-end speech recognition in english and mandarin. In *International conference on machine learning*, pages 173–182. PMLR, 2016.

[5] Rosana Ardila, Megan Branson, Kelly Davis, Michael Henretty, Michael Kohler, Josh Meyer, Reuben Morais, Lindsay Saunders, Francis M. Tyers, and Gregor Weber. Common voice: A massively-multilingual speech corpus, 2020.

[6] Bishnu S Atal and Suzanne L Hanauer. Speech analysis and synthesis by linear prediction of the speech wave. *The journal of the acoustical society of America*, 50(2B):637–655, 1971.

[7] Civil Aviation Authority. Cap 413: Radiotelephony manual. *Civil Aviation Authority*, 2020.

[8] Sandeep Badrinath and Hamsa Balakrishnan. Automatic speech recognition for air traffic control communications. *Transportation research record*, 2676(1):798–810, 2022.

[9] Alexei Baevski, Wei-Ning Hsu, Alexis Conneau, and Michael Auli. Unsupervised speech recognition. *Advances in Neural Information Processing Systems*, 34:27826–27839, 2021.

[10] Alexei Baevski, Henry Zhou, Abdelrahman Mohamed, and Michael Auli. wav2vec 2.0: A framework for self-supervised learning of speech representations, 2020.

[11] J Baker. Dragondictate–30k: Natural language speech recognition with 30,000 words. In *Proc. European Conf. on Speech Technology, Bd*, volume 2, pages 161–163, 1989.

[12] Leonard E Baum, Ted Petrie, George Soules, and Norman Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. *The annals of mathematical statistics*, 41(1):164–171, 1970.

[13] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[14] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006.

[15] Mohamed Reda Bouadjenek. Automatic speech recognition using ctc. `https://github.com/keras-team/keras-io/blob/master/examples/audio/ctc_asr.py`, 2021.

[16] Herve A Bourlard and Nelson Morgan. *Connectionist speech recognition: a hybrid approach*, volume 247. Springer Science & Business Media, 1994.

[17] Stanley F Chen and Joshua Goodman. An empirical study of smoothing techniques for language modeling. *Computer Speech & Language*, 13(4):359–394, 1999.

[18] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[19] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[20] Kenneth Ward Church. A stochastic parts program and noun phrase parser for unrestricted text. In *International Conference on Acoustics, Speech, and Signal Processing,*, pages 695–698. IEEE, 1989.

[21] Christopher Cieri, David Miller, and Kevin Walker. The fisher corpus: a resource for the next generations of speech-to-text. In *International Conference on Language Resources and Evaluation*, 2004.

[22] E. F. Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377–387, jun 1970.

[23] Ken H Davis, R Biddulph, and Stephen Balashek. Automatic recognition of spoken digits. *The Journal of the Acoustical Society of America*, 24(6):637–642, 1952.

[24] Steven Davis and Paul Mermelstein. Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE transactions on acoustics, speech, and signal processing*, 28(4):357–366, 1980.

[25] Li Deng and Douglas O'Shaughnessy. *Speech processing: a dynamic and optimization-oriented approach.* CRC Press, 2003.

[26] Wenqiang Du, Yikeremu Maimaitiyiming, Mewlude Nijat, Lantian Li, Askar Hamdulla, and Dong Wang. Automatic speech recognition for uyghur, kazakh, and kyrgyz: An overview. *Applied Sciences*, 13(1):326, 2022.

[27] Sean R Eddy. What is a hidden markov model? *Nature biotechnology*, 22(10):1315–1316, 2004.

[28] Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

[29] John S Garofolo, Lori F Lamel, William M Fisher, Jonathan G Fiscus, and David S Pallett. Darpa timit acoustic-phonetic continous speech corpus cd-rom. nist speech disc 1-1.1. *NASA STI/Recon technical report n*, 93:27403, 1993.

[30] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. *Neural computation*, 12(10):2451–2471, 2000.

[31] John J. Godfrey. Air traffic control complete ldc94s14a. web download., 1994.

[32] John J Godfrey, Edward C Holliman, and Jane McDaniel. Switchboard: Telephone speech corpus for research and development. In *Acoustics, Speech, and Signal Processing, IEEE International Conference on*, volume 1, pages 517–520. IEEE Computer Society, 1992.

[33] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning.* MIT press, 2016.

[34] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[35] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connec-

tionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on Machine learning*, pages 369–376, 2006.

[36] Anmol Gulati, James Qin, Chung-Cheng Chiu, Niki Parmar, Yu Zhang, Jiahui Yu, Wei Han, Shibo Wang, Zhengdong Zhang, Yonghui Wu, et al. Conformer: Convolution-augmented transformer for speech recognition. *arXiv preprint arXiv:2005.08100*, 2020.

[37] Dongyue Guo, Zichen Zhang, Bo Yang, Jianwei Zhang, and Yi Lin. Boosting low-resource speech recognition in air traffic communication via pretrained feature aggregation and multi-task learning. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2023.

[38] R. Hadsell, S. Chopra, and Y. LeCun. Dimensionality reduction by learning an invariant mapping. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 1735–1742, 2006.

[39] Awni Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, et al. Deep speech: Scaling up end-to-end speech recognition. *arXiv preprint arXiv:1412.5567*, 2014.

[40] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine*, 29(6):82–97, 2012.

[41] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[42] Konrad Hofbauer, Stefan Petrik, and Horst 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, May 2008. European Language Resources Association (ELRA).

[43] Fumitada Itakura. Line spectrum representation of linear predictor coefficients of speech signals. *The Journal of the Acoustical Society of America*, 57(S1):S35–S35, 1975.

[44] Frederick Jelinek. *Statistical methods for speech recognition*. MIT press, 1998.

[45] Zuluaga-Gomez Juan, Petr Motlicek, Qingran Zhan, Rudolf Braun, and Karel Vesely. Automatic speech recognition benchmark for air-traffic communications. Technical report, ISCA, 2020.

[46] Biing Hwang Juang and Laurence R Rabiner. Hidden markov models for speech recognition. *Technometrics*, 33(3):251–272, 1991.

[47] Daniel Jurafsky and James H Martin. Speech and language processing: An introduction to speech recognition, computational linguistics and natural language processing. *Upper Saddle River, NJ: Prentice Hall*, 2008.

[48] J. Kahn, M. Rivière, W. Zheng, E. Kharitonov, Q. Xu, P.E. Mazaré, J. Karadayi, V. Liptchinsky, R. Collobert, C. Fuegen, T. Likhomanenko, G. Synnaeve, A. Joulin, A. Mohamed, and E. Dupoux. Libri-light: A benchmark for asr with limited or no supervision. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7669–7673, 2020.

[49] Dennis H Klatt. Review of the arpa speech understanding project. *The Journal of the Acoustical Society of America*, 62(6):1345–1366, 1977.

[50] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[51] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[52] Vladimir I Levenshtein et al. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710. Soviet Union, 1966.

[53] Guan-Ting Lin, Chan-Jan Hsu, Da-Rong Liu, Hung-Yi Lee, and Yu Tsao. Analyzing the robustness of unsupervised speech recognition. In *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8202–8206. IEEE, 2022.

[54] Yi Lin, Qin Li, Bo Yang, Zhen Yan, Huachun Tan, and Zhengmao Chen. Improving speech recognition models with small samples for air traffic control systems. *Neurocomputing*, 445:287–297, 2021.

[55] Zachary C Lipton, John Berkowitz, and Charles Elkan. A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv:1506.00019*, 2015.

[56] Alexander H Liu, Wei-Ning Hsu, Michael Auli, and Alexei Baevski. Towards end-to-end unsupervised speech recognition. In *2022 IEEE Spoken Language Technology Workshop (SLT)*, pages 221–228. IEEE, 2023.

[57] Christopher Manning and Hinrich Schutze. *Foundations of statistical natural language processing*. MIT press, 1999.

[58] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Interspeech*, volume 2, pages 1045–1048. Makuhari, 2010.

[59] Abdelrahman Mohamed, Hung-yi Lee, Lasse Borgholt, Jakob D Havtorn, Joakim Edin, Christian Igel, Katrin Kirchhoff, Shang-Wen Li, Karen Livescu, Lars Maaløe, et al. Self-supervised speech representation learning: A review. *IEEE Journal of Selected Topics in Signal Processing*, 2022.

[60] Brian CJ Moore. *An introduction to the psychology of hearing*. Brill, 2012.

[61] Cory Myers, Lawrence Rabiner, and Aaron Rosenberg. Performance tradeoffs in dynamic time warping algorithms for isolated word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 28(6):623–635, 1980.

[62] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[63] Apoorv Nandan. Automatic speech recognition with transformer. `https://github.com/keras-team/keras-io/blob/master/examples/audio/transformer_asr.py`, 2021.

[64] H. Nyquist. Certain topics in telegraph transmission theory. *Transactions of the American Institute of Electrical Engineers*, 47(2):617–644, 1928.

[65] Julian James Odell. *The use of context in large vocabulary speech recognition*. PhD thesis, Citeseer, 1995.

[66] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[67] Douglas O'shaughnessy. *Speech communication: human and machine*. Universities press, 1987.

[68] Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. Librispeech: an asr corpus based on public domain audio books. In *2015 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 5206–5210. IEEE, 2015.

[69] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318. Pmlr, 2013.

[70] Douglas B Paul and Janet Baker. The design for the wall street journal-based csr corpus. In *Speech and Natural Language: Proceedings of a Workshop Held at Harriman, New York, February 23-26, 1992*, 1992.

[71] Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukas Burget, Ondrej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlicek, Yanmin Qian, Petr Schwarz, et al. The kaldi speech recognition toolkit. In *IEEE 2011 workshop on automatic speech recognition and understanding*, number CONF. IEEE Signal Processing Society, 2011.

[72] Vineel Pratap, Qiantong Xu, Anuroop Sriram, Gabriel Synnaeve, and Ronan Collobert. MLS: A large-scale multilingual dataset for speech research. In *Interspeech 2020*. ISCA, oct 2020.

[73] O Veronika Prinzo, Alfred M Hendrix, and Ruby Hendrix. Pilot english language proficiency and the prevalence of communication problems at five us air route traffic control centers. Technical report, FEDERAL AVIATION ADMINISTRATION OKLAHOMA CITY OK CIVIL AEROMEDICAL INST, 2008.

[74] Lawrence R Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[75] Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. Robust speech recognition via large-scale weak supervision, 2022.

[76] D Raj Reddy. Speech recognition by machine: A review. *Proceedings of the IEEE*, 64(4):501–531, 1976.

[77] Meta Research. wav2vec unsupervised (wav2vec-u). https://github.com/facebookresearch/fairseq/tree/main/examples/wav2vec/unsupervised, 2023.

[78] Douglas A Reynolds and Richard C Rose. Robust text-independent speaker identification using gaussian mixture speaker models. *IEEE transactions on speech and audio processing*, 3(1):72–83, 1995.

[79] Hiroaki Sakoe and Seibi Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE transactions on acoustics, speech, and signal processing*, 26(1):43–49, 1978.

[80] Steffen Schneider, Alexei Baevski, Ronan Collobert, and Michael Auli. wav2vec: Unsupervised pre-training for speech recognition. *arXiv preprint arXiv:1904.05862*, 2019.

[81] J Segura, T Ehrette, A Potamianos, D Fohr, I Illina, PA Breton, V Clot, R Gemello, M Matassoni, and P Maragos. The hiwire database, a noisy and non-native english speech corpus for cockpit communication. Citeseer, 2007.

[82] Luboš Šmídl. Air traffic control communication, 2011. LINDAT/CLARIAH-CZ digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

[83] Julius O. Smith. *Spectral Audio Signal Processing*. `https://ccrma.stanford.edu/~jos/sasp/sasp-citation.html`. online book, 2011 edition.

[84] Stanley Smith Stevens, John Volkmann, and Edwin Broomell Newman. A scale for the measurement of the psychological magnitude pitch. *The journal of the acoustical society of america*, 8(3):185–190, 1937.

[85] Igor Szoke, Santosh Kesiraju, Ondrej Novotny, Martin Kocour, Karel Vesely, and Jan "Honza" Cernocky. Detecting english speech in the air traffic control voice communication, 2021.

[86] Zheng-Hua Tan, Achintya kr. Sarkar, and Najim Dehak. rvad: An unsupervised segment-based robust voice activity detection method. *Computer Speech Language*, 59:1–21, 2020.

[87] Jörgen Valk and Tanel Alumäe. Voxlingua107: a dataset for spoken language recognition, 2020.

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

[89] Changhan Wang, Morgane Rivière, Ann Lee, Anne Wu, Chaitanya Talnikar, Daniel Haziza, Mary Williamson, Juan Pino, and Emmanuel Dupoux. Voxpopuli: A large-scale multilingual speech corpus for representation learning, semi-supervised learning and interpretation, 2021.

[90] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[91] Ronald J Williams and David Zipser. Gradient-based learning algorithms for recurrent. *Backpropagation: Theory, architectures, and applications*, 433:17, 1995.

[92] J.P. Woodard and J.T. Nelson. An information theoretic measure of speech recognition performance. 1982.

[93] Cheng Yi, Jianzhong Wang, Ning Cheng, Shiyu Zhou, and Bo Xu. Applying wav2vec2. 0 to speech recognition in various low-resource languages. *arXiv preprint arXiv:2012.12121*, 2020.

[94] Juan Zuluaga-Gomez, Amrutha Prasad, Iuliia Nigmatulina, Seyyed Saeed Sarfjoo, Petr Motlicek, Matthias Kleinert, Hartmut Helmke, Oliver Ohneiser, and Qingran Zhan. How does pre-trained wav2vec 2.0 perform on domain-shifted asr? an extensive benchmark on air traffic control communications. In *2022 IEEE Spoken Language Technology Workshop (SLT)*, pages 205–212. IEEE, 2023.

Acronyms

AI artificial intelligence. 1

ASR automatic speech recognition. 1, 3, 4, 26, 30, 32, 34, 39, 46, 59, 64, 65, 68, 82, 87, 88, 93, 94, 96

ATC air traffic control. 1, 3, 20, 26, 34, 66, 73, 78, 87, 88, 90, 96

CNN convolutional neural network. 16, 54, 64

CTC connectionist temporal classification. 15, 89

DNN deep neural network. vi, 5, 13, 26, 39, 40

DTW dynamic time warping. 7

GAN generative adversarial network. 25, 59, 60, 62–64, 91

GMM gaussian mixture model. 7

GPU graphics processing unit. 13, 32

GRU gated recurrent unit. 49

HiØ Østfold university college. 1, 32, 93

HMM hidden Markov model. 7, 12, 39

IDE integrated development environment. 33

KDA Kongsberg Defence & Aerospace. ii, 1, 3, 4, 87–90, 93–96

LPC linear predictive coding. 6, 12

LSTM long short-term memory. 49

MFCC mel-frequency cepstral coefficients. 8, 12, 46, 62, 92

NLP natural language processing. 46, 96

PCA principal component analysis. 24

RTS remote towers systems. ii, 1, 3, 4, 87, 88, 92, 96

STFT short-time Fourier transform. 10, 46, 61, 73, 76, 88, 89

VPN virtual private network. 33

WER word error rate. 19, 26, 64, 68, 75, 78, 82, 83, 85, 87–89, 92, 94

Appendix A

Conference paper

UNSUPERVISED LEARNING FOR AUTOMATIC SPEECH RECOGNITION IN AIR TRAFFIC CONTROL ENVIRONMENT

Lars Formoe[1], Dan Bruun Mygind[1], Espen Løkke[2], and Hasan Ogul[1]

[1] Department of Computer Science and Communication
Østfold University College, B R A Veien 4, 1757 Halden, Norway
{lars.formoe,dan.b.mygind,hasan.ogul}@hiof.no
[2] Kongsberg Defence & Aerospace, Drammen, Viken, Norway
espen.lokke@kongsberg.com

Abstract. This paper addresses the enduring challenge of domain-specific automatic speech recognition (ASR) with limited training data, particularly in air traffic control (ATC) communications involving highly accented speakers. While state-of-the-art models like wav2vec have achieved significant progress when fine-tuned on smaller, specialized datasets, the issue of having little or no transcribed data for specific domains like ATC remains unresolved. We present our findings using the wav2vec-U 2.0 model, an advanced self-supervised ASR framework that learns from raw audio and unpaired text without the need for transcriptions. By fine-tuning wav2vec-U 2.0 on the domain-specific ATCOSIM dataset, we explore its effectiveness in handling domain-specific ASR tasks with scarce or non-existent transcribed data. Our results demonstrate promising speech recognition accuracy, suggesting that wav2vec-U 2.0 can effectively address the problem with small amounts of transcribed data posed by highly specific ASR-domains such as ATC communications. Furthermore, we discuss the implications of our findings for the broader ASR research community and provide suggestions into potential future directions for improving ASR accuracy in ATC communications and other specialized domains with limited training data.

Keywords: Automatic speech recognition, wav2vec 2.0, wav2vec unsupervised, unsupervised learning, generative adversarial network, air traffic control communications

1 Introduction

Automatic Speech Recognition (ASR) has been a topic of interest in the air traffic control (ATC) domain for several decades, with the potential to enhance safety and efficiency by reducing controllers' workload and improving communication accuracy between pilots and controllers [30] [14]. However, the unique challenges associated with the ATC domain have made developing effective ASR systems particularly difficult.

ATC communications is a highly specialized domain that requires precise and accurate communication between pilots and controllers to ensure the safety and efficiency of air travel. Unlike other speech recognition applications, ATC communications often

involve technical terminology and specific communication protocols unique to the aviation industry [6]. Furthermore, ATC communications typically occur in high-noise environments, such as airports and control towers, which can significantly degrade speech signal quality and increase speech recognition tasks' complexity [9].

One key challenge in developing ASR systems for domain-specific applications is the limited availability of training data. Recently, an innovative ASR system, called Wav2vec-U 2.0, was released to address this challenge by learning from raw audio and unpaired text without the need for transcriptions [22]. It employs a self-supervised learning model and a generative adversarial network [13] to recognize words in audio recordings.

To the best of our knowledge, the promise of Wav2vec-U 2.0 in the ATC domain has not been explored yet. This paper presents our findings on implementing the wav2vec-U framework in the ATC domain using the publicly available dataset ATCOSIM [15]. Our experiments are geared towards assessing the efficacy of this ASR system in handling domain-specific challenges and evaluating its performance. By employing unlabeled data and the aforementioned framework, we have achieved word error rates below 30%.

2 Related Work

2.1 ASR in ATC Domain

The development of robust solutions for ATC involves addressing several challenges that are specific to this domain. For instance, noisy radio channels, a wide range of accents, and high speech rates are all individual challenges that coincide in the ATC domain. Recently, a study on methods to extract operational information from ATC audio yielded a Word Error Rate (WER) of 17% using a Recurrent Neural Network (RNN) model trained on a combination of ATC datasets that contained a total of 84 hours of transcribed speech data [7].

However, a key challenge in ATC is the limited available data. A recent study addressed this challenge by employing transfer learning techniques and leveraging multiple Chinese speech datasets. The study showed promising results, with an overall reduction of Character Error Rate (CER) of 2% when compared to the supervised learning approach [21].

2.2 Self-supervised representations for speech

Learning vector representations from high amount of labeled or unlabeled data is a recent trend to leverage the learned representations to improve performance on a local task for which a relatively small dataset is available. Wav2vec is an attempt to provide a learning representations of raw audio for speech recognition by unsupervised pre-training [26]. It is trained on large amounts of unlabeled audio data and the resulting representations are then used to improve acoustic model training. The model is a convolutional neural network that takes raw audio as input and computes a general representation that can be input to a speech recognition system.

Wav2vec 2.0 presents an improved version of the original wav2vec model. Wav2vec 2.0 employs a self-supervised learning framework that utilizes a contrastive loss to learn

speech representations from raw audio without labeled data [8]. In addition, the model learns to predict future audio samples by employing a transformer-based architecture and a masked language modeling objective. Experiments demonstrate that wav2vec 2.0 outperforms previous approaches, achieving state-of-the-art results on the LibriSpeech and Switchboard-300 datasets, even with limited labeled data for fine-tuning.

Wav2vec-U 2.0 is an unsupervised model designed to learn and understand audio representations, especially when labeled data is scarce [22]. It uses raw audio and unrelated text data, removing the need for transcriptions. The model combines the self-supervised wav2vec 2.0, pre-trained on raw audio, with a generative adversarial network (GAN) [13] to improve audio representations. Using contrastive learning, it aligns speech representations with text embeddings, allowing effective word recognition without labeled data.

2.3 Use of learned representations in ATC

wav2vec 2.0 fine-tuning A recent study conducted experiments on pre-trained Wav2Vec 2.0 models' robustness in downstream ASR tasks, explicitly targeting the air traffic control domain [31]. Using domain-specific datasets, a range of corpora has been utilized to ascertain the effectiveness of fine-tuning these pre-trained models, which have been exposed to substantial amounts of generic English audio. The results highlight the potential of implementing pre-trained Wav2Vec 2.0 models for ASR tasks within specialized domains.

While this study does not present explicit results concerning ASR performance on the ATCOSIM dataset [15], their experimental framework resembles the approach we have undertaken in our work. Using pre-trained Wav2Vec 2.0 models for ASR tasks in the air traffic control domain underscores the value of transfer learning and domain-specific fine-tuning. Furthermore, the study suggests that the versatility and adaptability of Wav2Vec 2.0 models make them ideal candidates for a wide range of specialized ASR tasks, including those in the air traffic control domain. Furthermore, the end-to-end nature of these models, as evidenced by their superior performance compared to hybrid-based ASR systems, emphasizes the importance of investigating and developing more streamlined and efficient ASR architectures.

wav2vec-U 2.0 A study investigated ASR for use with Uyghur, Kazakh, and Kyrgyz audio, employing unsupervised learning by implementing the wav2vec-U 2.0 framework [11]. With only 1.8 hours of Kyrgyz audio data from the Common Voice dataset [4], their approach achieved a 14.9% Character Error Rate (CER), demonstrating the potential of unsupervised learning for ASR in languages with scarce resources.

Another study into the robustness of unsupervised speech recognition investigated the utilization of unrelated text data within the same language across common datasets used in ASR research [20]. The findings of this research suggest a correlation between the nature of the text supplied for the GAN and the overall performance. It was observed that out-of-domain text data generally yielded inferior results, reinforcing the necessity of ATC-specific text data to achieve optimal performance.

3 Methods

The primary objective of ASR research is to generate the most accurate textual representation of audio possible. Here, we conducted an experimental study to assess the performance of recent end-to-end speech recognition technology in the ATC domain. To this end, we employed a rigorous preprocessing procedure on the text data to ensure consistency and a uniform input representation. All corpus transcriptions were filtered, transforming all letters to lowercase and removing special characters not part of the alphabet. The process reduces the number of tokens used in both models by 49%, impacting training cost and the output probability distribution in the output layer. However, the preprocessing of the models differentiates from this point. While fine-tuning the wav2vec 2.0 model can be initiated with the raw audio files and the paired transcripts. The wav2vec 2.0 Unsupervised preprocessing is more circumstantial. Although various representations of sentences, words, and phones were mapped accordingly, given the domain-specific nature of air traffic control communications, we elected to use the actual text data from the same dataset to maintain relevance and accuracy. This decision was based on the understanding that air traffic control communication data exhibits unique characteristics that may not be present in generic text data, thus requiring specialized treatment. Furthermore, the audio was files trimmed to remove silence [27] and a Mel-frequency cepstral coefficients (MFCCs) [16] representations of the trimmed audio was created for training purposes.

Subsequently, the ATCOSIM dataset of 9538 samples was partitioned into training, validation, and testing subsets at 70%, 20%, and 10% distributions, respectively. The distribution resulted in a split key of 6677/1907/954. The samples were selected randomly to ensure a fair representation of the data. This partitioning strategy allowed us to establish a robust evaluation framework, minimizing the risk of overfitting and ensuring that the model's performance could be assessed on unseen data.

wav2vec 2.0: The experiment's first phase was conducted with the wav2vec 2.0 framework and weights from four pre-trained models to evaluate the downstream infusion of the ATCOSIM dataset. Wav2vec 2.0 results are based on the following weight pipelines. Wav2vec2-base (95m param.) pre-trained on 53000 hours with no fine-tuning, wav2vec2-large-960h-lv60-self (317m param.) pre-trained on 53.000 hours of unlabeled data from the LibriVox subset Librilight [19] and fine-tuned on 960 hours of LibriSpeech [23]. Wav2vec2-xls-r-300m (300m param.) pre-trained on 436.000 hours of unlabeled multilingual data from multiple corpora CommonVoice [5], VoxPopuli [29], Multilingual LibriSpeech (MLS) [24], VoxLingua107 [28] and the Babel project. Wav2vec2-large-robust-ft-swbd-300h pre-trained on Libri-light, CommonVoice, Switchboard [12] and Fisher [10], then fine-tuned on 300 hours of noisy telephone data from the Switchboard corpus. All pre-trained pipelines provide a diverse and comprehensive source of training data for ASR systems, adding a robust context baseline for downstream tasks.

We implemented the pre-trained pipelines using the Huggingface platform [2], adapting the code for custom fine-tuning of the ATCOSIM corpus. Furthermore, we added a hyper-parameter loop to examine different training arguments' impacts on the results. The parameters used were warmup steps, epochs, batch size, and learning rate. All results are based on the 954 unseen samples from the ATCOSIM corpus.

wav2vec 2.0 Unsupervised: The experiment's second phase was conducted using the Wav2vec Unsupervised 2.0. To implement the wav2vec Unsupervised 2.0 framework, we followed the guidelines in the Fairseq GitHub repo [1]. In addition, we created an additional layer on the framework to execute the many steps involved in the process seamlessly and efficiently. For preparing the input to the Generative Adversarial Network (GAN), we used the pre-trained wav2vec2-large model with no fine-tuning of the model, extracting the contextual representations in latent space from the raw audio files. The wav2vec2-large model is equivalent to the Wav2vec2-base described in the above section but has a feature dimension of 1024 instead of 768. Clustering it with the recommended 64 centroids for the K-means training. For the language identification model, we used fastText lid176.bin model [17, 18] instead of the proposed model.

We then conducted several training runs with minor configuration adjustments to the training parameters, including learning rate, batch size, and the number of training epochs. This iterative approach allowed us to refine the model based on the results of each training run and facilitated the identification of optimal parameter settings for our specific task. The results have also been generated utilizing 954 unseen samples derived from the ATCOSIM corpus.

OpenAI Whisper: In addition to the first and second phases and for comparison purposes, we used the Whisper models [25] from OpenAi's GitHub repository [3] to conduct zero-shot predictions on our ATCOSIM test-set audio files. Using the small (244M. param.), medium (769M param.), and large (1550M. param.) architectures. Each model was pre-trained on 680.000 hours of labeled multilingual data. The integrated pre-processing within the framework allows for the simple input of an audio file as the sole requirement to receive a prediction. Subsequently, we developed a module for post-prediction analysis to evaluate the WER against outcomes from earlier phases while excluding upper- and lower-case letters and special characters from the output.

Table 1: Pre-trained models data foundation. Models containing unlabeled and labeled data are pre-trained and fine-tuned before the experiments.

Model	Unlabeled data (hours)	Labeled data (Hours)
wav2vec2-base	53.000	-
wav2vec2-xls-r-300m	436.000	-
wav2vec2-large-960h-lv60-self	53.000	960
wav2vec2-large-robust-ft-swbd-300h	NA	300
openai whisper (small, medium, large)	-	680.000

By comparing the performance of the unsupervised wav2vec 2.0 framework with that of self-supervised and supervised approaches using pre-trained models, we aimed to gain insight into the relative effectiveness of these methodologies in the context of air traffic control communications.

4 Results

4.1 wav2vec 2.0 downstream

The wav2vec2-large-960h-lv60-self pipeline described in section 3 had the best overall performance of the pre-trained models. Figure 1 show the loss across the 50-epoch training sequence. The model was used to predict text outputs from the unseen test subset. Using the jiwer package for Python[3], each prediction was given an individual *WER* score, yielding an average *WER* of 0.97%.

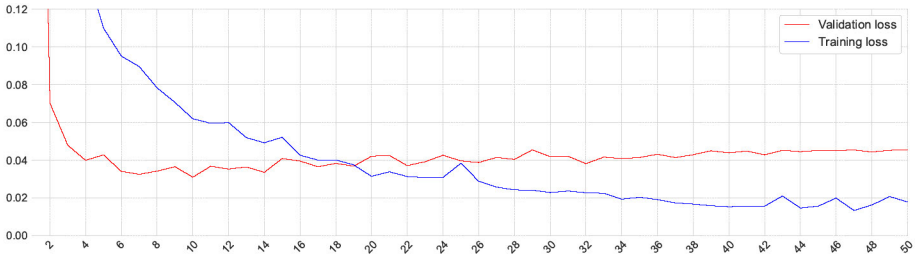


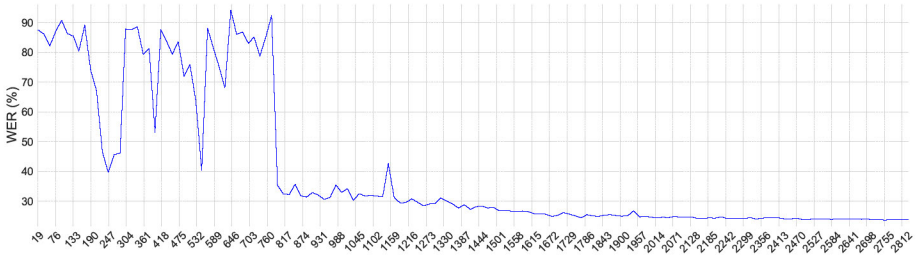
Fig. 1: Results from large-960h-lvl60-self model training.

4.2 wav2vec Unsupervised

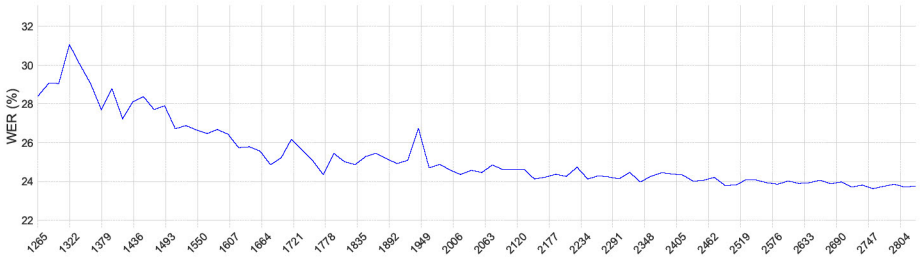
In terms of accuracy metrics, there appears to be some ambiguity within the framework itself. During training, there is a validation run performed every n-th epoch where one of the metrics is *valid_uer*. When running the script for generating phone labels - w2vu_generate.py - the metric used is *WER*. As both valid_uer and WER appear to be the same, for consistency, we have elected only to use the term WER.

The two different perspectives in figure 2a and 2b shows the progress of the *WER* throughout the training phase. The lowest *WER* achieved on the validation subset was 23.79% and 25.37% on the unseen test subset.

[3] `https://github.com/jitsi/jiwer`



(a) WER progression for an entire training run.



(b) Details from the latter part.

Fig. 2: wav2vec 2.0 Unsupervised on ATCOSIM dataset.

4.3 Whisper

Table 2 presents the performance metrics of the Whisper architectures on the test subset. Original WER results are based on the output with full token representation, and the corrected output is post-processed to match the token set used in 4.1 and 4.2. The best *WER* achieved by Whisper was 29.54%.

Table 2: Whisper model performance.

model	original WER	corrected WER
small	91.13%	46.95%
medium	84.22%	31.32%
large	82.94%	29.54%

5 Conclusion

While ASR technology has the potential to significantly benefit the ATC domain, its development and implementation face unique challenges. Recent advancements, such

as transfer learning and the wav2vec-U framework, offer promising solutions to address these challenges, paving the way for more accurate and reliable ASR systems in the ATC domain.

The findings of this study underscore the potential of using unsupervised learning and fine-tuning pre-trained Wav2Vec 2.0 models to advance research in creating robust ASR systems for the specialized domain of air traffic control communications. Furthermore, the results highlight the effectiveness of fine-tuning these models on smaller, domain-specific datasets to yield improved ASR performance.

Future research endeavors should refine the fine-tuning process by applying the experimental setup to larger corpora of audio data specific to the air traffic control domain. This process would involve using small transcribed datasets for fine-tuning across multiple airports within the same region. Exploring this approach is crucial, as it may reduce dependency on large volumes of transcribed data and facilitate ASR system development in low-resource languages and specialized domains.

In summary, this study has provided some insight into the potential of fine-tuning pre-trained Wav2Vec 2.0 models using unsupervised learning techniques for advancing research in robust ASR systems tailored to the air traffic control domain. Refining the fine-tuning process and leveraging unsupervised learning benefits can contribute to the ongoing efforts to develop more accurate and versatile ASR systems.

References

1. Fairseq facebookresearch. `https://github.com/facebookresearch/fairseq`, accessed: 2022-01-30
2. Huggingface platform. `https://huggingface.co`, accessed: 2021-11-30
3. Whisper openai. `https://github.com/openai/whisper`, accessed: 2022-02-10
4. Ardila, R., Branson, M., Davis, K., Henretty, M., Kohler, M., Meyer, J., Morais, R., Saunders, L., Tyers, F.M., Weber, G.: Common voice: A massively-multilingual speech corpus. arXiv preprint arXiv:1912.06670 (2019)
5. Ardila, R., Branson, M., Davis, K., Henretty, M., Kohler, M., Meyer, J., Morais, R., Saunders, L., Tyers, F.M., Weber, G.: Common voice: A massively-multilingual speech corpus (2020)
6. Authority, C.A.: Cap 413: Radiotelephony manual. Civil Aviation Authority (2020)
7. Badrinath, S., Balakrishnan, H.: Automatic speech recognition for air traffic control communications. Transportation research record **2676**(1), 798–810 (2022)
8. Baevski, A., Zhou, Y., Mohamed, A., Auli, M.: wav2vec 2.0: A framework for self-supervised learning of speech representations. Advances in neural information processing systems **33**, 12449–12460 (2020)
9. Benzeghiba, M., De Mori, R., Deroo, O., Dupont, S., Erbes, T., Jouvet, D., Fissore, L., Laface, P., Mertins, A., Ris, C., et al.: Automatic speech recognition and speech variability: A review. Speech communication **49**(10-11), 763–786 (2007)
10. Cieri, C., Miller, D., Walker, K.: The fisher corpus: a resource for the next generations of speech-to-text. In: International Conference on Language Resources and Evaluation (2004)
11. Du, W., Maimaitiyiming, Y., Nijat, M., Li, L., Hamdulla, A., Wang, D.: Automatic speech recognition for uyghur, kazakh, and kyrgyz: An overview. Applied Sciences **13**(1), 326 (2022)
12. Godfrey, J., Holliman, E., McDaniel, J.: Switchboard: telephone speech corpus for research and development. In: [Proceedings] ICASSP-92: 1992 IEEE International Conference on Acoustics, Speech, and Signal Processing. vol. 1, pp. 517–520 vol.1 (1992). https://doi.org/10.1109/ICASSP.1992.225858

13. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial networks. Communications of the ACM **63**(11), 139–144 (2020)
14. Helmke, H., Ohneiser, O., Mühlhausen, T., Wies, M.: Reducing controller workload with automatic speech recognition. In: 2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC). pp. 1–10. IEEE (2016)
15. Hofbauer, K., Petrik, S., Hering, H.: The atcosim corpus of non-prompted clean air traffic control speech. In: LREC. Citeseer (2008)
16. Hossan, M.A., Memon, S., Gregory, M.A.: A novel approach for mfcc feature extraction. In: 2010 4th International Conference on Signal Processing and Communication Systems. pp. 1–5 (2010). https://doi.org/10.1109/ICSPCS.2010.5709752
17. Joulin, A., Grave, E., Bojanowski, P., Douze, M., Jégou, H., Mikolov, T.: Fasttext.zip: Compressing text classification models. arXiv preprint arXiv:1612.03651 (2016)
18. Joulin, A., Grave, E., Bojanowski, P., Mikolov, T.: Bag of tricks for efficient text classification. arXiv preprint arXiv:1607.01759 (2016)
19. Kahn, J., Riviere, M., Zheng, W., Kharitonov, E., Xu, Q., Mazare, P., Karadayi, J., Liptchinsky, V., Collobert, R., Fuegen, C., Likhomanenko, T., Synnaeve, G., Joulin, A., Mohamed, A., Dupoux, E.: Libri-light: A benchmark for ASR with limited or no supervision. In: ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE (may 2020). https://doi.org/10.1109/icassp40776.2020.9052942, https://doi.org/10.1109%2Ficassp40776.2020.9052942
20. Lin, G.T., Hsu, C.J., Liu, D.R., Lee, H.Y., Tsao, Y.: Analyzing the robustness of unsupervised speech recognition. In: ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). pp. 8202–8206. IEEE (2022)
21. Lin, Y., Li, Q., Yang, B., Yan, Z., Tan, H., Chen, Z.: Improving speech recognition models with small samples for air traffic control systems. Neurocomputing **445**, 287–297 (2021)
22. Liu, A.H., Hsu, W.N., Auli, M., Baevski, A.: Towards end-to-end unsupervised speech recognition. In: 2022 IEEE Spoken Language Technology Workshop (SLT). pp. 221–228. IEEE (2023)
23. Panayotov, V., Chen, G., Povey, D., Khudanpur, S.: Librispeech: an asr corpus based on public domain audio books. In: 2015 IEEE international conference on acoustics, speech and signal processing (ICASSP). pp. 5206–5210. IEEE (2015)
24. Pratap, V., Xu, Q., Sriram, A., Synnaeve, G., Collobert, R.: MLS: A large-scale multilingual dataset for speech research. In: Interspeech 2020. ISCA (oct 2020). https://doi.org/10.21437/interspeech.2020-2826, https://doi.org/10.21437%2Finterspeech.2020-2826
25. Radford, A., Kim, J.W., Xu, T., Brockman, G., McLeavey, C., Sutskever, I.: Robust speech recognition via large-scale weak supervision (2022)
26. Schneider, S., Baevski, A., Collobert, R., Auli, M.: wav2vec: Unsupervised pre-training for speech recognition. arXiv preprint arXiv:1904.05862 (2019)
27. Tan, Z.H., kr. Sarkar, A., Dehak, N.: rvad: An unsupervised segment-based robust voice activity detection method (2022)
28. Valk, J., Alumäe, T.: Voxlingua107: a dataset for spoken language recognition (2020)
29. Wang, C., Rivière, M., Lee, A., Wu, A., Talnikar, C., Haziza, D., Williamson, M., Pino, J., Dupoux, E.: Voxpopuli: A large-scale multilingual speech corpus for representation learning, semi-supervised learning and interpretation (2021)
30. Yi, L., Min, R., Kunjie, C., Dan, L., Ziqiang, Z., Fan, L., Bo, Y.: Identifying and managing risks of ai-driven operations: A case study of automatic speech recognition for improving air traffic safety. Chinese Journal of Aeronautics (2022)

31. Zuluaga-Gomez, J., Prasad, A., Nigmatulina, I., Sarfjoo, S.S., Motlicek, P., Kleinert, M., Helmke, H., Ohneiser, O., Zhan, Q.: How does pre-trained wav2vec 2.0 perform on domain-shifted asr? an extensive benchmark on air traffic control communications. In: 2022 IEEE Spoken Language Technology Workshop (SLT). pp. 205–212. IEEE (2023)

Appendix B

Python code

B.1 grid_search.py

```
def grid_search():
    dataset = ['atcosim', 'hiwire']
    subset = ['clean', 'ln', 'hn', 'all']
    input_types = ['mfcc', 'stft']
    batch_sizes = [32, 64, 128]
    dropouts = [0.1, 0.2, 0.3]
    frame_sizes = [512, 1024, 2048]
    frame_steps = [256, 512, 1024]
    fft_bins = [512, 1024, 2048]
    mel_bands = [20, 40, 80, 100]
    heads = [1, 2, 4, 8, 16, 32]
    units = [100, 200, 400, 600]
    neurons = [100, 200, 300]
    num_layers_enc = [2, 4, 8, 16, 32]
    num_layers_dec = [2, 4, 8, 16, 32]


    for d, s, i, b, dp, fs, fss, fft, mel, h, u, nff, nle, nld in
    ↪  it.product(dataset, subset, input_types, batch_sizes, dropouts,
    ↪  frame_sizes, frame_steps, fft_bins, mel_bands, heads, units, neurons,
    ↪  num_layers_enc, num_layers_dec):
        if d == 'atcosim' and s in ['ln', 'hn']:
            continue
        elif d == 'hiwire' and s == 'all':
            continue
        else:
            print(f'Model:\nDataset - {d} | Subset - {s}\n'
                  f'Input - {i} | Batchsize - {b}\n'
                  f'Dropout - {dp} | Frame_sizes - {fs}\n'
                  f'Frame_steps - {fss} | Fft_bins - {fft}\n'
                  f'mel_bands - {mel} | heads - {h}\n'
                  f'units - {u} | neurons - {nff}\n'
                  f'num_layers_enc - {nle} | num_layers_dec - {nld}')
```

```
            threshold = transformer_atc(d, s, i, b, dp, fs, fss, fft, mel, h,
            ↪  u, nff, nle, nld)
            if threshold < 5.8:
                print("WER threshold reached")
                break
            else: continue
```

B.2 unsup_README.md

```
python main.py for full execution
python preprocess.py ^^Imodule 1 execution
python inputgan.py ^^Imodule 2 execution
python rungan.py ^^Imodule 3 execution
python generate.py^^Ifinal prediction on unseen data WER - standalone run after
↪  training, remember to update variables.)
Hardware: gpu cluster of two Nvidia A100 80gb GPUs

Prerequesits: ^^IFull install of the fairseq framework
↪  (https://github.com/facebookresearch/fairseq) v0.12.2 as of 05.15.2023
^^I^^INB! remember to export environmental variables and follow the
↪  installation documentation provided by fairseq^^I^^I

Default file structure: - place audio and dataset folder in run dir
-run
 --audio
    ---.wav files
 --dataset
    ---.csv train/valid
 --exe
    ---generated with sub structure when main.py or preprocess.py is executed.
     ----download: weights and language binaries
     ----ltr_ohn_wrd: prep files
     ----manifest: tsv files path and byte size
     ----final: trimmed audio files, prep files, trimmed tsv.
     ----prepped_data:model/final, preppedaudio, preppedtext
 --tmp
    ---generated when main.py or preprocess.py is runned.
 --output
    ---generated by gan_run.py saves log
 --multirun
    ---generated by gan_run.py saves model checkpoints
 --predictions
    ---output dir for model predictions

nohup activation. Check export CUDA_VISIBLE_DEVICES=0 in run_gan() matches your
↪  GPU setup. pip install gpustat or use nvidia-smi in commandline to see if
progress running correct on gpu. cat logfile for training details.
```

B.3 unsup_preprocess.py

```
import os
import re
import sys
import glob
import uuid
import shutil
import threading
import subprocess
import pandas as pd

"""
    Authors:        Lars Formoe and Dan Bruun Mygind
    Note:           The preprocess.py script is minted towards automating the
↪   preprocessing of the audio and text.
    Prerequesits:   - Full install of the fairseq framework
↪   (https://github.com/facebookresearch/fairseq) v0.12.2 as of 05.15.2023
                    - train/valid/test.csv with path to audio files and
↪   transcript.
                    - folder with raw audio, folder with dataset.
                    - place preprocess.py next to $FAIRSEQ_ROOT and run.
"""
#id
ids = uuid.uuid1()
ids = ids.hex[:6]

path_csv = "dataset" #input("Enter path to dataset/root/dir: ")
path_audio = "audio" #input("Enter path to audio/root/dir: ")

if os.path.isdir(path_csv) is False or os.path.isdir(path_audio) is False:
    print("Invalid path to dataset.")
    sys.exit("Error: invalid path.")
else:
    print("Input paths acepted.")

# audio files
audio_files = glob.glob(f"{path_audio}/*.wav")

def mkdir(dir):
    """
    check if dir exist else create new.
    :param dir: name of directories.
    """
    if os.path.isdir(dir):
        print(f"- {dir} dir exsist, no action needed.")
    else:
        os.mkdir(dir)
        print(f"- {dir} directory created.")

def generate_wrd(datafile):
    """
```

```
    creating .wrd files by mapping audio/.wav order with text from .csv.
    :param datafile: the dataset .csv files.
    :return: print status
    """
    # dataframe
    df = pd.read_csv(datafile)
    #datafile name
    reg = re.search(r"(?<=/)[^_.]+", datafile)
    # create .wrd
    open(f"exe/ltr_phn_wrd/{reg.group(0)}.wrd", 'w+')

    # map audio file and text from .csv
    for wav in audio_files:
        wav_re = wav.replace("audio/","")
        for index, row in df.iterrows():
            if wav_re in row["file"]:
                try:
                    with open(f"exe/ltr_phn_wrd/{reg.group(0)}.wrd", 'a') as f:
                        #writer = csv.writer(f)
                        f.write(row["text"]+"\n")
                        #writer.(row["text"])
                        f.close()
                    shutil.copy(f"{wav}", f"{path_audio}/{reg.group(0)}")
                except:
                    print(f"{wav} not in {datafile}")
    return print(f"- {reg.group(0)}.wrd created")

def generate_ltr(datafile):
    """
    read .wrd files and replace space with special char.
    :param: datafile:   filename train/valid/test
    :return: print status
    """
    with open(f'exe/ltr_phn_wrd/{datafile}.wrd', 'r') as inf,
    ↪  open(f'exe/ltr_phn_wrd/{datafile}.ltr', 'w') as out:
        for line in inf.readlines():
            print(" ".join(list(line.strip().replace(" ", "|"))) + " |",
            ↪  file=out)
    return print(f"- {datafile}.ltr created")

def generate_phn(datafile):
    """
    phominize the .wrd file en-us and saves it in .phn. The train.phn is split
↪   in words and valid and test in letters.
    :param datafile:    filename .word to .phn
    :return: print status
    """
    if datafile == "train":
        phn_gen = f'''
                echo "----- {datafile}.phn phoneme file -----"
```

```
                cat "exe/ltr_phn_wrd/{datafile}.wrd" |
↪   PHONEMIZER_ESPEAK_PATH=$(which espeak) phonemize -o
↪   "exe/ltr_phn_wrd/{datafile}.phn" -p '' -w ' ' -l en-us -j 70
↪   --language-switch remove-flags
                '''
        bash_generate(phn_gen, ".phn")
    else:
        phn_gen = f'''
            ^^Iecho "----- {datafile}.phn phoneme file -----"
            ^^Icat "exe/ltr_phn_wrd/{datafile}.wrd" |
↪   PHONEMIZER_ESPEAK_PATH=$(which espeak) phonemize -o
↪   "exe/ltr_phn_wrd/{datafile}.phn" -p ' ' -w '' -l en-us -j 70
↪   --language-switch remove-flags
            '''
        bash_generate(phn_gen, ".phn")


def generate_dict(datafile):
    """
    creating dictionary from .wrd file. key:value word"\t"phoneme:
    :param datafile: filename train/valid/test
    :return: print status
    """
    dict_gen = f'''
            echo "----- dict.{datafile} dictionary file -----"
            cat exe/ltr_phn_wrd/{datafile}.wrd|tr ' ' '\n'|sort|uniq >
↪   tmp/{datafile}.wl; cat tmp/{datafile}.wl | PHONEMIZER_ESPEAK_PATH=$(which
↪   espeak) phonemize -o tmp/{datafile}.wl.phn -p '' -w ' ' -l en-us  -j 70
↪   --language-switch remove-flags;paste tmp/{datafile}.wl
↪   tmp/{datafile}.wl.phn > exe/ltr_phn_wrd/dict.{datafile}
            '''
    bash_generate(dict_gen, "dict.")

def generate_manifest(src, datafile, path):
    """
    creating manifest files .tsv. filepath"\n"bitsize
    :param datafile: filename train/valid/test
    :return: print status
    """
    mani_gen = f'''
            echo "----- {datafile}.tsv manifest file -----"
            python $FAIRSEQ_ROOT/examples/wav2vec/wav2vec_manifest.py
↪   {src}{datafile} --ext wav --dest {path}{datafile} --valid-percent 0
            '''
    bash_generate(mani_gen, ".tsv")

def generate_vads(datafile):
    """
    generating rVAD files, identifying scilence.
    :param datafile: filename train/valid/test
    :return: print status
```

```
    """
    vads_gen = f'''
            echo "----- {datafile}.vads rVAD file -----"
            python $FAIRSEQ_ROOT/examples/wav2vec/unsupervised/scripts/vads.py
↪   -r $RVAD_ROOT < exe/manifest/{datafile}/{datafile}.tsv >
↪   exe/final/{datafile}.vads
            '''
    bash_generate(vads_gen, "VAD")

def remove_silence(datafile):
    remove = f'''
        echo "----- Removing {datafile} silence -----"
        python
↪   $FAIRSEQ_ROOT/examples/wav2vec/unsupervised/scripts/remove_silence.py --tsv
↪   exe/manifest/{datafile}/{datafile}.tsv --vads exe/final/{datafile}.vads
↪   --out exe/final/
        '''
    bash_generate(remove, "remove silence")

def generate_sentence():
    """
    generates a sortet file with sentences.
    """
    sen = '''cat exe/ltr_phn_wrd/*.wrd | grep -v '^$' | sort| uniq >
    ↪   exe/sentences.txt'''
    bash_generate(sen, "sentence")


def bash_generate(name_gen, type):
    """
    bash script engine.
    :param name_gen: what generator
    :param type: type generated
    :return: status
    """
    process = subprocess.Popen(['bash', '-c', name_gen],
    ↪   stdout=subprocess.PIPE, stderr=subprocess.PIPE)

    output_thread = threading.Thread(target=print_output, args=(process,))
    output_thread.start()
    process.wait()
    output_thread.join()

def print_output(process):
    for line in process.stdout:
        print(line.decode().strip())

if __name__ == '__main__':
    #----- Preprocess.py Init -----)

    #default directories.
```

```
dir = ["exe", "exe/ltr_phn_wrd", "exe/manifest", "exe/prepped_data",
↪  "exe/final", "exe/download", "tmp", "predictions"]

#add dynamic directories to dir, acording to dataset naming and split.
for file in glob.glob(f"{path_csv}/*.csv"):
    reg = re.search(r"(?<=/)[^_.]+", file)
    dir.append(f"{path_audio}/{reg.group(0)}")
    dir.append(f"exe/manifest/{reg.group(0)}")

# create default dir
[mkdir(name) for name in dir]

# initiate wrd, ltr, phn, dict, tsv, vads creation.
for file in glob.glob(f"{path_csv}/*.csv"):
    reg = re.search(r"(?<=/)[^_.]+", file)
    reg = reg.group(0)
    print(f"----- Generating {reg} files -----")
    generate_wrd(str(file))
    generate_ltr(reg)
    generate_phn(reg)
    generate_dict(reg)
    generate_manifest("audio/", reg, "exe/manifest/")
    #rename manifest files.
    try:
        shutil.move(f"exe/manifest/{reg}/train.tsv",
        ↪  f"exe/manifest/{reg}/{reg}.tsv")
    except:
        pass
    generate_vads(reg)
    remove_silence(reg)
    generate_manifest("exe/final/" ,reg, "exe/final/")
    try:
        shutil.move(f"exe/final/{reg}/train.tsv", f"exe/final/{reg}.tsv")
    except:
        pass
generate_sentence()

#move files to final dir.
for file in glob.glob("exe/ltr_phn_wrd/*"):
    shutil.move(file, "exe/final/")
#move final dir
shutil.move("exe/final",f"exe/prepped_data/{ids}/final")
#change .tsv new path
for tsv in glob.glob(f"exe/prepped_data/{ids}/final/*tsv"):
    print(tsv)
    reg = re.search(r"([^/]+)(?=\.)", tsv)
    reg = reg.group(0)
    df = pd.read_csv(str(tsv))
    df.rename(columns =
    ↪  {str(df.columns[0]):f"exe/prepped_data/{ids}/final/{reg}"}, inplace
    ↪  = True)
```

```
        print(df)
        df.to_csv(f"exe/prepped_data/{ids}/final/{reg}.tsv", index=False)

    [shutil.rmtree("audio/"+str(x)) if os.path.isdir("audio/"+str(x))==True
    ↪  else None for x in os.listdir("audio/")]
    print("-"*100)
    print(f"final directory moved to exe/prepped_data/{ids}\nAlgorithm
    ↪  successfully executed.")
```

B.4 unsup_inputgan.py

```
import os
import re
import sys
import glob
import uuid
import shutil
import subprocess
import pandas as pd
import preprocess as pre

"""
    Authors:        Lars Formoe and Dan Bruun Mygind
    Note:           The gan_input.py script is minted towards automating the
↪   process of generating the gan input.
    Prerequesits:   - Full install of the fairseq framework
↪   (https://github.com/facebookresearch/fairseq) v0.12.2 as of 05.15.2023
                    - run preprocess.py
                    - For full process run main.py
"""
def download(link):
    """
    downloading language identification binaries and model weights.
    """
    reg = re.search(r"([^/]+$)", link)

    if not os.path.isfile(f"exe/download/{reg.group(0)}"):
        download = f'''
                echo "----- language identification binaries and model weights
↪   -----"
                echo "wget {link} downloading."
                wget -P exe/download {link}
                '''
        pre.bash_generate(download, "download")
    else:
        print(f"Download: {link} allready exsist.")

def prepare_audio(file):
    """
    preparing audio input for gan. Creating MFCC and vector clustering.
```

```
    """
    audio_prep = f'''
                echo "----- audio input prepare  -----"
                export CUDA_VISIBLE_DEVICES=0
                export OPENBLAS_NUM_THREADS=1
                export GOTO_NUM_THREADS=1
                export OMP_NUM_THREADS=1
                echo "check hyperparams in
↪   $FAIRSEQ_ROOT/examples/hubert/simple_kmeans/learn_kmeans.py"
                zsh
↪   $FAIRSEQ_ROOT/examples/wav2vec/unsupervised/scripts/prepare_audio_v2.sh
↪   {file}/final {file}/preppedaudio exe/download/libri960_big.pt 64 14
                '''
    pre.bash_generate(audio_prep, "prep audio")

def prepare_text(file, sentences):
    """
    preparing text input for gan.
    """
    text_prep = f'''
                echo "----- text input prepare  -----"
                zsh
↪   $FAIRSEQ_ROOT/examples/wav2vec/unsupervised/scripts/prepare_text.sh en
↪   {sentences} {file}/preppedtext 5 espeak exe/download/lid.176.bin 0.5
                '''
    pre.bash_generate(text_prep, "prep text")


if __name__ == '__main__':
    download_list =
    ↪   ["https://dl.fbaipublicfiles.com/fairseq/wav2vec/libri960_big.pt",
    ↪   "https://dl.fbaipublicfiles.com/fasttext/supervised-models/lid.176.bin"]
    [download(link) for link in download_list]

    list_of_final = glob.glob('exe/prepped_data/*')
    latest_file = os.path.abspath(max(list_of_final, key=os.path.getctime))
    sentence_path = os.path.abspath("exe/sentences.txt")
    print(latest_file)
    print(sentence_path)

    prepare_audio(latest_file)
    prepare_text(latest_file, sentence_path)
    #move dict.phn.txt
    shutil.copy(f"{latest_file}/preppedtext/phones/dict.phn.txt",
    ↪   f"{latest_file}/preppedaudio")
    print(f"mv: {latest_file}/preppedtext/phones/dict.phn.txt to
    ↪   {latest_file}/preppedaudio")
```

B.5 unsup_rungan.py

```
import os
import re
import sys
import glob
import uuid
import shutil
import subprocess
import pandas as pd
import preprocess as pre

"""
    Authors:        Lars Formoe and Dan Bruun Mygind
    Note:           The gan_run.py script is minted towards automating the
↪   process of executing the gan training.
    Prerequesits:   - Full install of the fairseq framework
↪   (https://github.com/facebookresearch/fairseq) v0.12.2 as of 05.15.2023
                    - run preprocess.py
                    - run gan_input.py
                    - For full process run main.py
"""

def rungans(path):
    rungan = f'''
        export CUDA_VISIBLE_DEVICES=0
        PYTHONPATH=$FAIRSEQ_ROOT PREFIX=w2v_unsup_gan_xp fairseq-hydra-train \
            -m --config-dir
↪   $FAIRSEQ_ROOT/examples/wav2vec/unsupervised/config/gan \
            --config-name w2vu2.yaml \
            task.data={path}/preppedaudio \
            task.text_data={path}/preppedtext/phones \
            task.kenlm_path={path}/preppedtext/phones/lm.phones.filtered.04.bin
↪   \
            common.user_dir=$FAIRSEQ_ROOT/examples/wav2vec/unsupervised \
            model.code_penalty=2,4 model.gradient_penalty=1.5,2.0 \
            model.smoothness_weight=0.5,0.75,1.0 'common.seed=range(0,5)' \
        '''
    pre.bash_generate(rungan,"test")


if __name__ == '__main__':
    list_of_final = glob.glob('exe/prepped_data/*')
    latest_file = os.path.abspath(max(list_of_final, key=os.path.getctime))
    rungans(latest_file)
```

B.6 unsup_generate.py

```
import os
import re
```

```
import sys
import glob
import uuid
import shutil
import subprocess
import pandas as pd
import preprocess as pre

"""
    Authors:        Lars Formoe and Dan Bruun Mygind
    Note:           The generate.py script is minted towards generating
↪   predictions with gan trained weights .pt.
    Prerequesits:   - Full install of the fairseq framework
↪   (https://github.com/facebookresearch/fairseq) v0.12.2 as of 05.15.2023
                    - run preprocess.py
                    - run gan_input.py
                    - run rungan.py
                    - For full process run main.py
"""

def generate_pred(model,checkpoint,data,out):
    gen = f'''
        python $FAIRSEQ_ROOT/examples/wav2vec/unsupervised/w2vu_generate.py
↪   --config-dir $FAIRSEQ_ROOT/examples/wav2vec/unsupervised/config/generate
↪   --config-name viterbi
↪   fairseq.common.user_dir=$FAIRSEQ_ROOT/examples/wav2vec/unsupervised
↪   fairseq.task.data={model}/preppedaudio
↪   fairseq.common_eval.path={checkpoint} fairseq.dataset.gen_subset={data}
↪   results_path={out}/{model}

        '''
    pre.bash_generate(gen,"generate")


if __name__ == '__main__':
    #config
    model = "xyz"
    checkpoint = "multirun/2023-04-11/00-52-15/0/checkpoint_best.pt"
    data = "valid"
    #absolut path needed to run generator.
    model_abs = os.path.abspath(f"exe/prepped_data/{model}")
    check_abs = os.path.abspath(f"checkpoint")
    out_dir = os.path.abspath("predictions")
```

B.7 IAI_whisper.py

```
import os
import whisper
import numpy as np
import pandas as pd
from csv import DictWriter
from datasets import load_dataset, load_metric

os.environ["CUDA_VISIBLE_DEVICES"] = "0"

df = pd.read_csv("/media/datastore/x/whisper/data/validation.csv")
print(len(df.index))
for index, row in df.iterrows():
    if len(row['text']) < 10:
        df.drop(index, inplace=True)
df = df.reset_index(drop=True)
print(len(df.index))

model = "medium"
name = "medium"
ids = "whisper-"+str(model)

#Whisper
samples_file = []
samples_text = []
samples_pred = []
model = whisper.load_model(model)
for index, row in df.iterrows():
    out = model.transcribe(row['file'])
    samples_text.append(row['text'])
    samples_pred.append(out['text'])

#Post processing
dict_num ={
    "0": "zero",
    "1": "one",
    "2": "two",
    "3": "three",
    "4": "four",
    "5": "five",
    "6": "six",
    "7": "seven",
    "8": "eight",
    "9": "nine",
    "10": "ten",
}

dict_spec ={
    ".": "",
    ",": "",
    "!": "",
```

```
    "?": "",
    "'": "",
    "´": "",
}
#Correcting output
new_samples_pred = []
for i in samples_pred:
    new_seq = i
    for j in dict_num:
        sub = new_seq.replace(j, " "+str(dict_num[j]))
        sub = sub.replace("  "," ")
        new_seq = sub
    for k in dict_spec:
        sub = new_seq.replace(k, str(dict_spec[k]))
        new_seq = sub.lower()
    new_samples_pred.append(new_seq)

wer_metric = load_metric("wer")
val_wer_org = (wer_metric.compute(predictions=samples_pred,
↪  references=samples_text))
val_wer_new = (wer_metric.compute(predictions=new_samples_pred,
↪  references=samples_text))
print("OpenAI Whisper original prediction WER: " + str(round(val_wer_org * 100,
↪  2)) + "%")
print("OpenAI Whisper Corrected prediction WER: " + str(round(val_wer_new *
↪  100, 2)) + "%")

sample_wer = []
for index in range(len(new_samples_pred)):
    sample_pred = [str(new_samples_pred[index])]
    sample_ref = [str(samples_text[index])]
    sample_wer.append(wer_metric.compute(predictions=sample_pred,
    ↪  references=sample_ref))

df_samples = pd.DataFrame(list(zip(samples_text, samples_pred,
↪  new_samples_pred, sample_wer)), columns=['Target text','Original
↪  prediction','Corrected prediction', 'WER sample'])
df_samples.to_pickle("./evaluation/{a}.pickle".format(a=str(ids)+"_dfSamples"))


# Appending parameters to CSV file
headersCSV = ['ID', 'MODEL', 'ORIGINAL WER', 'CORRECTED WER']
dict={'ID': str(ids),
      'MODEL' : str(name),
      'ORIGINAL WER': val_wer_org*100,
      'CORRECTED WER': val_wer_new*100,
      }

with open('./evaluation/performance.csv', 'a', newline='') as write:
    dictwriter_object = DictWriter(write, fieldnames=headersCSV)
    dictwriter_object.writerow(dict)
```

```
print("-"*100)
print("csv-file updated!")
write.close()
```