University of Blida 1

Faculty of Sciences

Computer Science Department

Master Thesis in Computer Science

Option: Software Engineering

Design and Implementation of a Real-time Transcription Solution

Realized by: ALOUI Abdelaali Adel, SEMAN Walid

Supervised by: Dr. Hadjer Ykhlef, SIFI Abdelhak

In front of the jury composed of:

Dr. Meskaldji Khouloud	President
Dr. Cherif-Zahar	Examiner

July 2023

Acknowledgements

We express our sincere gratitude to our supervisors, Miss YAKHLEF.H, senior lecturer class B in the Informatics Department at University Blida 1, and Mr. SIFI.A, software engineer in application systems at ICOSNET, for their invaluable guidance, patience, and unwavering support throughout the project. Their insightful advice and availability greatly contributed to the development of our work.

We would also like to extend our thanks to the staff of ICOSNET for their warm hospitality and for providing us with the opportunity to collaborate on this project. Their cooperation and assistance were instrumental in the successful completion of our work.

Furthermore, we are grateful to the Informatics Department at Saad Dahlab University for the comprehensive educational training and enriching experiences that have shaped our academic journey.

We would like to express our deepest appreciation to our beloved family members, whose unwavering belief in our abilities and continuous encouragement have been a source of motivation throughout this endeavor. Additionally, we are grateful to our circle of friends, who have been supportive and offered their assistance whenever needed.

We also want to send our deepest thanks and appreciations to the open-source community for sharing their work freely and without constrictions.

Without the guidance, support, and encouragement from all these individuals and institutions, this project would not have been possible.

Abstract

In an increasingly interconnected world, effective communication across language barriers is essential. Real-time transcription and translation systems have emerged as solutions to facilitate seamless communication in multilingual settings. This thesis presents a system designed to transcribe English speech, translate it into French, and address challenges in capturing clear audio and handling noisy environments. The system incorporates an automatic speech recognition machine learning model capable of transcribing vocabulary typically used in meetings. It focuses on achieving real-time performance with reasonable latency, even on low-performance hardware. Consequently, our system successfully addressed the challenge of capturing clear audio in noisy environments and transcribing vocabulary commonly used in meetings. Despite acknowledging the constraints in accurately recognizing some words and occasional transcription errors, the system's ability to deliver real-time performance with minimal latency on hardware of modest capabilities is noteworthy and the translation system is robust and Effective at capturing semantics.

Résumé

Dans un monde de plus en plus interconnecté, une communication efficace au-delà des barrières linguistiques est essentielle. Les systèmes de transcription et de traduction en temps réel ont émergé en tant que solutions pour faciliter une communication fluide dans des environnements multilingues. Cette thèse présente un système conçu pour transcrire la parole en anglais, la traduire en français et relever les défis liés à la capture d'un son clair et à la gestion d'environnements bruyants. Le système intègre un modèle d'apprentissage automatique de reconnaissance automatique de la parole capable de transcrire le vocabulaire généralement utilisé lors des réunions. Il vise à atteindre des performances en temps réel avec une latence raisonnable, même sur du matériel peu performant. En conséquence, notre système a relevé avec succès le défi de la capture d'un son clair dans des environnements bruyants et de la transcription du vocabulaire couramment utilisé lors des réunions. Malgré la reconnaissance des contraintes liées à la reconnaissance précise de certains mots et aux erreurs de transcription occasionnelles, la capacité du système à offrir des performances en temps réel avec une latence minimale sur un matériel aux capacités modestes est remarquable et le système de traduction est robuste et efficace dans la capture de la sémantique.

ملخص

في عالم يزداد تداولا وتواصلا، يعد التواصل الفعال عبر حواجز اللغة أمرا أساسيا. ظهرت أنظمة التعرف على الكلام والترجمة في الوقت الحقيقي كحلول لتسهيل التواصل السلس في البيئات متعددة اللغات. تقدم هذه المذكرة نظاما مصممًا للتعرف على الكلام الإنجليزي وترجمته إلى اللغة الفرنسية، كذلك مواجهة التحديات في التقاط الصوت الواضح ومعالجة الصوت في البيئات التي تحتوي على ضجيج. يتضمن النظام نموذج ذكاء اصطناعي للتعرف التلقائي على الكلام قادرا على التعرف على المفردات المستخدمة عادة في الاجتماعات. النظام يركز على تحقيق أداء عالي في الوقت الحقيقي مع تأخير معقول، حتى على الأجهزة ذات الأداء المحدود. وكنتيجة، نجح نظامنا في التغلب على تحدّي التقاط الصوت الواضح في البيئات التي تحتوي على ضجيج والتعرف على المفردات المستخدمة شائعًا في الاجتماعات. على الرغم من بعض القيود في التعرف الدقيق على بعض الكلمات، فإن النظام قادر على تقديم أداء عالي في الوقت الحقيقي مع تأخير معقول على أجهزة ذات قدرات محدودة واما نظام الترجمة فهو فعّال في التقاط المعاني بدقة عالية.

List of Tables

List of Figures

List of Abbreviations

ASR: Automatic Speech Recognition

Hz: hertz

Sr: sampling rate

DFT: Discrete Fourier Transform

STFT: Short-time Fourier Transform

FFT: Fast Fourier Transform

%WER: Word Error Rate

ANN: Artificial Neural Network

GELU: Gaussian Error Linear Unit

RELU: Rectified Linear Unit

MLP: Multi-layer Perceptron

RNN: Recurrent Neural Network

FFNN: feed-forward neural network

TDNN: Time-Delay Neural Network

BLSTM: Bidirectional Long Short-Term Memory

LSTM: Long Short-Term Memory

MFCC: Mel Frequency Cepstral Coefficients

BERT: Bidirectional Encoder Representations from Transformers

CTC: Connectionist Temporal Classification

RNNT: Recurrent Neural Network Transducer

NLP: Natural language processing

AI: artificial intelligent

MT: Machine translation

SMT: Statistical Machine Translation

IBM: International Business Machines

NMT: Neural Machine Translation

VAD: voice activity detection

MHA: multi-head attention

ms: milliseconds

ALP: average log probability

WMT	Workshop on Statistical Machine Translation
BPE	Byte Pair Encoding
BELU	Bilingual Evaluation Understudy
Int8	Integer8 quantization
GUI	graphical user interface

List of content

Introduction

In an increasingly interconnected world, effective communication plays a vital role in bridging language barriers and fostering understanding between individuals. Language diversity, however, poses a significant challenge in situations where **people speaking different languages** need to interact, such as in multilingual meetings. To address this challenge, real-time transcription and translation systems have emerged as a promising solution, enabling seamless communication between individuals who do not share a common language. Typically, these systems are composed of multiple components working in tandem. Firstly, **real-time audio capture** is employed to record the spoken words. Subsequently, **Automatic Speech Recognition** techniques are utilized to transcribe the audio into textual format, effectively converting speech into written form. The transcribed text is then subjected to **machine translation** techniques for the purpose of translation. Finally, the translated text is presented to the user through a **graphical user interface**.

There are **different systems available** that perform either some or all of the required tasks, such as **open-source solutions**. Open-source systems often **rely on high-quality audio clarity** and encounter difficulties in comprehending diverse accents. Additionally, these systems encounter obstacles when dealing with **noisy background environments** and often struggle to operate in real-time with **acceptable latency**. Furthermore, capturing the **vocabulary commonly used in meeting** settings poses another challenge for these systems as they are intended for general use and long-form audio.

In our work, we specifically focused on **addressing these shortcomings** by building an **English to French** system that excels in **capturing clear audio**, even in **noisy environments**, through the use of a machine learning **voice activity detection** model. Additionally, we placed great emphasis on enhancing the **Automatic Speech Recognition** system's ability to accurately transcribe English speech containing vocabulary typically used in meetings by applying fine-tuning techniques to a pre-trained machine learning model. **Furthermore, we prioritized the development of a real-time system with reasonable latency, optimized to run efficiently and effectively translate English speech to French, even on low-performance hardware.**

In this thesis, **we begin by** providing a comprehensive explanation of **speech recognition** and the **machine learning techniques** employed in the **transcription process**. Moving forward, our attention shifts to the **translation task** in the **second chapter**, where we delve into **Natural Language Processing approaches**, specifically focusing on machine translation and comprehending its underlying processes. In the **third chapter**, we present the system's design and architecture, outlining the **real-time audio acquisition** and preprocessing, as well as the transcription and translation models. Noteworthy decisions made for each model are emphasized. Lastly, the **fourth chapter** encompasses implementation, evaluation, and the development of a graphical user interface, demonstrating the tools and libraries utilized throughout the project.

Chapter I

Speech Recognition

I.1 Introduction

Within this chapter, we explore the field of transcription and specifically delve into speech recognition. We cover the speech characteristics used in the transcription pipeline, including feature engineering and the extraction of log Mel spectrograms using various Fourier transform algorithms. Additionally, we present the application of machine learning for speech recognition, starting with an overview of fundamental machine learning concepts and categories. Our focus then shifts towards artificial neural networks (ANNs), where we examine their building blocks, showcasing an example of a perceptron. Finally, we emphasize the transformative potential of the transformer architecture along with some previous work on Automatic speech recognition.

I.2 What Is Speech Recognition

Speech recognition, also known as automatic speech recognition (ASR), enables computers to interpret and understand spoken language [1]. Specifically, it involves transforming spoken words or phrases into text that a computer can understand and process. To this end, algorithms are used to analyze audio signals to extract characteristics of speech [2]. Speech is usually depicted as a sound-wave Figure I.1 which is a representation of a physical disturbance that caused a vibration in the air around the source of sound [3].

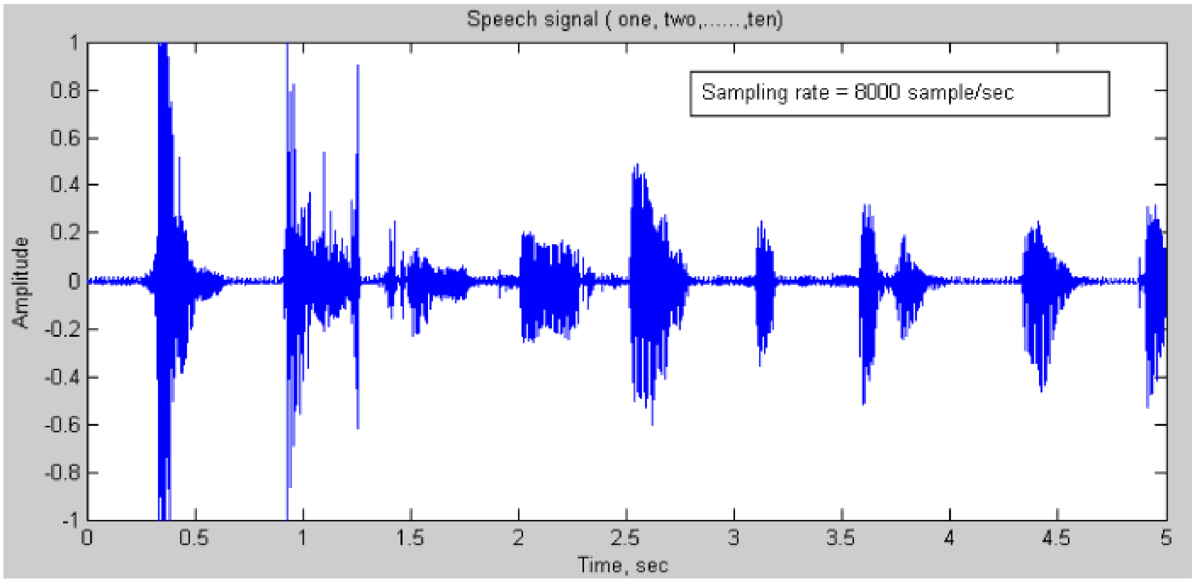


Figure I.1: Speech signal representation sampled at 8 kHz [4].

I.3 Applications

Speech recognition has many applications in the real world such as:

- Personal assistants which are software applications that use speech recognition technology to interpret voice commands and respond accordingly like Seri and Alexa [1].
- Voice biometrics, this means the programs that use voice characteristics and patterns to authenticate or confirm a person's identity like forensic investigations or surveillance systems [5].
- Transcription which is a powerful tool for capturing spoken language and translating it into written text, and it is used in a variety of fields including healthcare, legal and entertainment.

I.4 Speech characteristics

Audio has many characteristics we need in speech recognition such as:

- Frequency is the Vibrational motion that repeats itself in a regular interval of time called the period [6] and it is measured in hertz (Hz),
- Duration refers to the length of time a sound persists. It is typically measured in seconds, and it represents how long a sound wave lasts, this is an important factor in analyzing and manipulating sound recordings, because the longer the duration, the easier it is to lose context due to hardware limitations.
- Pitch, is how humans hear the sounds and it can be different from one to another and it is scaled by frequencies [6],
- Timbre is the difference between two sounds with similar pitch, loudness and duration. It helps us more to define the difference between musical instruments [6]. Figure I.2 illustrates these characteristics.

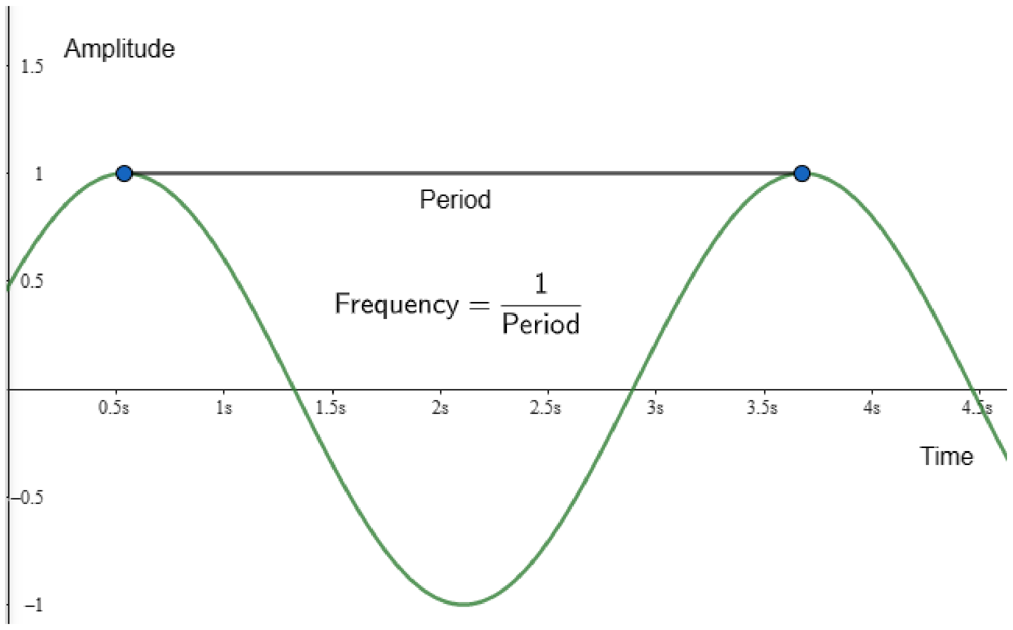


Figure I.2 : Audio characteristics.

I.5 Speech recognition design

The design process begins first by **data collection**, which is gathering audio samples and their corresponding transcription that will be employed to train and test the speech recognition system. Then we **pre-process** the collected data for analysis by Digitization, segmentation, noise reduction or data augmentation. We then move on to **feature extraction**, features that are measurable characteristics of speech signal in a form that can be processed by a computer [7] and extracting them from the set of pre-processed data is needed to train our system. Following that, **language modeling**, which develops a probabilistic word sequence system that predicts the most probable word sequence for the input speech signal. After this **training and testing phase**, we train the speech recognition system using a large amount of speech dataset and testing the accuracy of the system using a separate dataset. Finally, we implement and deploy a speech recognition system in a real environment, such as real-time speech-to-text transcription.

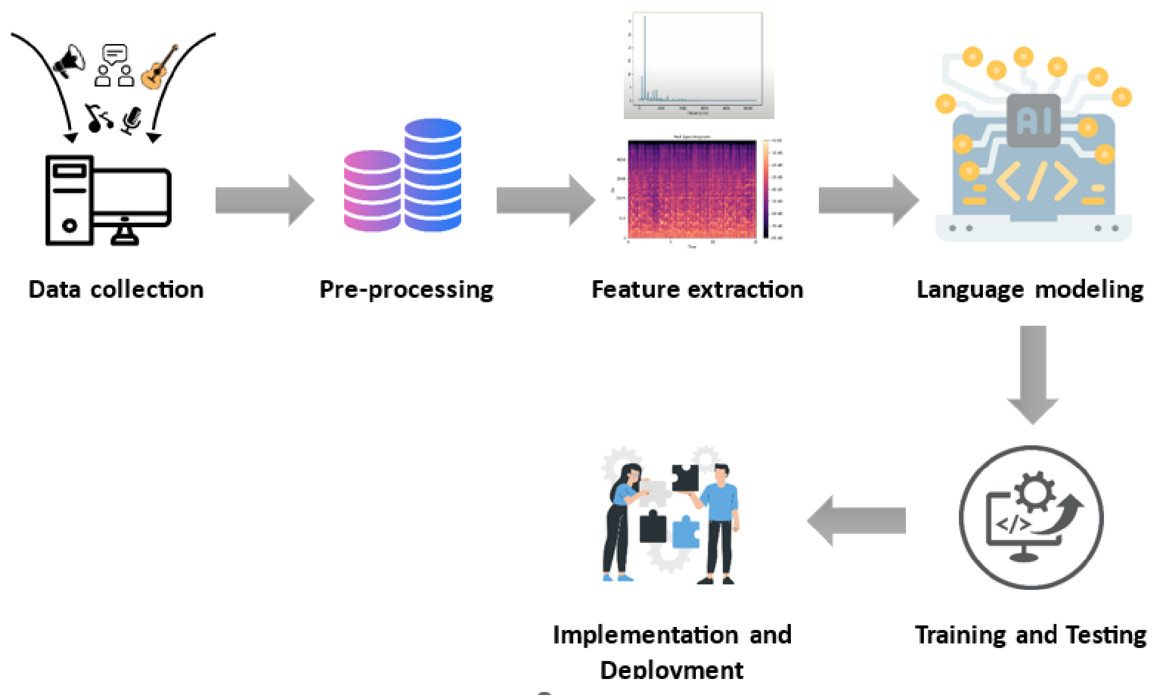


Figure I.3: Automatic speech recognition pipeline.

Features and their extraction process play a crucial role in our system serving as the input information that enables accurate transcription of speech. Further details will be discussed in coming sections.

I.6 Feature extraction

Common features used in speech recognition include Mel-frequency cepstral coefficients and log-Mel spectrograms because of their efficacy in representing the characteristics of human speech.

The choice of features can greatly affect the accuracy of speech recognition systems. In our study, we chose log-Mel spectrogram because of its logarithmic scale that provides a more perceptually relevant representation of the frequency content of the speech signal [8]. To clarify, the section below represents how we extract the Log-Mel spectrogram feature specifying its steps.

I.6.1 Fourier transform

To extract log-Mel spectrogram features we need to convert from the time domain which is the representation of the signal in time to the frequency domain which is the representation in terms of their frequency components. To be able to do that, we need a computational function called the Fourier transform. However, when we deal with any audio signal today, we deal with a discrete signal that is used and manipulated with the digital machines, and the signal we get is continuous signal, so we need to pre-process it by converting it from continuous signal to digital signal.

The method used to get a digital signal is sampling and quantizing it by measuring the amplitude of the signal at regular intervals in time (period T) to a finite set of discrete values, and it is determined by the bit-depth of the digital signal. The higher the bit-depth, the more accurate and the closer the digital signal is to the original continuous signal. The rate at which these measurements are taken is called the sampling rate (Sr).

The Fourier transform function works on continuous signals, but for discrete signals, we use the **Discrete Fourier Transform** (DFT) instead. The DFT enables us to analyze the frequency components of the signal. By applying the DFT, we obtain a frequency domain representation that is independent of time, as shown in Figure I.4.

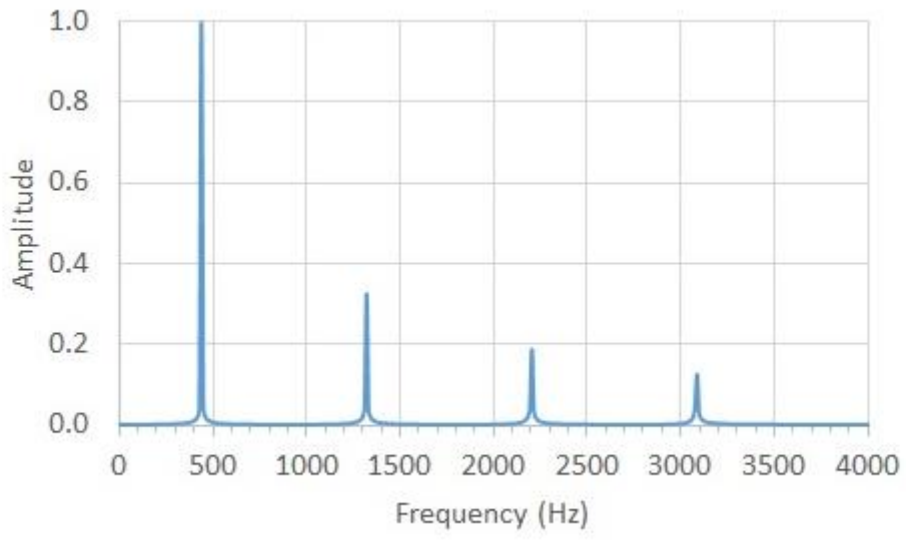


Figure I.4: Frequency domain representation [9].

However, to capture the evolution of frequency components over time, we need to use **the Short-Time Fourier Transform** (STFT). The STFT is essentially the DFT applied to segments of the signal called frames [10]. These frames are obtained by dividing the signal into equal-sized time windows. The STFT equation I.1. is defined as:

$$\boldsymbol{STFT(n,w) = \sum_{k=-\infty}^{\infty} x(k)w(k-n)\, e^{-iwk}} \tag{I.1}$$

x(k) is the signal, w(k) is a window function of finite support, n denotes the time index, omega is the frequency index, i is the imaginary unit.

Applying this function will lead to another problem called spectral leakage, which is a discontinuity in the edges of frame intervals; it manifests as a high-frequency component that is not present in the original signal.

To reduce the spectral leakage, we use a mathematical function called **Hann windowing I.2.** that is zero-valued outside of the chosen interval.

$$\boldsymbol{w(n) = 0.5 * \left(1 - cos\left(\frac{2\pi n}{(N-1)}\right)\right)} \qquad (\mathrm{I.2})$$

w(n) is the value of the window at index n, n is the index of the sample within the window (0 ≤ n ≤ N-1), N is the total number of samples in the window.

However, the application of a windowing function to the frames results in a loss of frequency information at the frame edges. To mitigate this issue, we adopt an overlapping strategy for the frames, ensuring that adjacent frames share a portion of their data.

By incorporating the **Fast Fourier Transform** (FFT) algorithm into the DFT computation, the STFT can be computed efficiently. The FFT is an algorithm that computes the DFT with reduced computational complexity, making it widely used in practice for fast frequency analysis.

As a result of this process, we obtained the spectrogram shown in Figure I.5 where Frequencies are shown increasing up the vertical axis, and time on the horizontal axis. The legend to the right shows that the color intensity increases with the density.

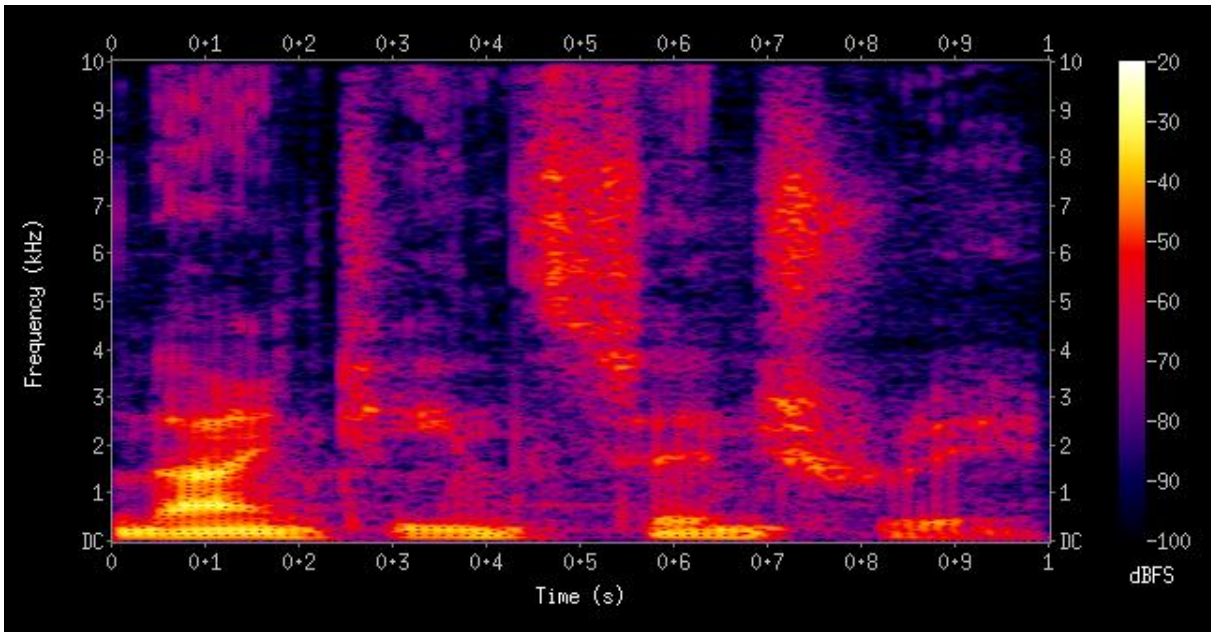


Figure I.5: Spectrogram of an audio file with the spoken phrase "Nineteenth century" [11].

I.6.2 Logarithmic Mel-spectrogram

Normal spectrograms express the frequency linearly which is a problem regarding the human's perception. For a more perceptual representation in which pitch differences are easier to discern, we will use the Mel concept and transform the frequency in a logarithmic form. This is done with a series of triangular overlapping filters whose peaks are spaced according to the Mel scale. The output of these filters is typically logarithmically compressed to produce the final logarithmic Mel-spectrogram representation Figure I.6.

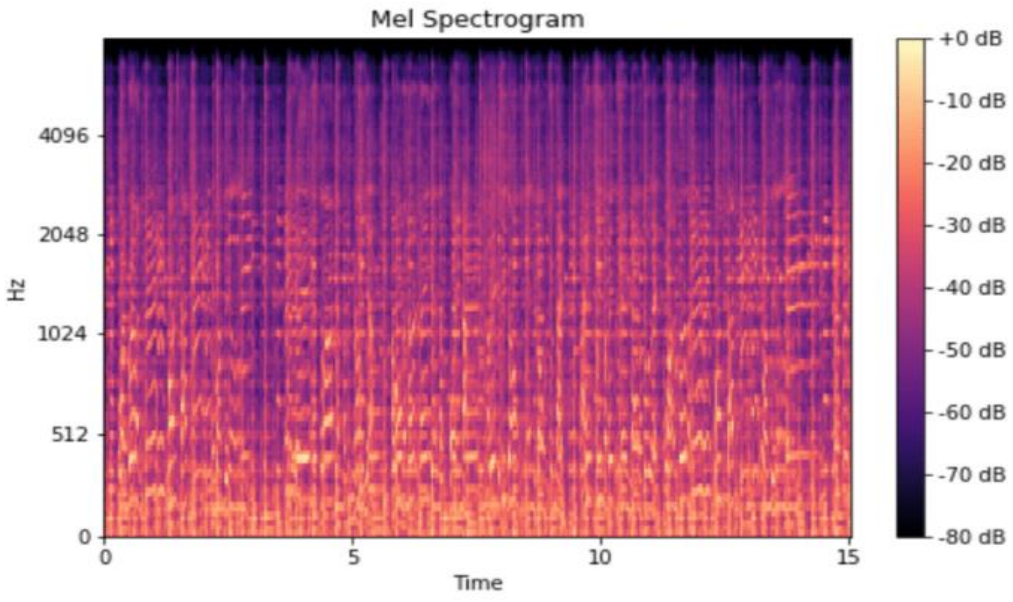


Figure I.6: Representation of Mel spectrogram [12].

I.7 Machine Learning for Speech Recognition

I.7.1 Fundamentals of machine learning

Machine learning, a field within artificial intelligence, has emerged as a powerful approach for analyzing and extracting insights from complex data [13]. It encompasses a wide range of algorithms and techniques that enable computers to learn patterns and make predictions without explicit programming. There are three types of machine learning algorithms [14]: **supervised**, **unsupervised**, and **reinforcement learning**.

Supervised learning involves training a model using **labeled data**, where the input features and corresponding output labels are provided. The goal is to learn the **mapping** between the input and output to make accurate predictions on new, unseen data. **In our case** we use supervised learning to train models using labeled dataset to transcribe spoken language into written text.

Supervised learning can be further categorized into two main types: **classification** where the output labels are discrete and represent different classes or categories. The goal is to train a model that can assign the correct class label to new input data. **For example**, email spam detection is a classification task where the model predicts whether an email is spam or not based on features like subject line, sender, and content. And we also have **regression** where the output labels are continuous numerical values.

In contrast, **unsupervised learning** deals with **unlabeled data**. Without explicit guidance, unsupervised learning algorithms aim to discover patterns, structures, and relationships within the data. In addition, **reinforcement learning** focuses on training an agent to interact with an environment and learn optimal actions based on rewards or punishments. For example, a model called **AlphaGo** trained to play the board game “Go” [15].

Labeled datasets are typically divided into **two sets**: **training** and **testing** sets [16], with typical ratios of 80%-20% or 70%-30%. Datasets can be clean or noisy. While clean datasets are carefully controlled and free of errors, outliers, or inconsistencies, noisy datasets mirror real-world difficulties since they contain mislabeled samples and incomplete data. The **training set** is used to **train** the model so it can generalize and make accurate predictions on unseen data. While the **testing set** provides an unbiased **evaluation** of the model's performance. It consists of unseen data that the model has not encountered during training. Several evaluation metrics exist, and in the context of speech recognition, **%WER (Word Error Rate)** [17] is a common metric used for evaluating the performance of the model's predictions by quantifying the rate of errors relative to a reference or ground truth. **A lower %WER indicates better performance.** The following is the %WER equation I.3.

$$\%\boldsymbol{WER} = (\boldsymbol{S} + \boldsymbol{D} + \boldsymbol{I})/\boldsymbol{N} \qquad (\mathrm{I.3})$$

S is the number of **substitution errors** (words in the reference transcript that are replaced by incorrect words in the output of the system). **D** is the number of **deletion errors** (words in the reference transcript that are missing in the output of the system). **I** is the number of **insertion errors** (extra words in the system's output that are not present in the reference transcript). **N** is the **total number of words** in the reference transcript

I.7.2 Limitations of traditional machine learning

Traditional machine learning approaches in general across the fields are subject to several limitations [18]. One major drawback is their reliance on **handcrafted features**, which require expert knowledge and **manual engineering**. This feature engineering process can be time-consuming and may not fully capture the intricate characteristics of speech signals. Moreover, traditional machine learning models struggle to handle **temporal dependencies** which refer to the relationships and patterns that exist over time in spoken language. **Contextual information** present in speech data, hindering their ability to model complex patterns effectively. Additionally, these models may face challenges in adapting to variations in acoustic conditions, such as **background noise** and speaker variability like different **accents**, as they are often trained on clean and controlled data.

To address these limitations, deep learning has emerged as a powerful solution in speech recognition. **Deep learning** models, such as **Transformers**, can automatically learn high-level representations from raw speech data without the need for explicit **feature engineering**. These models excel at capturing **temporal dependencies** and extracting complex patterns, enabling them to achieve state-of-the-art performance in speech recognition tasks.

I.7.3 Artificial Neural Networks: From Perceptrons to Transformers

Artificial Neural Networks (ANNs) [19] are machine learning models that simulate the behavior of the human brain. **Neurons**, or nodes, are the basic components of ANNs. They receive **input** data, perform **computations**, and produce **outputs**. Neurons are interconnected and organized into **layers**. The most basic form of an (ANN) is a single-layer **perceptron** illustrated in Figure I.7**,** often used for binary classification tasks It consists of multiple **units**, where one unit takes input values, another unit multiplies them by corresponding **weights**, and later a unit produces an output based on an **activation function** on the weighted sum of input, The activation function plays a crucial role in determining if neurons activate based on a specific threshold. Additionally, the activation function introduces **non-linear transformations** to the weighted inputs of the neurons. This nonlinearity enables the neural network to capture **complicated patterns** and enhance its ability to make accurate predictions. One such **function** is **GELU (Gaussian Error Linear Unit) I.5. [20]** That depends on **tanh function I.4.** Both defined as**:**

$$\boldsymbol{tanh(x) = (e^{x} - e^{-x}) / (e^{x} + e^{-x})} \qquad (\mathrm{I}.4)$$

$$\boldsymbol{GELU(x) = 0.5 \times x \times \left(1 + tanh\left(\sqrt{\frac{2}{\pi}} \times (x + 0.044715 \times x^{3})\right)\right)} \qquad (\mathrm{I}.5)$$

x is the input and **e** is **Euler's number** equal to 2.71828.

To account for bias in the neural network, a **bias term** is added to each neuron. The bias term is an additional learnable parameter that allows the network to shift the activation function's threshold, providing more flexibility in the decision-making process.

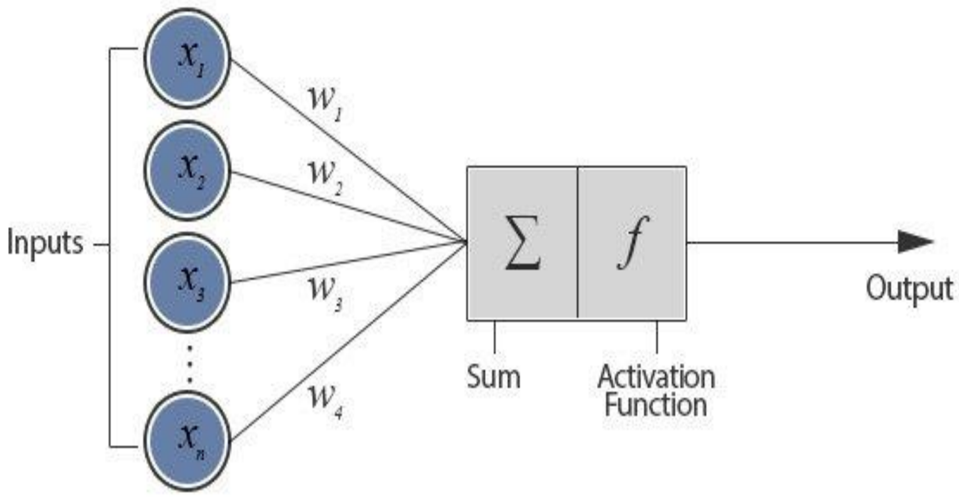


Figure I.7: Single-layer perceptron [21].

The perceptron model is limited in its ability to handle complex non-linear patterns in data. To address these limitations, more advanced ANNs, like **Multi-layer Perceptrons (MLPs)** were developed. MLPs address perceptron limitations by introducing multiple **hidden layers** in addition to the **input and output layers** as illustrated in **Figure II.8**. MLPs utilize hidden layers to capture and represent complex patterns in the data. The input layer receives the initial data, the hidden layers transform and learn complex patterns, and the output layer generates the final output.

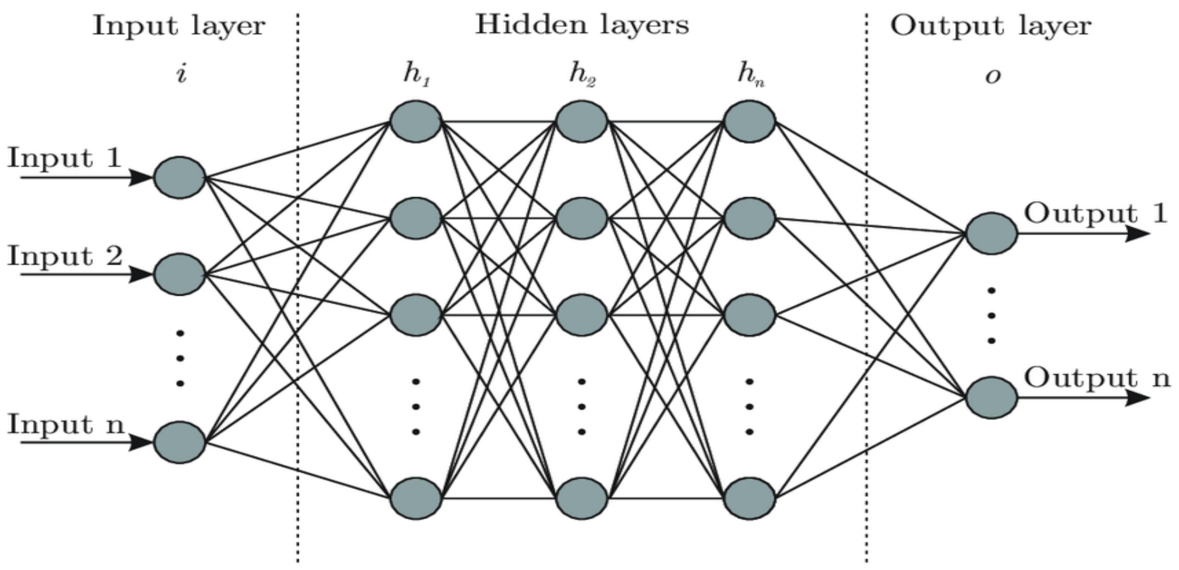


Figure I.8: An example of an MLP with an input layer, one or more hidden layers, and an output layer [22].

MLPs are powerful models for **non-sequential data** but lack the ability to capture and leverage the temporal dependencies in **sequential data**. Sequential data refers to data with an inherent order or temporal dependency between elements, such as time series or natural language sentences. To address this limitation, **sequential models** were developed specifically for handling sequential data. These models, such as Recurrent Neural Networks (**RNNs)** and **Transformers** [23], were developed to handle sequential data by capturing its temporal information. RNNs use recurrent connections to maintain memory and capture dependencies between elements in the sequence. Transformers, on the other hand, employ **self-attention** mechanisms to handle long-range dependencies more effectively. Transformers have gained popularity for their ability to parallelize computations and capture global context, making them superior **[24]** to RNNs in certain scenarios. In our system design, we have employed the transformers architecture.

I.7.4 Transformers

Transformer architecture, illustrated in **Figure I.9**, incorporates components such as **self-attention** and **multi-head attention** to enable efficient and effective information processing.

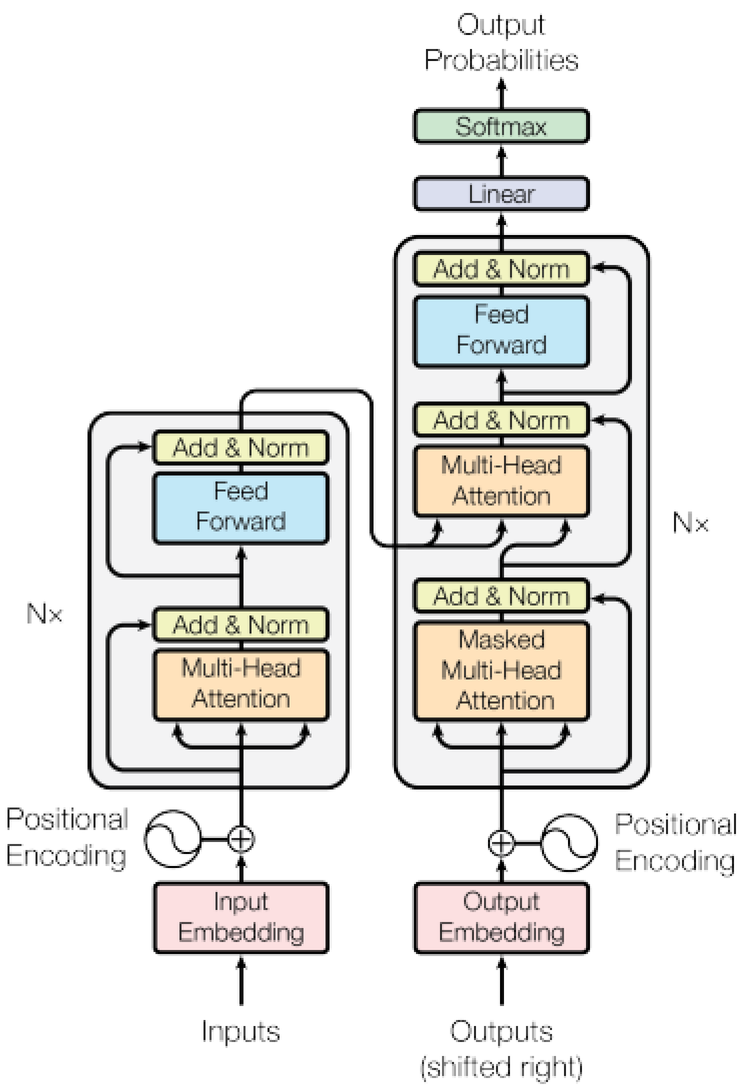


Figure I.9: Transformer model architecture [23].

I.7.4.1 Attention

Self-attention and multi-head attention are crucial constituents that enable the model to evaluate the importance of various segments of the input while making predictions.

I.7.4.1.1 Self-attention

Self-attention which formula is represented in I.6. Is a technique that empowers a transformer architecture to **attend** to different **positions** of the input sequence in order to compute a representation of the input, refer to Figure I.10. First, the input is transformed into **three vectors**, **queries** ‘Q’, **keys** ‘K’, and **values** ‘V’, by multiplying the input by **learned weight matrices**. Then, the **dot or scalar product** a **measure of vector alignment and similarity in direction** is calculated between the **query vector** and the **key vector** after the **key vector is transposed**, which involves flipping its rows and columns, then dividing each by the **square root of 'dk'** (the dimension of queries and keys), followed by applying a **SoftMax function** represented in I.7. To the result to obtain a set of attention weights. **SoftMax function** takes a vector of real numbers as input and transforms them into a probability distribution that adds up to **1**. Finally, the weighted sum of the value vectors is taken, and the resulting vector is the output that is computed by the function:

$$\boldsymbol{Attention(Q,K,V\,) = \ softmax\left(\frac{QK^T}{\sqrt{dk}}\right)V} \qquad (\mathrm{I.6})$$

$$\boldsymbol{softmax\ = \frac{exp(xi)}{sum\big(exp(xj)\big)}} \qquad (\mathrm{I.7})$$

For $softmax$ function **xi** represents the score associated with a particular token, **exp(xi)** is the exponential of the score, which ensures that the resulting values are positive. **sum(exp(xj))** represents the sum of the exponentiated scores across all the tokens.

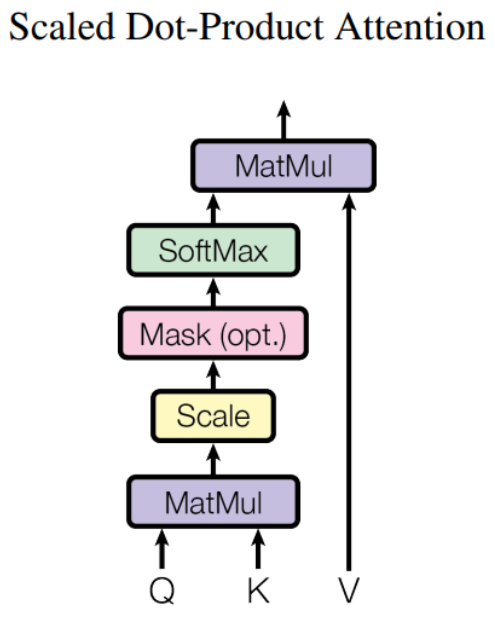


Figure I.10: Self attention layer [23].

I.7.4.1.2 Multi-head attention

Multi-head attention is an enhanced version of self-attention that allows the model to attend to different parts of the input simultaneously, providing a richer representation. In this mechanism Figure I.11, the input is divided into multiple subspaces, and the self-attention operation is independently performed on each subspace. This enables the model to focus on different aspects of the input.

After applying **self-attention** to each subspace, the **outputs** of the sub-attention heads are **concatenated together.** This concatenation captures a diverse set of information and perspectives from the input. To **restore the original dimensionality**, the concatenated outputs are linearly transformed **using a projection matrix W^O**

Mathematically, the multi-head attention can be defined as in I.8. :

$$\boldsymbol{MultiHead(Q,K,V)} = \boldsymbol{Concat(head_1,\ldots,head_h)W^O} \quad (\mathrm{I.8})$$

Where each $head_i$ represented in I.9**. performs the self-attention operation as:**

$$\boldsymbol{head_i} = \boldsymbol{Attention\left(QW_i^Q, KW_i^K, VW_i^V\right)} \qquad (\mathrm{I.9})$$

The projection matricesW_i^Q,W_i^K and W_i^V map the inputs Q, K, V to the appropriate dimensions (d_{model} $\times$ d_k) and (d_{model} $\times$ d_v), respectively.

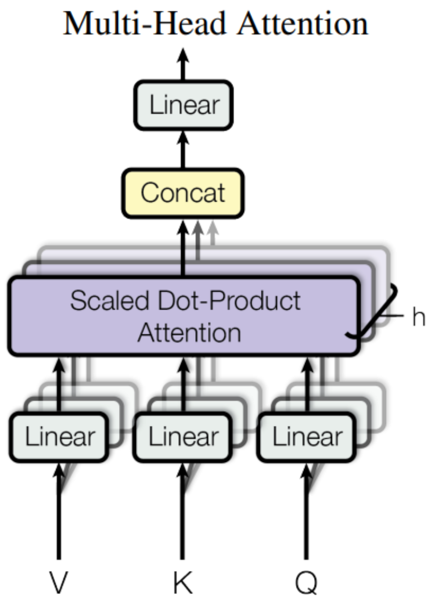


Figure I.11: Multi-Head Attention consisting of several attention layers running in parallel [23].

I.7.4.2 Transformer architecture

In the context of speech recognition, the network mechanics can be explained as follows: The input, such as a log-Mel spectrogram or audio waveform, is first passed through the **embedding layer** to assign a vector representation to each frame. To incorporate **positional information**, the embedding vectors are augmented with positional encoding using the following equations represented in I.10. And I.11.

$$PE(pos, 2i) = sin\left(\frac{pos}{10000^{\left(\frac{2i}{dmodel}\right)}}\right) \qquad (I.10)$$

$$PE(pos, 2i+1) = cos\left(\frac{pos}{10000^{\left(\frac{2i}{dmodel}\right)}}\right) \qquad (I.11)$$

For each position (pos) and dimension (2i or 2i+1) in the embedding, the positional encoding values are calculated using the corresponding sine or cosine functions. These values are then added to their respective embedding vectors. This ensures that **even time steps** receive positional encoding through the sine function, while **odd time steps** receive it through the cosine function. By incorporating these mathematical operations, the network gains positional context, enabling it to capture positional dependencies within the sequential data effectively.

I.7.4.2.1 Encoder

The encoder layer plays a crucial role in transforming the input sequence into an abstract continuous representation that captures the complete information of the sequence. It comprises **two submodules**: **multi-head attention** (as mentioned earlier) and a **feed-forward neural network (FFNN)**. The multi-head attention mechanism enables the model to attend to different parts of the input simultaneously, providing a more comprehensive representation. The feed-forward neural network, consisting of linear layers with a RELU activation in between, further processes the attention output, enriching its representation.

Residual connections are incorporated around each submodule, where the **output is added to the input**. Which acts as a form of memory within the network. Additionally, a **normalization layer** stabilizes the network, contributing to more efficient training.

These operations are essential for encoding the input into a continuous representation with attention information. This encoding assists the decoder in focusing on relevant words during the decoding process. By **stacking** the **encoder layer multiple times**, we can further enhance the encoding by allowing each layer to learn distinct attention representations. This has the potential to significantly improve the predictive capability of the transformer network.

I.7.4.2.2 Decoder

The **decoder's primary** objective is to **generate text sequences based on the encoded information**. The decoder **consists** of multiple sublayers, including **two multi-head attention layers**, a **FFNN layer**, **residual connections**, and **layer normalization**. During the decoding process, the decoder takes previous outputs as inputs and utilizes attention information from the encoder. The **decoding continues** until an **end token** is generated, indicating the completion of the text sequence.

Initially, the input undergoes an embedding layer to obtain vector representations, followed by a positional encoding layer that incorporates positional information.

Next, the input is fed into the **first multi-head attention layer**. Given the autoregressive nature of the decoder, a **masking method** is employed to prevent **future token conditioning**. This masking ensures that the model focuses solely on the relevant words by **assigning zero attention to future tokens**. The output of the first multi-head attention layer provides a masked output vector that guides attention during the decoding process. Moving on to the second multi-head attention layer, the encoder's outputs **serve as queries and keys**, while the output from the first multi-head attention layer **serves as values**. This alignment mechanism allows the decoder to determine the relevant encoder inputs to attend to. The output of the second multi-head attention layer is further processed through a FFNN layer.

Finally, the output of the last FFNN layer passes through a linear layer acting as a classifier. A **SoftMax layer** then produces **probability scores for each word** class. The predicted word corresponds to the index with the **highest probability score**. This output is added to the decoder input list, and the decoding process continues until an **end token** is predicted. To enhance the decoder's performance, **multiple layers are stacked**, with each layer taking input from the encoder and the preceding layers. This layer stacking enables the model to learn diverse combinations of attention patterns from its attention heads, potentially improving the accuracy of text sequence predictions in the context of speech recognition.

I.7.4.3 Training

To train a transformer model [25], the key steps involve **preparing the training data**, defining the **loss function**, and **optimizing** the **model parameters**. The training data consists of **input sequences**, such as log-Mel spectrograms or audio features, along with their corresponding **target outputs**. In transformers the encoder and decoder work together to generate accurate predictions. During training, the model learns to **minimize** the difference between its predictions and the target outputs. This is achieved by defining a suitable loss function, such as the **cross-entropy loss** [14], which quantifies the difference between **predicted probabilities** and **true labels**, the following is the equation represented in I.12.

$$\boldsymbol{L = -\sum_{i=1}^{n} t_i log(p_i)} \qquad (\mathrm{I}.12)$$

L represents the cross-entropy loss, **t** denotes the true probability distribution of the target labels, **p** represents the predicted probability distribution.

Gradients are crucial in **minimizing the loss function**. They indicate the direction and magnitude of adjustments needed to improve the model's predictions. By following the gradients, we update the model's parameters to move towards lower loss and enhance the model's performance. **Backpropagation** is the algorithm used to efficiently compute these gradients. It involves **propagating the errors** from the **output layer back to the input layer** of the model. By applying the chain rule of calculus, the errors are distributed across the layers, and the gradients are calculated with respect to each parameter. This allows the model to understand how changes in the parameters affect the overall prediction error. The gradients obtained from backpropagation guide the **optimization process**. They provide information on how the model parameters should be updated to reduce the loss function. Optimization algorithms, such as **Adam [26]**, utilize these gradients to iteratively adjust the parameters in a way that minimizes the loss, the following is it equation I.13.

$$\boldsymbol{O_{new} = O_{old} - (learning\ rate * m) / (\sqrt{v} + \varepsilon)} \qquad (\mathrm{I}.13)$$

$\mathrm{O_{new}} = \mathrm{O_{old}}$ Are the new and old values of the model parameter o, respectively **learning rate** is the step size that controls the magnitude of parameter updates, m average value of the gradients, v Variance of the gradients, **ε** is a small value added for numerical stability to avoid division by zero.

To train a model, there are two approaches: **starting from scratch** or utilizing transfer learning. In the first approach, the model is trained from random parameters using gradients to gradually improve its performance. In the second approach, a **pretrained model** is used as a starting point and its parameters are **fine-tuned** on a specific task.

I.8 Previous work on Speech Recognition

Kaldi [27]: Kaldi is an open-source toolkit for building and working with automatic speech recognition (ASR) systems, written in C++ and released in 2011 and it is widely used in both research and industry. One of it prominent models for **English ASR** is **ASpIRE Chain Model released in September 2019[28],** it was trained on **Fisher English dataset [29]** that has been augmented with impulse responses and noises to create multi-condition training. **ASpIRE** is based on **TDNN (Time-Delay Neural Network) and BLSTM (Bidirectional Long Short-Term Memory).**

DeepSpeech [30]: **Mozilla DeepSpeech** is an open-source implementation of a **neural network architecture introduced by Baidu in 2017**[31], which utilizes Recurrent Neural Networks (**RNNs**) with Long Short-Term Memory (**LSTM**) units and Mel Frequency Cepstral Coefficients (**MFCCs**) as features. The latest model release from Mozilla was in December 2020, it was trained on the following datasets: Fisher [29], LibriSpeech [32], Common Voice [33], SWITCHBOARD [34] and approximately 1700 hours of transcribed WAMU (NPR) radio shows explicitly licensed to Mozilla.

HuBERT [35]: **HuBERT released in Jun 2021 , Facebook's** latest approach for learning self-supervised speech representations using offline k-means clustering algorithm inspired by DeepCluster [36] method for self-supervised visual learning introduced in 2018 that learns a neural network’s parameters and their cluster assignment , HuBERT further benefited from Google’s Bidirectional Encoder Representations from Transformers (BERT) [37] by leveraging its masked prediction loss over sequences method to showcase the sequential nature of speech, Making it possible to **use HuBERT for Automatic Speech recognition**. Among Facebooks released HuBERT models , there are two models **pretrained** on Libri-Light [38] which is **more than 60,000 hours of unlabeled speech with limited supervision of only 10h,**

1h or 10 minutes of labelled speech , then those models where then **fine-tuned on 960h of LibriSpeech** [32] dataset .

NVIDIA NeMo [39]: NVIDIA NeMo is a toolkit for speech recognition with pre-trained models. Its most prominent ASR models are based on Conformer Architecture introduced by Google Brain in 2020[40] which **integrate convolutional layers into the Transformer architecture to capture both local and global dependencies**, NVidia has two variants: NVIDIA **Conformer-Connectionist Temporal Classification (CTC) released on April 2022** that uses CTC loss function and decoding instead of **Recurrent Neural Network Transducer (RNNT)/Transducer loss function** (**non-autoregressive**) and NVIDIA **Conformer-Transducer** (**autoregressive**) **released on June 2022** same as the one introduced in Google paper and uses **RNNT/Transducer loss/decoder**. As for the datasets, NVIDIA Conformer has been trained on a large and varied list of datasets, Such as: LibriSpeech [32], Fisher [29], Switchboard [34], Common Voice [33], People's Speech [41] 12,000 hours dataset and many others.

Whisper [42]: Whisper is a general-purpose speech recognition model. It is trained on a large dataset of diverse audio and is also a multitasking model that can perform multilingual speech recognition, speech translation, and language identification. It is based on a slightly modified vanilla transformer architecture.

I.9 Conclusion

This chapter offers a fundamental overview of speech recognition, highlighting the process of extracting the log-Mel spectrogram, which will serve as input for transcription. Furthermore, we delve into the various approaches of machine learning applied to transcription, focusing on transformers. We also showed some previous work on Automatic speech recognition. In the upcoming chapter, we will shift our focus to the task of translation.

Chapter II

Natural Language Processing

II.1 Introduction

In this chapter, we introduce the second part of our system, which focuses on machine translation. We explore the field of natural language processing and its various tasks. Specifically, we delve into machine translation and discuss the different approaches utilized, along with providing an overview of the translation process itself and some previous work.

II.2 What is natural language processing

Natural language processing (NLP) is a subfield of AI that focuses on enabling machines to understand, interpret, and generate human language. It encompasses a range of tasks such as language translation, sentiment analysis, information retrieval, text summarization, and more. NLP plays a crucial role in bridging the gap between human communication and machine understanding [43].

NLP encompasses various tasks with real-world applications. One such task is **sentiment analysis**, which involves determining the sentiment expressed in a given text [44]. Another application is **text summarization**, where extractive summarization systems generate concise summaries of provided texts [45]. Lastly, **machine translation** systems like Google Translate offer translation services for text, websites, and documents in multiple languages.

II.3 Machine translation

Machine translation is the field of study that focuses on developing systems capable of automatically translating text or speech from one language to another. It aims to bridge the language barrier and enable effective communication across different languages [44]. The section below represents Machine translation approaches and its categorization.

II.3.1 Statistical Machine Translation

Statistical Machine Translation (SMT) has been a prominent approach in machine translation for many years. It relies on statistical models to align and translate text. SMT operates on the basis of a large parallel corpora, extracting translation patterns and probabilities to generate translations [46].

One widely used SMT model is the phrase-based model, which breaks sentences into smaller units and translates them in chunks. Alignment models, such as IBM models, help identify word alignments between source and target languages. However, SMT has its limitations. It struggles with handling long-range dependencies, maintaining context, and generating fluent translations [46], also it tends to be slow.

II.3.2 Neural Machine Translation

Neural Machine Translation (NMT) represents a significant advancement in machine translation, overcoming some of the limitations of SMT. NMT models use neural networks, particularly the encoder-decoder framework, to learn and generate translations [44]. Unlike SMT, NMT models operate at the sentence level and consider the entire source sentence during translation.

A crucial component of NMT is attention mechanism, which allows the model to focus on relevant parts of the source sentence when generating translations. Attention mechanisms help address the issue of long-range dependencies and enable the model to capture contextual information more effectively. NMT models have shown improved fluency, accuracy, and the ability to handle complex linguistic structures [44], also it tends to be fast.

The introduction of transformers has revolutionized NMT. Transformers leverage self-attention mechanisms to capture dependencies between words and learn representations of the source sentence. Self-attention allows the model to weigh different parts of the sentence based on their relevance, enabling more accurate and context-aware translations [23]. As shown in the previous chapter.

Transformers have proven to be highly effective in capturing long-range dependencies, making them suitable for machine translation tasks. They have surpassed traditional models in terms of translation quality, achieving state-of-the-art results on various benchmark datasets [44].

II.4 NLP design for Machine translation

In order to gain insights into the underlying mechanisms of NLP design for machine translation, it is essential to comprehensively investigate the following key components and processes:

II.4.1 Data collection

Data collection is the first step to train our machine translation system. The quality and quantity of the data used for training can significantly impact the accuracy and performance of the system. High-quality data that is relevant to the target translation task can reduce the need for extensive preprocessing and improve the translation accuracy.

II.4.2 Pre-processing

Pre-processing the data is a necessary step for getting a better and efficient translating, and this step include:

- **Tokenization** is the first step to convert the raw text into a sequence of tokens that can be processed by the model. Tokens serve as the basic building blocks for language understanding [47].
- **Elimination stops words:** Removing commonly occurring words (articles, prepositions, pronouns) with low semantic value. It adds little meaning to the translation task and can be safely removed to reduce the vocabulary size and noise in the data.
- **Lowercasing:** Converting all the text to lowercase. It helps in standardizing the text and reducing the vocabulary size by merging words with different capitalizations.
- **Removing punctuation:** Eliminating punctuation marks from the text. Punctuation marks typically do not carry significant meaning in machine translation tasks. Removing them simplifies the text and reduces noise in the data.
- **Text normalization:** Applying normalization techniques, such as stemming or lemmatization, to reduce inflected or variant forms of words. It helps in reducing the vocabulary size and capturing the essence of words by converting them to their base or canonical forms [44].

II.4.3 Model Architecture

This model is often based on transformer architecture that improves handling of long-range dependencies, accelerates training and inference by parallel computation of the input sequence and capturing Global and Local Information efficiently. We cover all these details in the previous chapter.

II.5 Previous work on machine translation

Moses [48]: Moses, a widely adopted implementation in Statistical Machine Translation (SMT), was developed by the University of Edinburgh. It employs a beam search algorithm to select the best translation candidates. Moses predominantly utilizes a phrase-based translation model, translating short text chunks. **Pretrained models are available for English-to-French translation** [49]. Notably, Moses leverages word alignments generated by **GIZA++** [50].

M2M-100 [51]: M2M-100 is a multilingual encoder-decoder (sequence-to-sequence) model based on the transformers architecture designed for Many-to-Many multilingual translation, introduced by **Facebook** in 2020. This model possesses the capability to perform direct translation across 9,900 language pairs encompassing 100 distinct languages. Its training data comprises an extensive corpus of 7.5 billion sentences, sourced from the **CCMatrix [52]** and **CCAligned [53]** corpuses.

NLLB-200 [54]: Introduced in August 2022, NLLB-200 succeeds M2M-100 as a multilingual model. It covers 200 languages with an extensive training dataset of 18 billion sentence pairs. Notably, alongside CCMatrix and CCAligned, Facebook incorporated NLLB-Seed a professionally translated dataset sourced from Wikipedia text, NLLB-200 is also based on the transformers architecture.

Opus-MT [55]: A set of unidirectional language machine translation models, utilizing vanilla Transformers architectures, were trained on Opus Corpus[56] a curated dataset that stands out for its **predominantly noise-free characteristics like translation errors**, in contrast to the CCMatrix and CCAligned .

II.6 Conclusion

In this chapter, we introduced NLP and its real-life applications. We primarily focused on the foundational principles of machine translation, emphasizing the distinctions between SMT and NMT approaches. Furthermore, we acknowledged the suitability of the transformer model architecture for translation. And we also went through some previous work on machine translation. In the following chapter, we will explore the design of our system in greater detail.

Chapter III

Design

III.1 Introduction

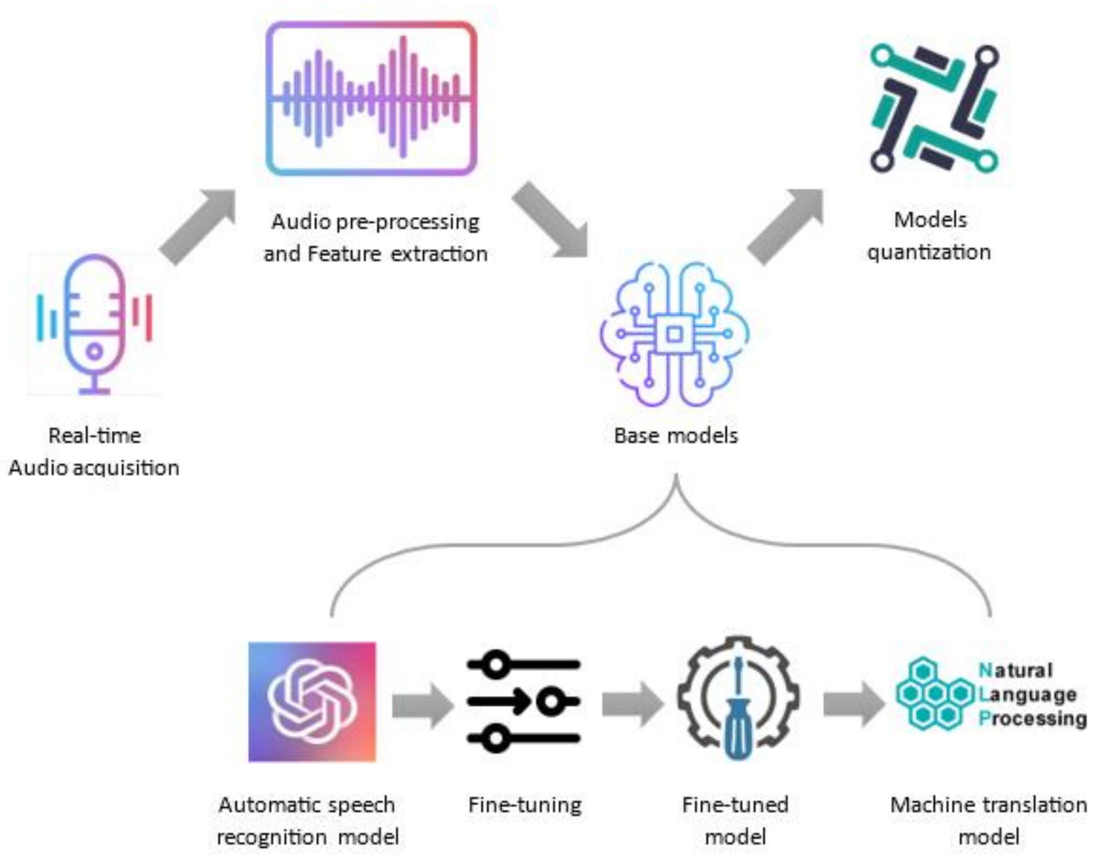
This chapter presents a comprehensive exploration of our system design and the architecture of our models, focusing on the two main components: the transcription model and the translation model. We initiate our examination by addressing the real-time acquisition of audio, followed by a detailed discussion on the preprocessing and feature extraction procedures employed for the acquired audio in real-time. Moving forward, we delve into the core aspects of our "base models" starting with the ASR model and discussing the process of fine-tuning it. Additionally, we delve into the machine translation model and how it compares to other models. Finally, we address quantization, optimizing our models efficiency and enabling their deployment in resource-constrained environments.

III.2 Our Transcription System Pipeline: From Audio Input to Transcribed Text

Our **system comprises various interconnected components** that operate cohesively in a **cascading manner**. This design choice is made due to **limitations in the availability of data for an end-to-end approach**, as well as the increased complexity associated with its maintenance and adaptation in response to emerging technological advancements. Additionally, this approach allows us to **independently optimize each individual component**. It starts by **audio acquisition in real-time** from the microphone device. Then, the captured audio undergoes additional **pre-processing and feature extraction**. The processed audio is then fed into the first model, named **Whisper** [42], developed by OpenAI and released on **September 21, 2022**. Whisper performs **English transcription** on the input audio, generating corresponding written text as its output. The **transcribed text** from the Whisper model serves as the input to the second model, **Opus-MT** [55], developed by the University of Helsinki. Opus-MT is a translation model specifically designed for English to French translation. It takes the transcribed English text and produces the translated text in French as its output.

Both of our **base models** are **pretrained models**. While we do not make further changes to the translation model, we apply the concept of **fine-tuning or transfer learning** to the transcription model. This fine-tuning process allows us to **enhance the performance** of the model by building on its existing knowledge and **adapting it to our specific requirements**.

As a final step, we apply **model quantization** techniques to optimize and adapt the models for real-time inference. This optimization reduces the computational requirements of the models while maintaining their performance. Our transcription system pipeline is illustrated in figure III.1.



Audio pre-processing and Feature extraction
Models quantization
Real-time Audio acquisition
Base models
Automatic speech recognition model
Fine-tuning
Fine-tuned model
Natural Language Processing
Machine translation model

Figure III.1: Transcription System Pipeline: From Audio Input to Transcribed Text.

III.3 Real-time Audio acquisition

In the context of our real-time transcription solution, audio acquisition plays a crucial role. As our system relies on **continuous audio streams** rather than a **30s pre-recorded audio** file whisper expects, doing so it is important to consider the quality of the captured audio. This quality is influenced by both the **microphone** used and the **recording environment**. To fulfill the requirements of our model, it is crucial that the microphone possesses a frequency response capable of capturing frequencies up to 16 kHz. In the design and testing phases of our system, a generic microphone was employed along with standard audio specifications. These specifications are summarized in the table provided below. Nevertheless, our system maintains its flexibility to accommodate the utilization of alternative microphones. To ensure a seamless audio acquisition process, we have **buffering** as an integral component. The buffering mechanism involves storing a small portion of the audio stream in memory before processing it further. By doing so, we facilitate smoother data flow and effectively prevent audio dropouts or interruptions during the transcription process.

Table III.1: Microphone and audio specifications.

Specifications	Values
Microphone frequency response	50Hz - 16KHz
Microphone sensitivity	-55 dB ± 2 dB
Speech Language	English
Chunk size	1024 Sample
Channel	Mono
Self-noise	15dB - 30dB
Sample rate	16000 Hz

III.4 Real-time audio preprocessing and feature extraction

In case the microphone used has a higher frequency response than 16 kHz, we apply down sampling. **Down sampling** is the process of reducing the sample rate of an audio signal, which involves decreasing the number of samples taken per second. During down sampling, consecutive samples are combined or averaged together to generate a reduced number of samples. Whisper faces challenges with **hallucinations**, where predictions may include texts that are not actually spoken in the audio input. Although Whisper was trained to detect speech and noise in audio, it alone is not sufficient. Therefore, we have integrated a voice activity detection (VAD) mechanism to further pre-process the audio. This mechanism effectively eliminates background noise and selectively provides only speech segments as input to our ASR model "whisper". To accomplish this, we utilize **Silero VAD [57]**, an open source pretrained VAD model trained on approximately 13k hours of speech from different domains, including various background noise and quality levels. Silero VAD is a multi-head attention (MHA) based neural network [58] capable of handling audio chunks of different lengths, such as (30,60,100) ms. It supports multiple sampling rates, including 8000 Hz and **16000 Hz**, providing compatibility with a wide range of audio sources and devices. The VAD module in Silero is designed to process audio signals and compute **speech probabilities** for each audio chunk. Each chunk is assigned a probability value, and if the value **exceeds a specified threshold**, it is considered as speech. Conversely, values below the threshold indicate **silence** or **non-speech** segments. After processing the audio and calculating the speech probabilities, Silero VAD applies additional criteria to determine the **boundaries of speech segments**. These criteria include defining the **minimum** and **maximum duration** for speech segments, as well as specifying the **minimum duration of silence** required to separate speech segments. To ensure smoother transitions and avoid abrupt cutoffs, Silero VAD module also incorporates **speech padding**, which adds a small duration of silence before and after each speech segment. The output of Silero VAD is a **list of speech segments**, represented by their **start and end timestamps**. These segments indicate the specific portions of the audio where speech is detected. The primary goal of the VAD module is to accurately identify speech while minimizing false positives or negatives, thereby providing reliable and precise speech detection capabilities. In the following table we define the VAD options we used with Silero.

Table III.2: VAD options we used with Silero.

Option	Value
Speech threshold	40%
Minimum speech duration	100ms
Maximum speech duration	30s
Minimum silence duration	100ms
Window size samples	1024
Speech padding	50ms

After preprocessing the audio and obtaining clear chunks, we utilize the Whisper library for **feature extraction**. Whisper extracts the log-Mel spectrogram, employing specific feature parameters outlined in the table below.

Table III.3: Feature parameters.

Parameters	Values
Feature size	80
Sampling rate	16kHz
Length of the overlapping windows	160
Size of the Fourier transform	400

III.5 Base models

Our approach incorporates two models: "Whisper" for English transcription and "Opus-MT" for English-to-French translation. In this context, we will introduce these models and provide an overview of the fine-tuning process involved in training the transcription model.

III.5.1 Automatic Speech Recognition model

Whisper is a transformer model that has been trained on a vast amount of **weakly supervised data,** which is data that is labeled or annotated with limited or less precise information compared to fully supervised data. Whisper's training dataset consists of a staggering 680,000 hours of audio data, accompanied by their respective transcriptions. Notably, 438,000 hours of this dataset are dedicated specifically to English speech recognition. In contrast to models trained exclusively on pristine, noise-free data adhering to strict standards, Whisper derives its strength from the incorporation of a diverse range of recordings, including noisy ones. This diversity equips Whisper with a broader capacity for generalization and adaptability to various recording environments.

This model is designed to be **multilingual** and **multitask**, capable of transcribing other languages and translating them into English. It also includes functionalities like voice activity detection and language detection. However, we will mainly focus on the **English**

transcription-only models. Whisper offers a **range of model sizes** including tiny, base, small, medium, and large, with all but large that have an English-only transcription model. Figure III.2 illustrates Whisper architecture.

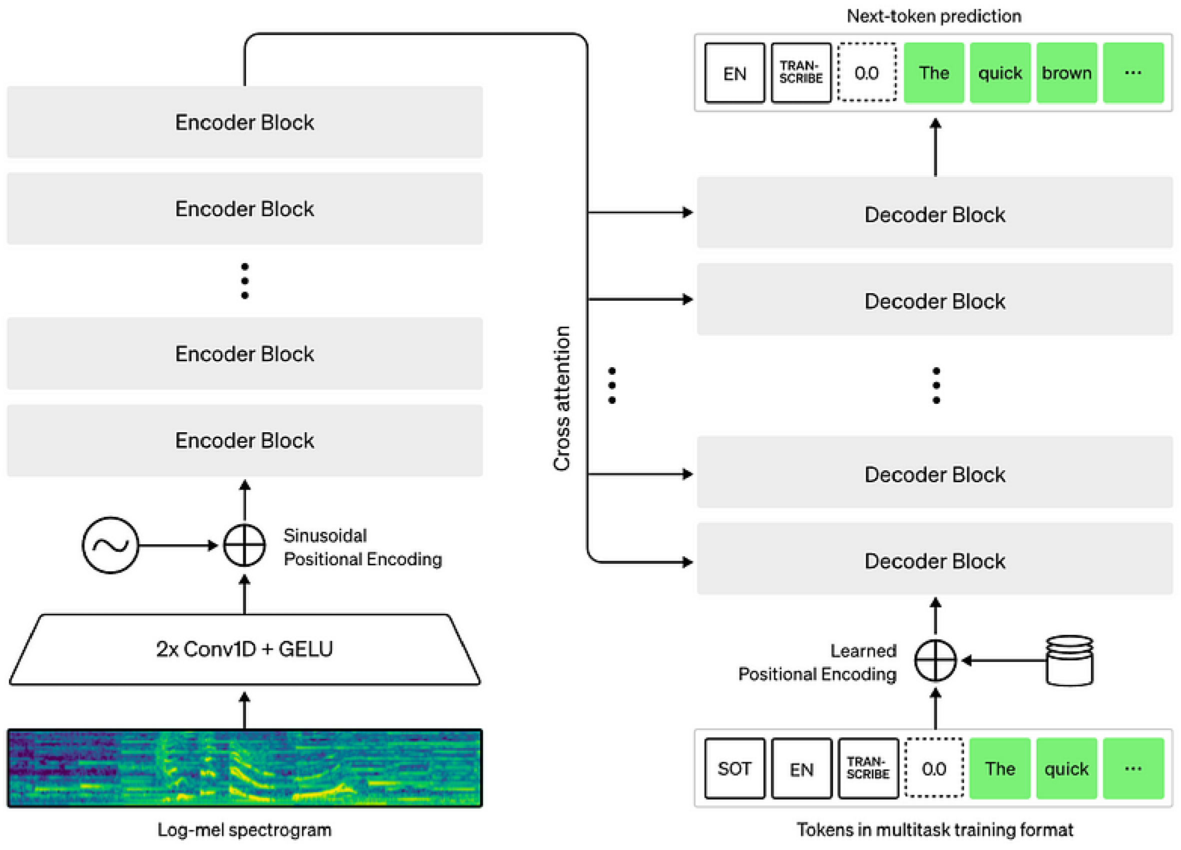


Figure III.2: Overview of the Whisper encoder-decoder architecture.

Whisper is a model specifically designed to process 16 kHz audio inputs. It begins by extracting log-Mel spectrograms using a **window size of 25 milliseconds** and **a stride of 10 milliseconds**, with the same **feature parameters** we mentioned earlier. These spectrograms are then subjected to **feature normalization**, which scales the values between -1 and 1, aiming for an approximately zero mean, to **standardize** and ensure that the input features have **similar statistical properties.**

The encoder continues the processing of the input by employing a small stem, which consists of two 1D convolution layers. These layers are instrumental in capturing local patterns and extracting relevant features from the input sequence. During the convolution operation, a fixed-size window (filter) with a width of 3 slides along the sequence, focusing on neighboring elements. The GELU activation function is applied in both convolution layers, introducing non-linearity to the extracted features. The second 1D convolution layer operates with a stride of two, determining the step size of the sliding window. **Sinusoidal position embeddings** are added to the stem's output, and subsequently, the encoder Transformer blocks are applied. To enhance performance, pre-activation **residual blocks** are utilized, and a final **layer normalization** is employed on the encoder's output.

The decoder component of Whisper employs **learned position embeddings** and shared input-output token representations. Both the encoder and decoder possess the **same width** and **number of transformer blocks**, ensuring **consistency** throughout the model.

Whisper incorporates several **special tokens** that play distinct roles during the transcription process. These tokens include, the beginning of prediction using **<|startoftranscript|>** token, the Whisper VAD sub-system indicating no speech in audio with **<|nospeech|>** token. **<|transcribe|>** token specifying the task of transcribing audio instead of translation, **predicting timestamps** or not by including a **<|notimestamps|>** token. In real-time scenarios, it is not worth using whisper timestamps due to their lack of accuracy. This is because our chunk size, which is smaller than Whisper's default 30-second duration. Our chunk size does not provide **sufficient context** for reliable **timestamp predictions**. Lastly **<|endoftranscript|>** token for end of transcription.

The following table provides a summary of the key aspects and specifications of whisper models, including the **width of the embeddings**, the **number of transformer encoder-decoder layers**, and the total number of parameters in the model. **Parameters** refer to the learnable weights and biases of the transformer model.

Table III.4: Whisper models [42].

Model	Model Layers	Embedding Width	Attention Heads	Model Parameters
Tiny	4	384	6	39 million
Base	6	512	8	74 million
Small	12	768	12	244 million
Medium	24	1024	16	769 million
Large	32	1280	20	1550 million

When it comes to selecting the appropriate model for **our purposes**, we considered our ability to test and fine-tune the model in the next steps. The **Tiny model**, although too small for extensive fine-tuning, proved to be **efficient for real-time transcription** on weaker hardware. Therefore, we utilized it to test our application. On the other hand, the medium model is **relatively complex** for our **specific dataset**, and we lacked the necessary hardware resources to experiment with it adequately. Additionally, the large model did not offer an English-only variant. Hence, we opted for the Base and Small models for fine-tuning as they best **suited our requirements and constraints**.

When decoding audio input in Whisper, the model **predicts probabilities** for **each token**, representing the likelihood of that token being the correct output. Whisper employs **several decoding strategies** to transform these probabilities into output sequences, including **Greedy decoding**, **Beam search**, and **Greedy decoding with Best-of-n sampling and temperature fallback**. Each strategy has its own characteristics, complexity, and impact on our application.

Greedy decoding is a **heuristic** strategy that selects the **token with the highest probability** at each step. It offers **fast decoding** but may lack accuracy and diversity. It focuses on immediate probabilities and **locally optimal choices**, potentially missing out on better solutions. The output may lack diversity and exhibit bias. While known to be **computationally efficient** [59], greedy decoding may not explore alternative sequences effectively.

Beam search is a decoding strategy that expands **multiple candidates** at each step and maintains a list of the **top-k candidates** based on accumulated probabilities. It offers **improved output** sequence accuracy compared to greedy decoding, but it is known to be **slower [59].** Furthermore, beam search employs a **patience** concept, which determines how long the search continues without finding an improved candidate. If no better candidate is found within the specified patience limit, the search may terminate early.

Greedy decoding with best-of-n sampling and temperature fallback is a decoding strategy that incorporates a **temperature parameter** to adjust the diversity of token probabilities, by changing SoftMax function represented in III.1.

$$\boldsymbol{SoftMax(xi)} = \frac{\boldsymbol{exp}\left(\frac{\boldsymbol{xi}}{\boldsymbol{temperature}}\right)}{\boldsymbol{sum}\left(\boldsymbol{exp}\left(\frac{\boldsymbol{xj}}{\boldsymbol{temperature}}\right)\right)} \qquad (\mathrm{III.1})$$

When the **temperature is increased** during decoding, the **output** becomes **more diverse and varied** due to the **introduction of randomness**. On the other hand, **decreasing the temperature** makes the **output more deterministic** and focused. It is important to note that this strategy is only applied when the **temperature value is higher than zero**. The **best-of-n sampling strategy** considers multiple candidates **(n)** at each decoding step, promoting exploration. Additionally, **Temperature fallback** allows for adjusting token probabilities with different temperature values if the initial temperature fails to produce satisfactory results according to the **average log probability (ALP)** over sampled tokens as a criterion for determining if the decoding process has been successful, the **equation for ALP** is as follows III.3. And sum of log probabilities is represented in III.2.

$$\boldsymbol{sum\ of\ log\ \ probabilities} = \boldsymbol{log(p1)} + \ \boldsymbol{log(p2)} + \ldots + \boldsymbol{log(pn)} \qquad (\mathrm{III.2})$$

$$\boldsymbol{average\ lo\,g\ \ probability} = \frac{\boldsymbol{sum\ of\ log\ probabilities}}{\boldsymbol{number\ of\ tokens}} \qquad (\mathrm{III.3})$$

Here, **p1, p2..., pn** represents the predicted probabilities of each token in the sequence.

If the average log probability **falls below a certain threshold**, it is considered a failure and another temperature in the list gets applied. This temperature fallback provides flexibility in finding an optimal balance between randomness and accuracy in the generated output, but it also **increases the computational complexity**, making the greedy decoding process **slower**.

Other decoding strategies exist that the whisper paper did not implement, such as **Stochastic Beam Search (SBS)** which introduces randomness during the sampling process, enhancing output diversity yet potentially trading of quality [60]. In the **Whisper paper**, they initially start with a **beam search of size 5** and **a temperature of 0**. However, based on the result of **ALP** if it is below the **threshold of -1**, they **transition** to **greedy decoding with best-of-n sampling and temperature fallback**. Our **approach**, is to take into consideration and focus on **latency** which will be further explored and explained in the **implementation chapter**.

III.5.2 Fine-tuning

In the context of fine-tuning an ASR model, several important steps are involved illustrated in figure III.3.

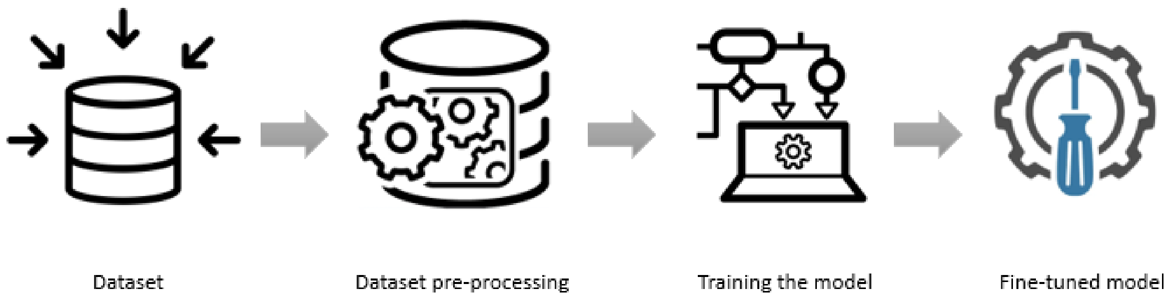


Figure III.3: Fine-tuning process.

The **Dataset** is a thoughtfully curated compilation of audio recordings and their corresponding transcriptions. It encompasses a wide variety of sources, including interviews, meetings, lectures, and conversational interactions, with the goal of capturing diverse speakers, accents, and speech contexts. The audio recordings in the dataset may have different sampling rates, typically 8 kHz, 16 kHz, or 44.1 kHz, depending on the specific data source and audio channel. Alongside the audio files, the dataset includes comprehensive metadata that provides valuable information about the audio recordings. This metadata includes information about audio recording segments corresponding to distinct speakers. Additionally, timestamps indicating the start and end times of each speech segment are available, along with channel IDs, microphone-IDs and other relevant metadata. Several datasets exist, such as Common voice [33], TED-LIUM [61], LibriSpeech [32], and our dataset of choice is **AMI corpus** [62] that encompasses a wide range of meeting scenarios, capturing different microphone qualities and reflecting real-life settings such as different accents.

Data pre-processing is a crucial step in ensuring the dataset is prepared effectively for training. It involves essential procedures such as **down sampling** the audio to an appropriate rate and **segmenting it into speech chunks** shorter than or equal to **30s**. Signal processing techniques are applied to enhance the audio quality such as **removing noise**. In the case of whisper, we also **pad shorter audio clips with noise** until they reach a duration of 30 seconds. Furthermore, **feature extraction** is employed to convert the audio data into log-

Mel spectrograms. On the **transcriptions side**, pre-processing steps include **cleaning and tokenizing the transcriptions**, **normalizing the text**, and **removing punctuation** or **special characters**. The pre-processed dataset is then **split** into a training **set**, which are used for fine-tuning the model, and a test **set**, which are reserved for evaluating the performance of the trained model. We used a ratio of 80% of data for training and 20% for testing.

Fine-tuning is the process of adapting a pre-trained model for a specific task by training it on a **new dataset**. During this process, **Hyperparameters** play a crucial role in determining the model's performance. Hyperparameters are settings that control various aspects of the **training process**, such as **learning rate**, **batch size** which **is** the **number of data examples processed together** during one training step in machine learning, and **regularization techniques**. Some of the hyperparameters used during the training of "Whisper" are presented in the following table. We will utilize these hyperparameters as a starting point for our own fine-tuning experiments, aiming to refine and adapt the model further for our specific speech-to-text task.

Table III.5: Whisper training hyperparameters.

Hyperparameter	**Value**
Steps	1048576
Warmup steps	2048
Batch Size	256
Optimizer	Adam
Weight Decay	0.1
Learning Rate Schedule	Linear Decay
Learning rate **(Small Model)**	5 × 10−4
Learning rate **(Base Model)**	1 × 10−3

For fine-tuning, we also consider regularization techniques such as **Weight Decay [63] and Dropout**. Weight Decay discourages large weight values in the model by **adding** a penalty term to the **cross-entropy loss function**. It helps control the model's complexity and prevents overfitting. Dropout randomly deactivates some neurons during training, preventing overreliance on specific neurons and promoting robust learning. **Dropout is usually a percentage of neurons to deactivate.**

For Weight decay the equation is as follows III.4.

$$\textit{Weight Decay} = \frac{\lambda}{2} * ||W||^2 \qquad (III.4)$$

W represents the weights of the model, **||W||²** represents the sum of the squares of the weights in the model. And **λ** is the regularization coefficient.

III.5.3 Machine translation model

III.5.3.1 Opus machine translation

Our system utilizes the Opus-MT (English to French) model for machine translation. This model is built on a vanilla transformer architecture, consisting of **6 layers** in both the encoder and decoder. Each layer is equipped with **8 attention heads**, and the **embedding width is set to 512**. The model has a total of **75 million parameters,** with source and target vocabulary of 32000. The Model was trained on **Opus corpus [56]**, which is a **versatile collection of translated texts** that encompasses a **wide range of domains**. Making it suitable for **general use**, unlike other vanilla transformer model that only use WMT (Workshop on Statistical Machine Translation) dataset. Opus captures the nuances, complexities, and vocabulary found in different spoken styles and topics. It provides the necessary capacity to handle various domains effectively, making it an excellent choice for English to French translation tasks. The Opus dataset underwent preprocessing using **Sentence Piece** [64] tokenizer, known for its precise tokenization and **computational efficiency**. Unlike traditional **Byte Pair Encoding (BPE)** methods with $\boldsymbol{O}(\boldsymbol{N}^2)$ complexity **used in vanilla transformers**, Sentence Piece employs an optimized algorithm with an efficient $\boldsymbol{O}(\boldsymbol{N}\,\boldsymbol{log}(\boldsymbol{N}))$ approach. This algorithm substantially reduces the computational cost of tokenization and provides better handling of **rare and out-of-vocabulary words**, allowing for improved capture of their **morphology and semantic meaning**.

Opus-MT utilizes **guided alignment for attention** [65] during training, leveraging **word alignments** generated with the efmaral [66] algorithm. This technique incorporates alignment information as a form of supervision, guiding the model to attend to relevant parts of the **source sentence** while generating the **target sentence** by the attention mechanism. Unlike the traditional method that relies solely on self-attention. The inclusion of guided alignment **improves generalization** in Opus MT. By incorporating alignment information, the model learns **alignment patterns** that can be applied to **unseen data**, which in our case is crucial since our fine-tuned ASR model now can generate some in domain words that are out of the norm. As for decoding strategies Opus-MT can decode with greedy decoding and beam search.

III.5.3.2 Opus-MT compared to other models

For our approach, we require a robust translation model that **excels in one direction translation task** as we employ a **cascade approach to build our system**, where **we intend to add other one-directional models to our system** as the needs of the users or company grows **keeping the system always efficient hardware wise**. However, **we aim to strike a balance between model inference speed and translation accuracy**, without compromising the latter significantly. In assessing the accuracy of these models, we refer to using the **Bilingual Evaluation Understudy (BELU)** [67] metric, BELU is a metric used to assess the quality of machine-generated translations by comparing them to reference or human-generated translations. It measures the similarity between the machine-generated text and the reference text in terms of n-grams (sequences of words). BLEU is calculated by counting matching n-grams in the machine-generated text and the reference text, and then computing a precision score based on these counts. The precision score is typically multiplied by a brevity penalty to account for differences in length between the machine-generated and reference text. The equation for BELU is represented in III.5.

$$\boldsymbol{BELU = BP * exp\left(\sum_{n=1}^{1} \frac{1}{n} P_n\right)} \tag{III.5}$$

Where BP stands for Brevity Penalty, which penalizes the score when the Machine Translation is too short compared to the Reference (Correct) translations and **n** ∈ [1,4]
Pn is the **n-gram** modified precision score. The **mathematical expression for Brevity Penalty** is given as follows in III.6. And equation for **Pn** in III.7.

$$BP = min\left(1, \frac{Machine\ Translation\ Output\ Length}{Maximum\ Reference\ Output\ Length}\right) \quad (III.6)$$

$$Pn = \frac{\sum n - grams\ count\ in\ Machine\ Translated\ Text}{\sum n - grams\ count\ in\ Reference\ Text} \quad (III.7)$$

In the following comparative analysis with Opus-MT, Opus-MT is compared to four models, comprising two variants from M2M-100 and two from NLLB-200. Notably, **we excluded Moses pretrained models** from our comparison, as they have lagged behind the current state of the art, with the academic community **predominantly shifting towards transformer-based models**, especially that SMT models are **not tailored towards real-time usage**. The evaluation relies on the BLEU metric across **17 machine translation evaluation standard datasets with different versions** including TICO-19[68], Tatoeba [69], Flores 101[70], WMT news translation [71], Multi30k [72]. It is important to note that even though **M2M-100 and NLLB-200 are multilingual models**, these models are expected to exhibit **robust pattern recognition learned from other languages reflecting in higher accuracy in high resource languages like English and French**. The following tables present in-depth details about the models, and the BLEU results where some where tested by us and some we obtained from the Opus dashboard for lack of performant hardware.

Table III.6: Opus-MT comparison models details.

Model	Opus-MT(1)	NLLB-200-Distilled(2)	NLLB-200(3)	M2M-100(4)	M2M-100(5)
Attention heads	8	16	16	16	16
Embedding width	512	1024	1024	1024	1024
Model layers	6	12	24	24	48
Model parameters	75 Million	600 Million	1.3 Billion	418 Million	1.2 Billion

The table above shows two NLLB-200 model variants**: one is distilled with 600 million parameters**, while the other is not distilled and has **1.3 billion parameters**. **Distillation**

involves a smaller model learning from a larger model's knowledge. In contrast, both M2M-100 models are not distilled but differ in complexity, and both were trained on the same dataset

Table III.7: Opus-MT compared to M2M-100 and NLLB-200 variants using BELU metric.

Dataset	Model 1	Model 2	Model 3	Model 4	Model 5
Flores 101	47.9	45.1	47.4	39.0	44.7
Multi30k 2016 Flickr	48.8	47.8	50.5	33.3	42.6
Multi30k 2017 Flickr	49.3	45.9	49.6	34.2	41.9
Multi30k 2017 Mscoco	51.6	48.7	50.2	36.2	46.8
Multi30k 2018 Flickr	41.4	39.2	42.7	29.9	37.4
WMT News Discuss 2015	38.4	36.2	37.9	31.7	35.6
WMT News SC 2009	29.9	30.2	31.6	26.1	28.4
WMT News 2009	29.4	29.4	30.5	25.4	27.7
WMT News 2010	31.7	32.3	33.6	28.4	31.7
WMT News 2011	34.2	34.1	35.1	29.9	32.2
WMT News 2012	31.7	32.0	33.3	28.2	30.4
WMT News 2013	33.1	33.7	35.0	28.4	31.5
WMT News 2014	39.8	39.4	41.1	33.0	37.4
Tatoeba 2020-07-28	50.8	47.8	49.6	37.9	43.7
Tatoeba 2021-03-30	51.4	48.4	50.2	38.3	44.2
Tatoeba 2021-08-07	51.8	48.6	50.6	38.7	44.6
Tico 19	37.6	38.1	40.1	34.2	39.1
Average	**41.1**	**39.8**	**41.7**	**32.5**	**37.6**

As evident from the table above, **Opus-MT outperforms Models 2, 4, and 5** by a notable margin in terms of Average BLEU scores, with **difference of Averages of 1.3, 8.6, and 3.5 respectively**. While **Opus-MT only falls slightly short compared to Model 3** with a difference of just **-0.6** although According to NLLB-200 paper there is a model with **54.5 Billion parameters** that is better than the 1.3 billion variant. Regardless, this is acceptable in exchange for a faster and less complex model as **Opus-MT is only 75 Million parameters** model especially that it’s observable that the **larger models didn’t gain much accuracy from**

recognizing patterns in other languages probably due to the noisy training data CCMatrix and CCAligned, Overall, the Real-time performance of our system takes precedence in this context, and this trade-off is deemed reasonable.

III.6 Models quantization

To achieve **faster inference** with **minimal latency** in real-time applications and reduce the memory footprint and computational requirements of models, quantization [73] is utilized. The process of quantization involves **decreasing the precision of numerical values** in a model such as the **weights**, while ensuring that the **performance impact remains minimal**. This is achieved by **converting high-precision** floating-point numbers, commonly used in deep learning models, into **lower-precision** fixed-point or integer representations.

There are various types of quantization, and in our case, we employed **Integer8 (INT8)** quantization. INT8 quantization represents values using 8-bit signed integers, allowing for a range of [-128, 127]. There are several reasons why INT8 quantization was preferred:

Memory Efficiency: INT8 quantization offers the highest memory efficiency among the available formats such as **int16** and **float16**. By utilizing only 8 bits to represent each value, it significantly reduces the memory footprint compared to higher-precision formats.

Computation Efficiency: INT8 operations are generally more efficient due to hardware support and optimized instructions for 8-bit integer arithmetic. Modern hardware platforms, including both CPUs and GPUs, often provide dedicated acceleration for INT8 operations.

The process of **INT8 quantization** involves several **steps** to convert high-precision floating-point values into 8-bit signed integers (INT8). **Firstly**, **scaling** is performed to map the original range of values to the range of -128 to 127, which is the valid range for INT8 representation. This scaling factor is computed based on the **maximum absolute value** among all the numbers in the model. **Next**, the **scaled values** are **rounded** to the nearest whole number to convert them into discrete integers compatible with the INT8 format. Rounding ensures that the values are **approximated** to the **closest integer values**. **Lastly**, **clipping** is applied to ensure that the rounded values fall within the acceptable range of -128 to 127. If any rounded value exceeds this range, it is adjusted or "clipped" to the nearest boundary value (-128 or 127) to maintain the integrity of the INT8 format. By following these steps of scaling, rounding, and clipping, the original high-precision floating-point values are transformed into the INT8 format. To preserve the performance of our transformer model, we applied quantization using the following formulas [74] specifically to the weights of the embedding and linear layers. These layers involve **matrix multiplication operations**, which are **computationally demanding** and **time-consuming**. By quantizing only the weights, we aimed to optimize the model's efficiency without compromising its overall performance.

$$\boldsymbol{scale}[\boldsymbol{i}] = \frac{\mathbf{127}}{\boldsymbol{max}\big(\boldsymbol{abs}(\boldsymbol{W}[\boldsymbol{i},:])\big)} \qquad (\text{III}.8)$$

This equation III.8. Calculates the scaling factor for each row **i** of the weight matrix **W**

$$\boldsymbol{WQ}[\boldsymbol{i},\boldsymbol{j}] = \boldsymbol{round}(\boldsymbol{scale}[\boldsymbol{i}] * \boldsymbol{W}[\boldsymbol{i},\boldsymbol{j}]) \qquad (\text{III}.9)$$

This equation III.9. Rounds each element **W [i, j]** in the weight matrix **W** after scaling.

The table below illustrates the impact of quantization on our models in terms of their disk storage size. Later on, we will explore how this also influences their Inference time.

Table III.8: Variation in Models Size Following Quantization

Model	Size Before	Size After
Opus-MT	301 Megabyte	76 Megabyte
Whisper Tiny	151 Megabyte	40 Megabyte
Whisper Base	291 Megabyte	75 Megabyte
Whisper Small	967 Megabyte	242 Megabyte

III.7 Conclusion

Within this chapter, we presented a comprehensive overview of our system design and the architecture of our models. We discussed the various choices made to ensure the development of robust models capable of effectively handling the designated tasks. In the upcoming chapter, we will proceed with the implementation of the system and conduct a series of experiments to evaluate its performance.

Chapter IV

Implementation

IV.1 Introduction

In this chapter we focus on the implementation and results of our design choices where we conduct a series of experiments, assessing the impact of the preprocessing step and quantization technique. We also evaluate real-time latency and conduct fine-tuning on whisper and later on compare the adapted model to previous work. Furthermore, we showcase the graphical user interface (GUI) of the application and provide an overview of the tools and libraries employed in the process. Throughout this chapter, we present detailed methodologies and insightful results, providing a comprehensive understanding of how our project has been implemented.

IV.2 Tools and libraries

During the development of our project, we leveraged the power of **Python** as our primary programming language along with a **variety of tools and libraries**. Here is an overview of the key tools and libraries that played a crucial role in our project's advancement:

IV.2.1 Tools:

- **VSCode**: VSCode is a lightweight and highly customizable code editor [75].
- **Anaconda**: To simplify the installation and management of Python and related packages for data analysis and machine learning, we relied on **Anaconda** [76].
- **Jupyter Notebook**: We leveraged the interactive web-based environment of **Jupyter Notebook** to create and share documents that combine live code, visualizations, and explanatory text [77].
- **TensorBoard**: For monitoring training progress and analyzing performance metrics we used **TensorBoard**, a web-based tool that offers comprehensive visualization capabilities [78].
- **AnyDesk**: Due to the geographical distance between us and the company, we utilized AnyDesk, a remote desktop software, to access and control one of their systems remotely [79].

- **PyCharm**: PyCharm is an Integrated Development Environment (IDE) tailored for Python, with powerful **debugging** tools, **profiling** tools, and an integrated version control [80].

IV.2.2 Libraries:

- **CTranslate2**: CTranslate2 is a C++ and Python library for efficient inference with Transformer models that adapts to quantized models [81], also allows for padding removal, caching mechanism, Parallel and asynchronous execution among other things.
- **FasterWhisper**: faster-whisper is a reimplementation of OpenAI's Whisper model using CTranslate2 [82].
- **Hugging Face Transformers**: Transformers provides APIs and tools to easily download and train state-of-the-art pretrained models [83].
- **JiWER**: JiWER is a simple and fast python package to evaluate an automatic speech recognition system, it supports %WER measure [84].
- **PyAudio**: PyAudio provides Python bindings for PortAudio v19, the **cross-platform** audio I/O library [85].
- **NumPy**: NumPy adds support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays [86].
- **Pandas**: Pandas is a software library written for the Python programming language for data manipulation and analysis [87].
- **Flet**: Flet is a python flutter wrapper framework that allows building interactive **cross-platform** multi-user web, desktop and mobile applications [88].
- **SentencePiece**: SentencePiece is an unsupervised text tokenizer and detokenizer mainly for Neural Network-based text generation systems where the vocabulary size is predetermined prior to the neural model training [89].
- **Matplotlib**: a software library written for the Python programming language for data manipulation and analysis [90].

- **Onxx Runtime:** ONXX Runtime: cross-platform, high performance ML inferencing and training accelerator**,** used to load the Silero model [91].
- **Librosa:** A python package for music and audio analysis [92].

Other built-in Python modules, such as **Wave**, **Threading**, **Queue**, **Time**, **Timeit**, and **Datetime**, provide functionalities for working with audio files, managing threads, implementing queues, handling time-related operations such as sleep function, measuring code execution time, and manipulating dates and times.

IV.3 Experiments

IV.3.1 Fine-tuning experiment:

During our fine-tuning experiments, we conducted the training on a computer system with specific specifications as shown in the following table. As for the AMI dataset, it was initially archived with a size of 9.24 GB. After extraction, the dataset expanded to approximately 20.8 GB. However, after the preprocessing step, the dataset size increased significantly to approximately **120 GB**. With **108502 audio chunks** for **training**, and **25741 audio chunks** for **testing, all with their corresponding transcriptions**. The AMI corpus transcriptions have mostly uppercase text and missing punctuations, but the transformer architecture allows the model to learn continuously and leverage its previous knowledge regardless of this issue. The guided alignment feature in translation models helps later restore proper casing and punctuations, ensuring accurate translations.

Table IV.1: Training computer specification.

Specification	Value
Operating system	Windows 10
CPU	AMD Ryzen 5 3600 6-Core 4.2 GHz
RAM	32GB
DISK	ADATA 1 Terabyte SSD
GPU	NVIDIA GeForce RTX 3060 12 GB

These considerations highlight the **resource requirements** and data storage considerations involved in working with the AMI dataset for fine-tuning. Regarding **freezing layers**, we decided **not to freeze** any layers because the model is based on transformers. Typically, one would choose to freeze either the encoder or decoder, but we opted not to do so in order to allow the model to **learn both new audio patterns and new vocabulary simultaneously**.

First, we tested the **initial %WER** of **“small”** and **“base”** models table III.4 on our dataset. Where we used a **beam search** of **size 5** and when ALP gets below the threshold of (-1) we use **greedy decoding** with **temperature fallback** while also having **best of 5 sampling**, the list of **temperatures** used is **(0.2, 0.4, 0.6, 0.8, and 1.0)**. We got these results for the **“small”** model where **initial %WER is 17.5%**, and for the **“base”** model **initial %WER is 20.5%**. To fine-tune the models, we **adjusted the learning rate**. Initially, the **learning rate** was set to the **paper-training learning rate** of each model **divided by a factor of 40**, as **recommended by the paper authors**. We also utilized the Adam optimizer, following the same approach outlined in the paper. Additionally, we ensured diversity in the training samples by **shuffling the data** at the beginning of each epoch where an **"epoch"** refers to a **single complete pass through** the **entire training dataset** during the training of a neural network.

We first started by fine-tuning the **“small” model** but given the **hardware limitations** and the size of the "small" model, we employed a **multi-step approach** to overcome these challenges. Initially, the model underwent fine-tuning, followed by **subsequent iterations** where we **gradually increased the batch size**, enabling better utilization of available resources. The "small" model was fine-tuned with a **learning rate of 1e-05**, spanning a total of **6 epochs**. In subsequent figures, **we used steps as a means of simplifying the representation of metric changes**, where **21000 steps corresponds to 6 epochs**.

Now let's analyze the **training loss** of our model, which serves as an indicator of how well the model is learning during the training process, by measuring dissimilarity between the predicted outputs of the model and the actual output.

Overall, the **training loss showed a decreasing trend**, with occasional rapid drops that can be attributed to optimizer-induced decreases in the learning rate and the corresponding increase in batch size. Details and trends of the training loss can be observed in Figure IV.1, where we logged **training loss every 25 steps.**

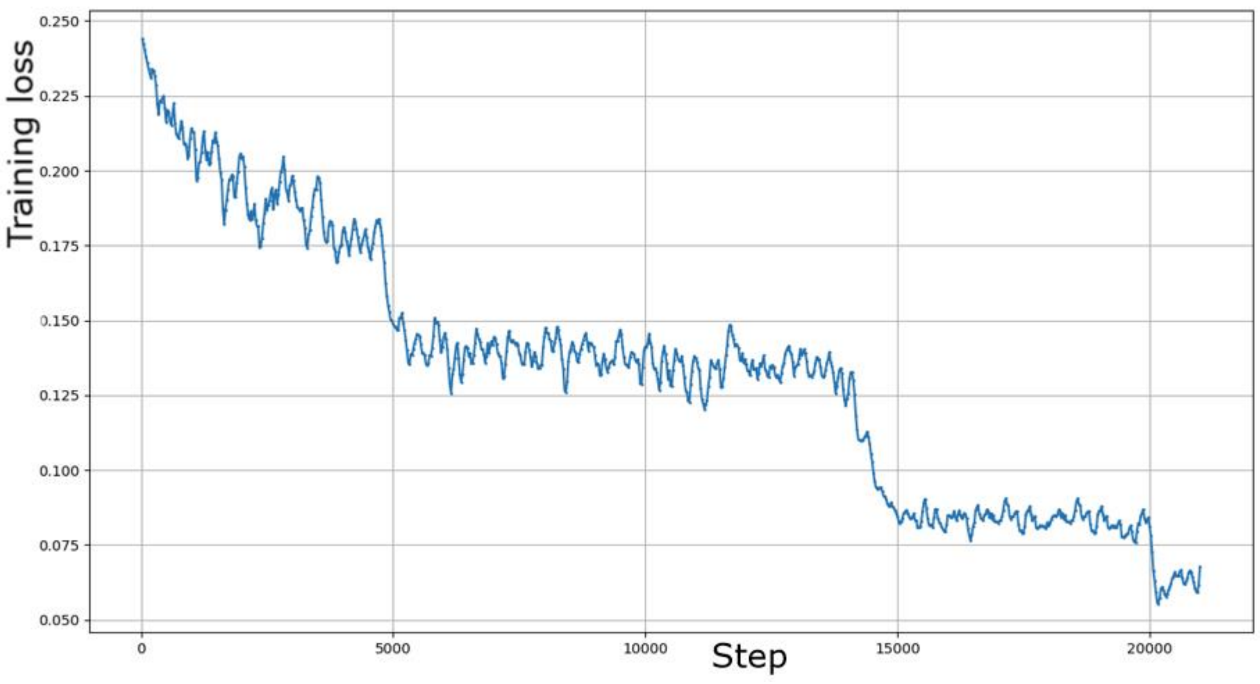


Figure IV.1: "Small" model training loss over 6 epochs.

Next, we examine the **validation loss** of our model, which provides insights into the **model's performance on unseen data** during the training process. During the **initial stages**, the validation loss **decreased rapidly**, indicating that the model was learning and making improvements. However, after a certain number of steps, we observed an **increase in the validation loss**, reaching a **local High** suggesting it encountered certain difficulties. Following that, the validation loss started to decrease again, eventually **reaching its lowest point** suggesting that the model overcome these challenges. Results are illustrated in Figure IV.2, where we logged **validation loss every 1000 step.**

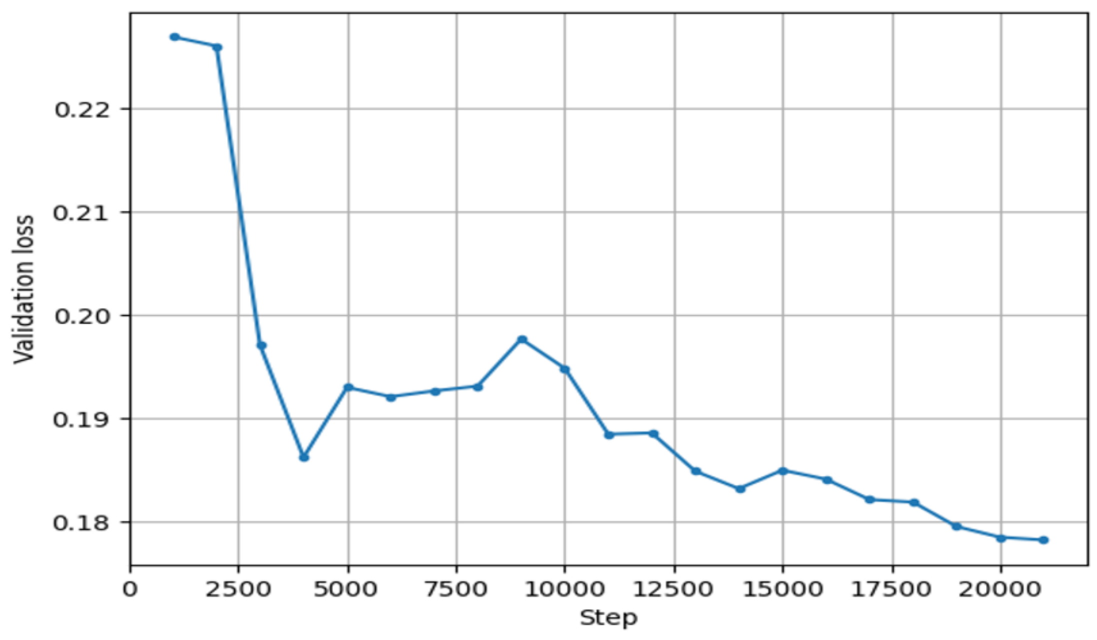


Figure IV.2: "Small" model validation loss over 6 epochs.

Next, let's analyze the %WER of our model throughout the training process. **Initially**, the %WER was observed to be relatively high at **28.12%**. This can be attributed to the **random initialization of the model's weights**, which temporarily affected its performance. However, as the training progressed, the %WER quickly decreased and continued to decline gradually. Eventually, the %WER reached a **final value of 11.42%,** having a **total improvement of 6.08%.** Results illustrated in Figure IV.3, where we logged **%WER every 1000 steps.**

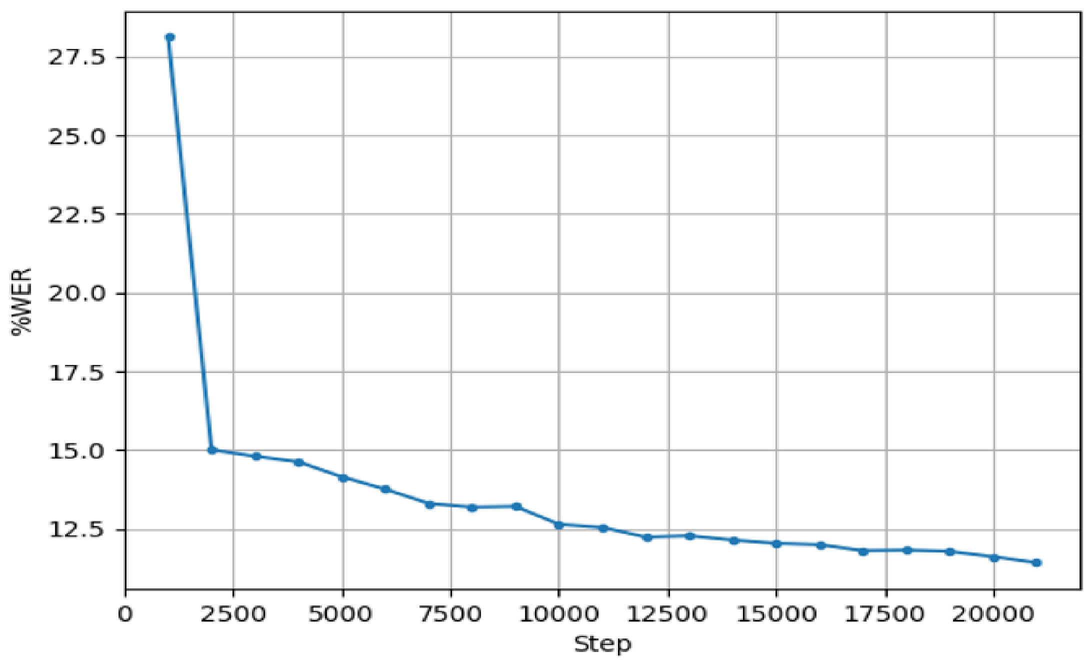


Figure IV.3: "Small" model %WER over 6 epochs.

In the next step, we will explore the process of fine-tuning the **"base"** model. This allows us to observe how a **smaller and less complex model**, with fewer parameters, responds to our dataset. Additionally, we aim to assess the **impact of weight decay and dropout regularization**, along with a **lower batch size**. The use of a lower batch size acts as a form of regularization, as it exposes the model to fewer samples. This discourages overreliance on specific subsets of the data and promotes better generalization to unseen examples. During this fine-tuning process, we will be working with **two instances of the "base" model**. The following table provides a **summary of the hyperparameters** employed for this test for each model. Keep in mind 2 epochs are equal to 14000 steps.

Table IV.2: base models fine tuning hyperparameters.

Model instances	Weight decay	Batch size	Learning rate	Dropout	Epoch
Base 1	none	32	2.5e-0.5	none	2
Base 2	0.1	8	2.5e-0.5	0.1	2

During the training process, both models exhibited similar trends in terms of the **training loss**. However, **Base 1** experienced a **slightly higher peak at the beginning** compared to the other model. Nevertheless, as the training progressed, both models converged to similar results, indicating that the implemented **regularization techniques had little effect**. Results are illustrated in Figure IV.4, where we logged **training loss every 25 steps.**

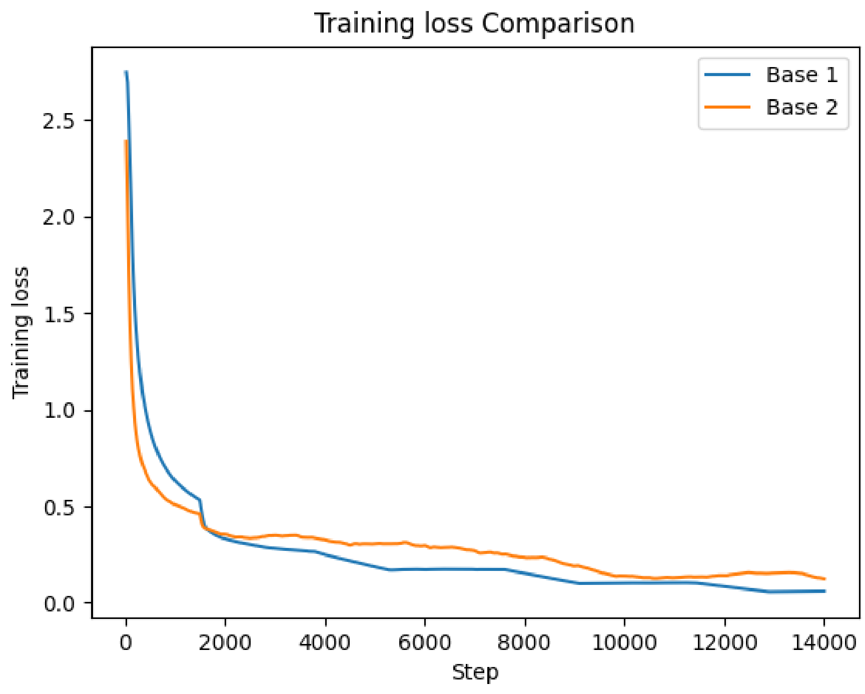


Figure IV.4: Training loss of base models.

During the **validation process**, **Base 2** exhibited a **significantly higher validation loss** compared to Base 1 initially, indicating that the implemented **regularization techniques had a strong impact on Base 2's performance**. In contrast, Base 1 did not experience such a high validation loss in the early stages. However, as the training progressed, Base 2 showed remarkable improvement and eventually achieved a lower validation loss than Base 1. Interestingly, Base 1's validation loss increased in the later stages. These findings suggest that the **regularization techniques were beneficial for Base 2** in terms of generalizing to unseen data. Results are illustrated in Figure IV.5, where **we logged validation loss every 2000 steps.**

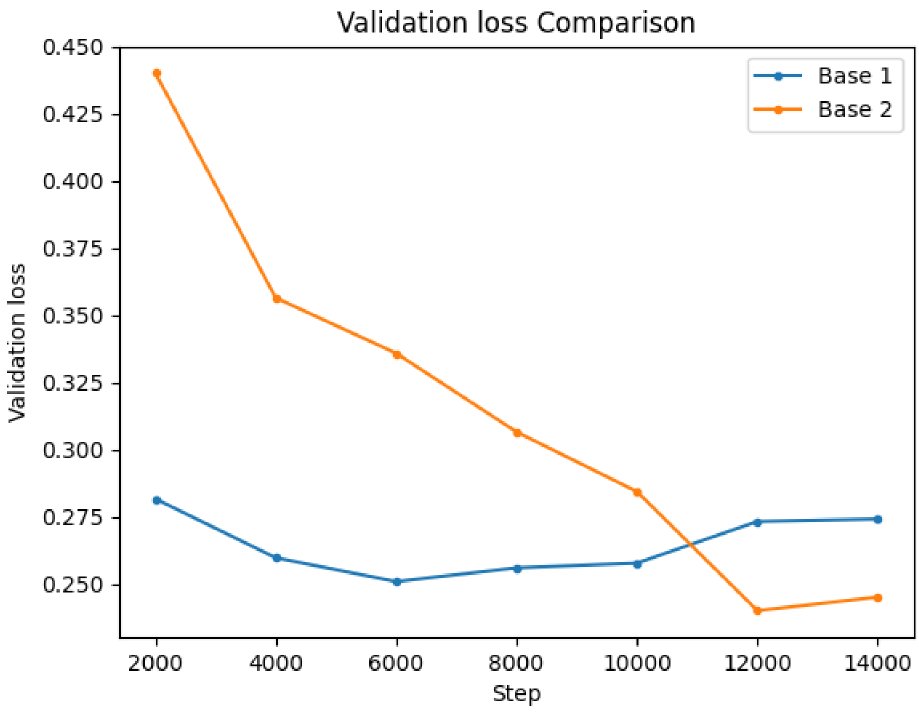


Figure IV.5: Validation loss of base models

Lastly, **we analyzed the change in %WER for the "base" models**. Like the small model, both base models **initially experienced an increase in %WER**. However, Base 2 exhibited a larger increase, indicating that the applied regularization techniques had a more noticeable effect on its performance. Towards the end of the training, Base 1 managed to reduce its %WER but experienced a slight increase again, suggesting potential overfitting to the training data. **Base 1 reached a %WER as low as 14.66% but then increased to a final %WER of 15.79%** having a **total improvement of 4.71%**. On the other hand, Base 2 continued to decrease its %WER, indicating better generalization due to the effects of regularization. **Base 2 achieved a final %WER of 14.12%** having a **total improvement of 6.38%**, showcasing its improved performance on unseen data. Results are illustrated in Figure IV.6, where **we logged %WER every 2000 steps.**

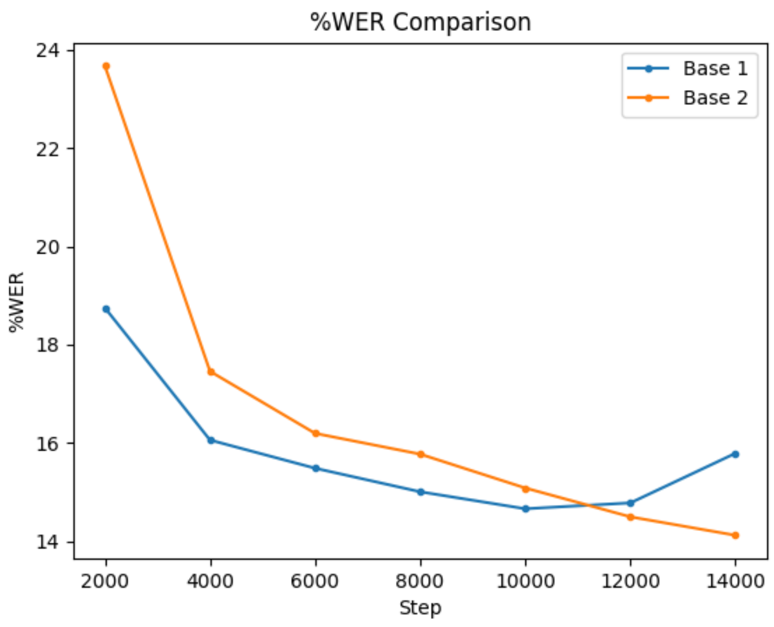


Figure IV.6: %WER of the base models

IV.3.2 Real-time performance experiments:

In our implementation, we **prioritized real-time performance** as a key consideration. To assess the application's real-time capabilities, we conducted a **series of tests** on a computer system equipped with a **dual-core Intel i7-7500U** processor running at a base frequency of 2.7GHz and capable of boosting up to 3.5GHz. The system also had 16GB of RAM and a 128GB SSD, and it was running a Linux operating system. In the context of our chosen models, the primary consideration is the performance of the CPU and GPU. It's worth noting that this particular machine lacks a GPU.

In the **First test**, we evaluated the inference time of the "**tiny" Whisper model** and the **Opus-MT model using the default implementation** compared to the **CTranslate2 implementation** of both models using **int8 quantization**. For Whisper, we used beam search of size 5 and temperature fallback. For Opus MT, we only employed beam search of size 5. We utilized a 3-minute audio sample from the AMI dataset for transcription purposes. The resulting transcription contained 383 words. Additionally, we used a 223-word example from the Opus dataset for translation. The outcomes of these tests are presented in the following table, where we can easily notice that CTranslate2 implementation in both models is significantly faster for just a fraction more of Errors, **Error rate** is the fraction of errors to the total number of words.

Table IV.3: Inference time results for different implementations.

Implementation	**Inference time**	Error Rate
Default Whisper implementation	43.14s	0.01827%
CTranslate2 Whisper with **quantization**	8.45s	0.02088%
Default Opus-MT implementation	4.83s	0.00448%
CTranslate2 Opus-MT with **quantization**	2.7s	0.00896%

In the **Second test**, we tried to measure the **impact of the preprocessing step** which consists of filtering noise with **Silero**, we used the **same strategy and audio sample**. As a result, we got an inference time of **7.64s and got 4 wrong out of 383 making an error rate of 0.01044%** which means that Silero has **improved both inference time and error rate**. Silero loading and inference itself did not add any measurable inference time because the model is only 1mb in size, which is too low to cause any impact.

For the **Third test** we try to measure the worst-case inference time of our system to see if the worst case is going to be noticeable by the user and give him a bad experience, for testing we used the **"tiny" whisper model**.

To conduct the evaluation, we acquired **three audio samples**, each spanning 30 seconds in length, from various sources on the internet. These **samples were manually transcribed and translated**. **Here are the details of the samples we utilized:**

- Sample 1[93] is of an American president speech with light background audience clapping, this sample contains a total of 67 words.
- Sample 2[94] is of a man talking while another person in the background is talking and moving chairs around causing noise. We selected this sample to see if it affects the final transcription quality. This sample contains a total of 65 words.
- Sample 3[95] is a women's motivational speech with background music. This sample contains a total of 44 words.

Our testing approach involved segmenting each audio sample into **0.5-second incremental chunks** and later processing them. This aligns with our app's functionality, as we **wait for a 0.5-second** before sending the first audio chunk to whisper. **For instance,** we started by **transcribing only the first 0.5 seconds** of each audio sample and measured the inference time. Subsequently, we **transcribed 1 second**, **then 1.5 seconds**, and so on, until we **covered the entire 30-second duration**. This process resulted in **60 measurements of inference time** for a single 30-second audio. Considering that we conducted this evaluation with three different samples, we obtained a total of **180 measurements of inference time**,

In addition, we accounted for an estimated time of **10ms** for each inference, representing the duration it takes for audio chunks to **travel from the microphone to the buffer queue**. This additional time was considered as part of the overall latency for each inference.

Additionally, for each audio sample, we computed various latency metrics. This included calculating the **average latency over the entire 30 seconds** of audio, as well as the **average latency over the initial 20 seconds**. Furthermore, we determined the maximum latency observed throughout the entire 30 seconds and the maximum latency limited to the first 20 seconds. By examining these metrics, we aimed to identify any instances of **particularly high latency**. Moreover, we carefully inspected the results for any errors, such as incorrect transcriptions or translations, to ensure the accuracy and reliability of the system.

Results for the three audio samples with greedy decoding for both Opus-MT and Whisper while also using Silero pre-processing are illustrated in Figure IV.7, and other measurements in table below.

Table IV.4: latency metrics over audio samples with greedy decoding.

Measurement	Sample 1	Sample 2	Sample 3
Average latency over 30s	0.78s	0.79s	0.76s
Average latency on first 20s	0.64s	0.68s	0.63s
Maximum latency over 30s	1.19s	1.21s	1.13s
Maximum latency on first 20s	0.92s	0.88s	0.94s

In **Sample 1**, there were no mistakes or errors in the transcriptions. The model performed accurately without any errors. In **Sample 2**, we encountered one mistranscribed word and **one hallucinated word** that the speaker did not actually say. These were identified as errors in the transcription. In **Sample 3**, we noticed **one hallucinated word**, one miss-translated word, and one mistranscribed word.

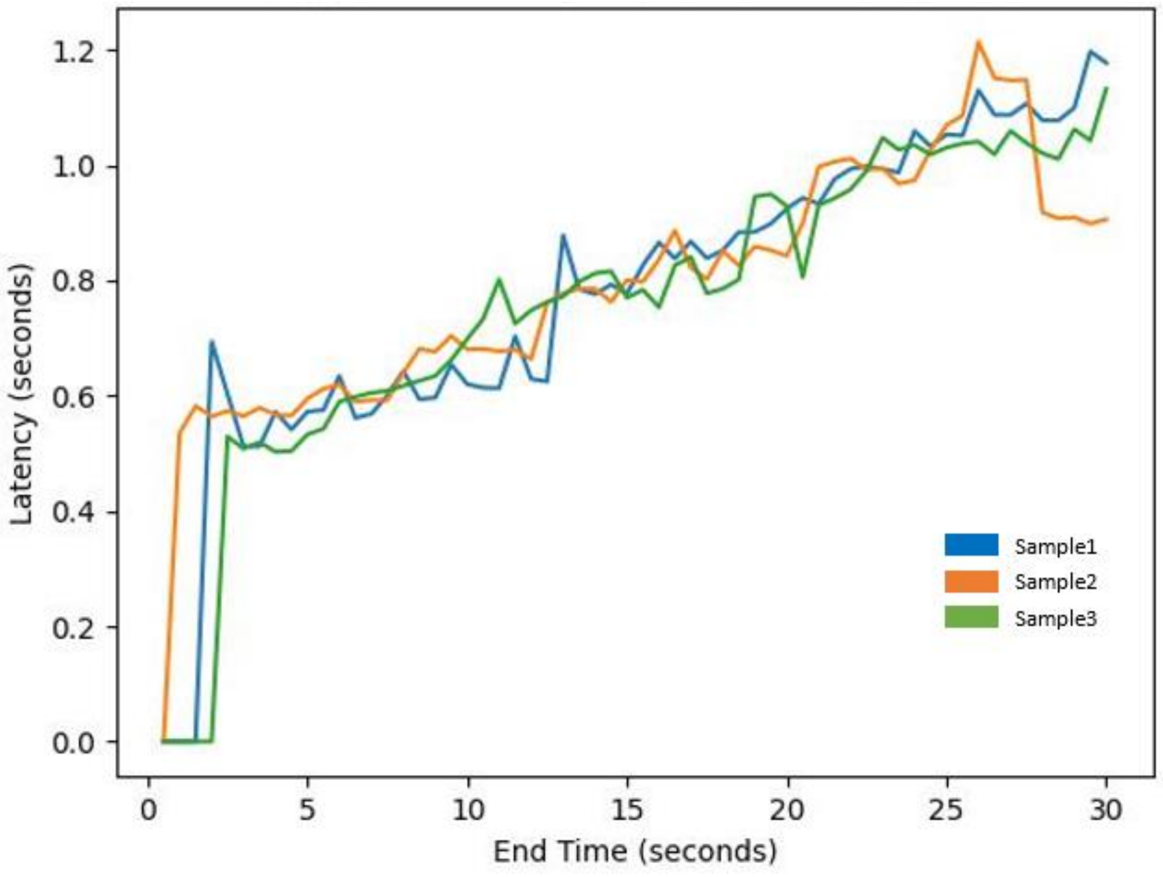


Figure IV.7: Accumulated latencies over 30s with greedy decoding

The outcomes for the three audio samples were obtained using beam search with a size of 5 for both Opus-MT and Whisper. We also applied a temperature fallback strategy for Whisper. Additionally, the Silero pre-processing method was employed. Results are illustrated in Figure IV.8.

Table IV.5: latency metrics over audio samples with beam search and temperature fallback.

Measurement	Sample 1	Sample 2	Sample 3
Average latency over 30s	0.99s	1.03s	0.99s
Average latency on first 20s	0.75s	0.78s	0.81s
Maximum latency over 30s	1.74s	1.79s	1.55s
Maximum latency on first 20s	1.17s	1.27s	1.21s

In **Sample 1** and **Sample 3**, there were no mistakes or errors in the transcriptions. The models performed accurately without any errors. In **Sample 2**, we noticed **two hallucinated extra words** and one mistranscribed word. These errors occurred in the transcription process.

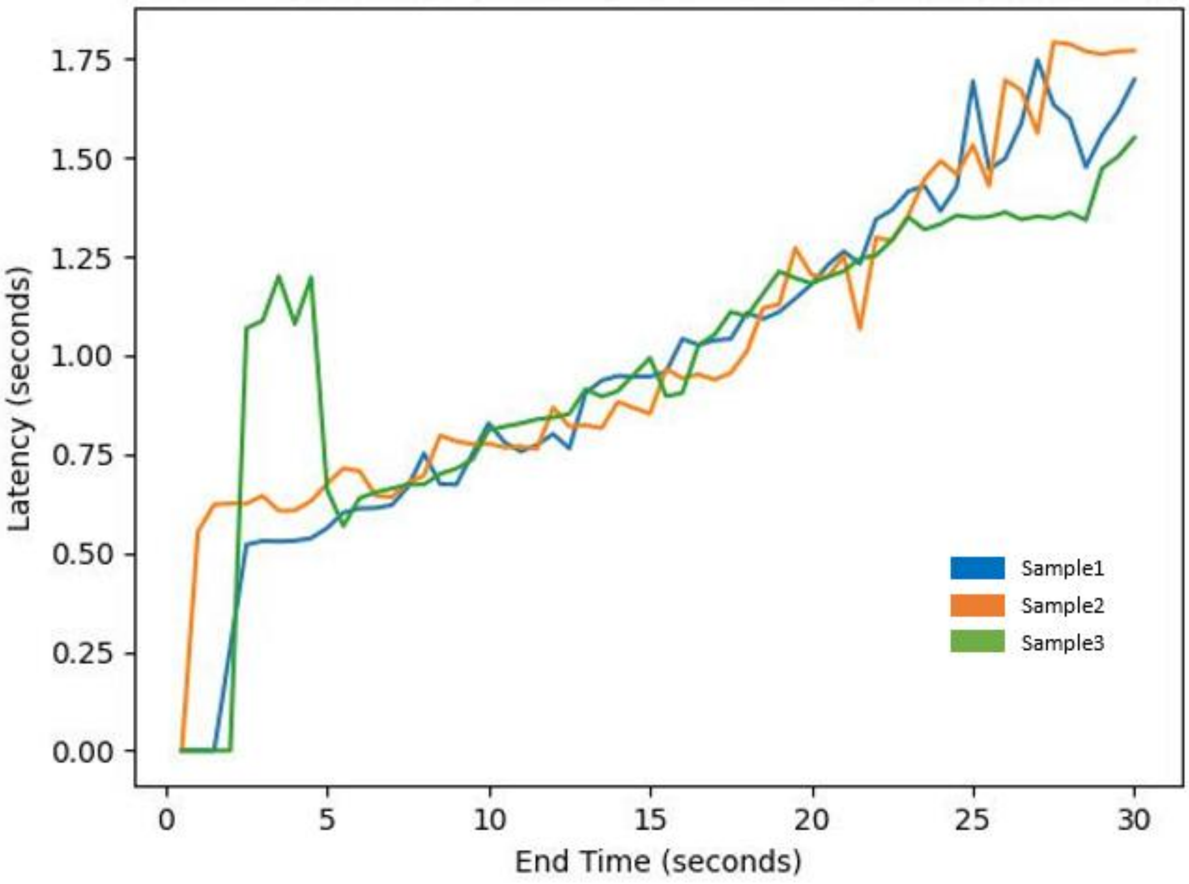


Figure IV.8: Accumulated latencies over 30s with beam search

IV.4 Adapted model compared to models of previous work

In this section, we will perform a comparative analysis between our whisper model and previous research models. We will examine various aspects of our system design and model selection.

Training Data: When it comes to training data in recent times **there is 2 approaches**, One is to train the models with **only gold standard data** free from transcription errors and misaligned or missing audio **the latter** is to **train the model on noisy-data** with **less quality data** but with **more quantity** but it may contain transcription errors and audio issues.

For the first approach we have **3 models trained on only gold standard data** which are **Kaldi ASpIRE** , **DeepSpeech** and **NVIDIA NeMo Conformer models** for the **second approach** we have **Whisper** and **HuBERT** models. The **primary distinction** between the training of HuBERT and Whisper lies in the fact that **HuBERT underwent pre-training on the unlabeled Libri-Light** dataset, whereas **Whisper's training utilized a large scale labeled dataset.** In terms of the **diversity and quantity of training data**, Kaldi ASpIRE was **exclusively trained on an augmented Fisher dataset with added noise**, which is **insufficient to encompass the full spectrum of real-life speaker diversity**. This dataset comprised only approximately **1,700 hours of audio**. On the other hand, DeepSpeech was trained using a **broader range of datasets**, albeit **without noise augmentation**. Consequently, DeepSpeech **exhibits a bias favoring low-noise settings and clear audio inputs**, and it also leans towards **representing US** males and this according to DeepSpeech own model description on GitHub. The total training duration for DeepSpeech amounted to approximately **7,500 hours**.

Within the first approach, the **NVIDIA NeMo Conformer models boasted the most extensive and diverse dataset**, encompassing approximately **22,000 hours of audio**. However, this still **falls behind models trained using the second approach**, **As HuBERT was pre-trained on 60,000 hours of unlabeled audio** and then **fine-tuned on 960 hours of LibriSpeech** labeled audio data. However, it's worth noting that all **models from both the first and second approaches pale in comparison to Whisper's training dataset**, which comprises a staggering **438,218 hours of labeled audio data Plus our 100h of AMI dataset fine-tuning**. This dataset excels in **capturing the utmost diversity in speaker characteristics,**

environmental conditions, and speaking styles. Importantly, it should be highlighted that **Whisper's training data is approximately 620% larger than that of HuBERT.**

Complexity and Architecture: For real-time application like ours we require that the model **doesn't have significant latency** , and for latency what it affects it most is the models architecture used , for an example **Kaldi ASpIRE** uses a has **TDNN-BLSTM architecture** which is a hybrid acoustic model consisting of TDNN layers and BLSTM layers with **biggest latency hurdle been BLSTM** because while still struggling with **longer-range dependencies they are inherently bidirectional,** meaning they process input sequences in both the forward and backward directions which is slow. On top this architecture only process input sequences sequentially, one time step at a time unlike transformer based models that **parallelize the input sequence processing** making them **much faster.** This applies to **DeepSpeech** also as it's based on **RNN-LSTM** which only process sequences sequentially.

But when it comes to **HuBERT**, **NVIDIA NeMo Conformer** and **Whisper** they are based on the transformers architecture which reduces latency significantly. While HuBERT and Whisper are closer to vanilla transformer, the NVIDIA NeMo Conformer **lacks computational and memory efficiency as it adds convolutional layers on top** and its autoregressive decoding variant introduces **sequential dependencies** which means it **generates output tokens one at a time in a sequential manner.** The **table below** provides a concise overview of the key characteristics of various transformer model variants of Whisper, HuBERT and NVIDIA Nemo Conformer. With **Model(1)** been **HuBERT Large** variant , **Model(2) HuBERT Extra Large** Variant, **Model(3)** NVIDIA NeMo **Conformer-CTC Large** variant , **Model(4)** NVIDIA NeMo **Conformer-Transducer Extra Large** Variant , Model(5) **Whisper "Base"** and Model(6) is **Whisper "Small"**. For non-transformer based like Kaldi and DeepSpeech it's hard to estimate number of parameters because of their design but we estimate that DeepSpeech has 47 million parameters for its acoustic model and 50-100 million for its language model meaning about 97-147 million parameters in total, Kaldi ASpIRE has about 64-75 million parameters. As a fact **if we combine both the "base" whisper variant and Opus-MT** we will have **only 149 Million parameter**.

Table IV.6: Models details of the transformers based variants.

Model	Parameters	Attention heads	Embedding width	Models Layers
1	316 Million	16	1024	24
2	963 Million	16	1280	48
3	120 Million	8	512	17Encoder,1Decoder
4	600 Million	8	1024	24Encoder,1Decoder
5	74 Million	8	512	6
6	244 Million	12	768	12

When it comes to practicality, the most effective and conventional approach is to rely on Word Error Rate (WER), which encapsulates a model's performance within a single metric. The table presented below showcases the **%WER results for various models** tested **on the AMI dataset**. This dataset poses **significant challenges**, given its inclusion of non-native speakers, background noise from other speakers, and frequent pauses. Nonetheless, it closely mirrors real-life meeting scenarios, essentially representing spontaneous speech situations. In the assessment of DeepSpeech, we encountered difficulties with the inference engine, preventing us from obtaining a conclusive %WER result. Instead, we were only able to provide an estimated range.

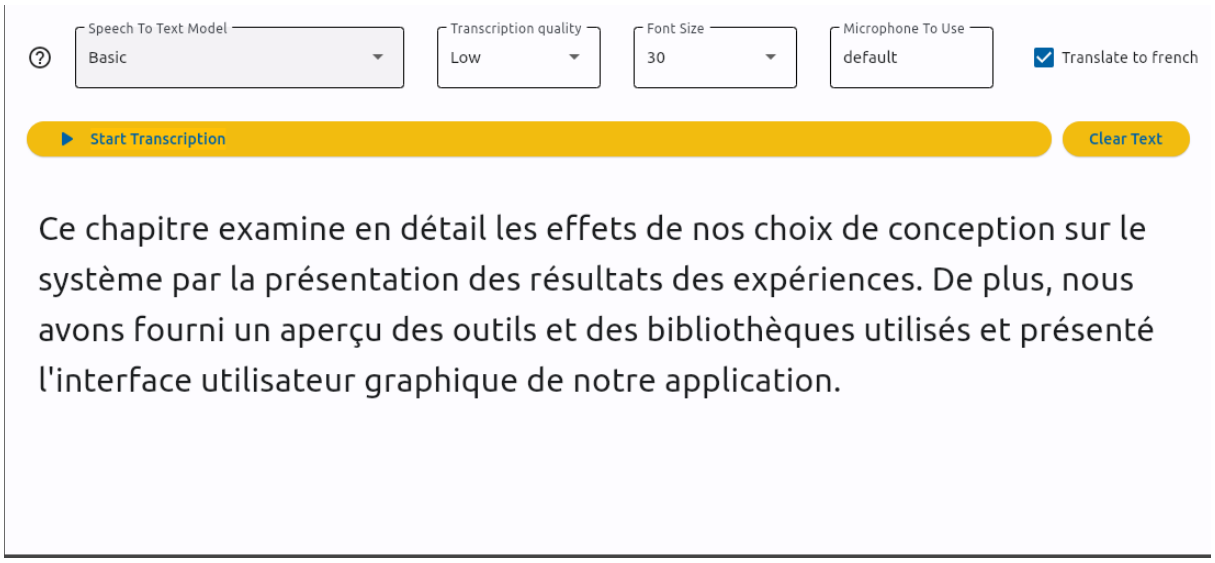
Table IV.7: Comparison of (%WER) between adapted models and models from prior research

Model	Word Error Rate (%WER)
HuBERT Large	33.7%
HuBERT Extra Large	33.3%
Kaldi ASpIRE	35.82%
DeepSpeech	40% to 60%
Conformer-CTC Large	15.9%
Conformer-Transducer Extra Large	20.5%
Adapted “Base” Whisper	14.12%
Adapted “Small” Whisper	11.42%

It is evident from the table above that the **'Base' model outperforms all previous models despite having only 74 million parameters**, making it less complex than any other model. Its performance falls short only when compared to our Whisper 'Small' variant.

IV.5 Graphical user interface

To simplify the use of our app for the average user, we have developed a Graphical User Interface (GUI) that prioritizes a minimalist design. Our primary objective was to create an interface that is not only easy to understand but also intuitive to interact with. By concealing complex details, we aimed to provide a streamlined user experience, allowing users to focus on the core functionality of the application.
The graphical user interface (GUI) of our application is depicted in Figure IV.9.



Ce chapitre examine en détail les effets de nos choix de conception sur le système par la présentation des résultats des expériences. De plus, nous avons fourni un aperçu des outils et des bibliothèques utilisés et présenté l'interface utilisateur graphique de notre application.

Figure IV.9: Graphical user interface for our transcription application.

Our GUI offers users a range of options to select from, designed to enhance usability and avoid confusion. To assist users in understanding the choices available, we have included a **tooltip icon "?"** that **provides explanations** for each dropdown menu.

For the **"Speech to Text Model" dropdown** menu, as shown in Figure IV.10, users can choose between different options. The **"Basic"** model which is whispers **"tiny"** model suitable for general-purpose and the **"AMI Base"** and **"AMI Small"** models which are our fine-tuned models targeted for meetings.

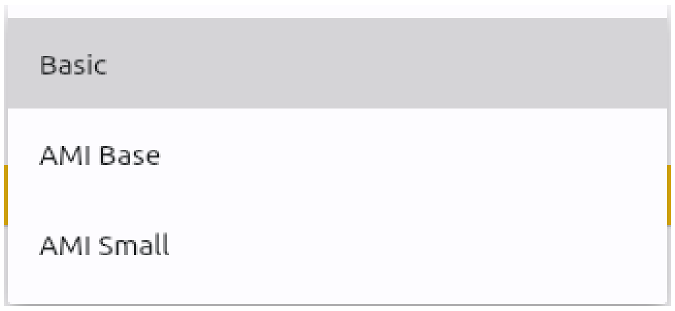


Figure IV.10: "Speech to Text Model" dropdown.

Within the **"Transcription quality" dropdown** menu, depicted in Figure IV.11, users are presented with three options: "**Low**," "**Medium**," and "**High**." The "Low" setting utilizes greedy decoding for Opus-MT and Whisper along a 20-second buffer to enable shorter contextual analysis. In contrast, the "Medium" setting also uses greedy decoding but utilizes a longer buffer of 30 seconds, leading to improved transcription quality. Lastly, the "High" utilizes a 30-second buffer with beam search of size 5 for both Opus-MT and Whisper along with a temperature fallback for whisper.

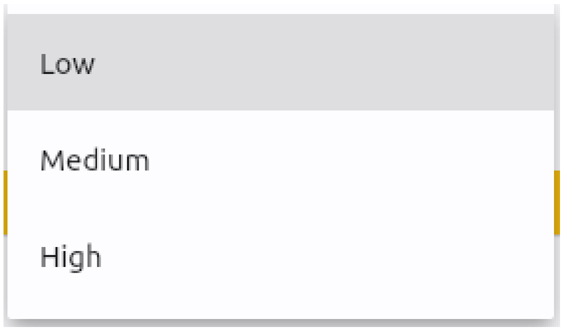


Figure IV.11: "Transcription quality" dropdown.

Additionally, we have incorporated a **"Font Size" dropdown** menu, enabling users to adjust the font size of the transcription output. This **range spans from 8 to 70**, catering to individual preferences. To facilitate audio input selection, we have included a **"Microphone to Use" dropdown** menu, enabling users to choose from their available microphone options. Furthermore, a **"Translate to French" checkbox** has been included, allowing users to opt for the translation of English transcriptions to French. Lastly, we have integrated a **"Clear Text" button**, providing users with the ability to easily clear the output text as desired.

IV.6 Conclusion

This chapter comprehensively examines the effects of our design choices on the system through the presentation of experiments results. Furthermore, we provided an overview of the tools and libraries employed, and showcased the graphical user interface (GUI) of our application.

General conclusion

In this thesis, our **primary objective** was to design a system capable of transcribing English speech, translating it into French, and effectively handling noisy environments while ensuring clear audio quality. Additionally, we focused on developing a system that could accurately transcribe the vocabulary commonly employed in live meetings. Furthermore, our goal was to achieve **real-time performance with reasonable latency for the system**.

Our **key contribution** involved designing an audio capture mechanism capable of obtaining noise-free, clear audio in real-time, even within noisy environments. We proceeded by fine-tuning and adapting a **transcription system specifically for meetings**, conducting a series of experiments to evaluate its performance. Moreover, we incorporated a translation system that could capture and translate text while preserving context and meaning. To ensure efficient operation, we focused on **minimizing latency** by conducting multiple experiments and optimizing the models for low-performance hardware. Furthermore, we **created an intuitive and responsive graphical user interface** to facilitate user interaction with our system.

While endeavoring to develop a robust system, we **encountered various problems and challenges** that imposed **limitations** on its performance. The most notable limitations include the system's inability to capture all words accurately within a meeting setting, despite substantial improvements. This shortfall can be attributed primarily to the significant hardware requirements necessary for improving such complex artificial intelligence models. Moreover, despite our efforts to mitigate transcription errors through the acquisition of clean audio by noise reduction methods, occasional **hallucinations still occur** where the system transcribes words the speaker did not say. In terms of real-time capabilities, while we have **achieved reasonable latency**, there is room for improvement. Deploying our system on more robust hardware has the potential to further elevate real-time performance. Alternatively, **optimizing our machine learning models** through techniques such as **model pruning or distillation** is another avenue worth exploring to achieve this objective.

Reference

[1] D. Yu, L. Deng, and Springerlink (Online Service, *Automatic Speech Recognition: A Deep Learning Approach*. London: Springer London, 2015.

[2] X. Huang, A. Acero, and H.-W. Hon, *Spoken Language Processing*. Prentice Hall, 2001.

[3] J. Pierce, "Sound waves and sine waves," in Music, Cognition, and Computerized Sound: An Introduction to Psychoacoustics, P. R. Cook, Ed. Cambridge, MA: MIT Press, 1999, pp. 37-56.

[4] R. Ahmed Khalil, "Adaptive Filter Application in Echo Cancellation System and Implementation using FPGA," AL-Rafdain Engineering Journal (AREJ), vol. 16, no. 5, pp. 20–32, Dec. 2008, doi: https://doi.org/10.33899/rengj.2008.44853.

[5] D. A. Reynolds, T. F. Quatieri, and R. B. Dunn, "Speaker Verification Using Adapted Gaussian Mixture Models," *Digital Signal Processing*, vol. 10, no. 1–3, pp. 19–41, Jan. 2000, doi: https://doi.org/10.1006/dspr.1999.0361.

[6] T. D. Rossing, P. A. Wheeler, and F. Richard Moore, *The Science of Sound*. Pearson, 2002.

[7] L. R. Rabiner and B. H. Juang, *Fundamentals of speech recognition*. Englewood Cliffs, N.J.: Ptr Prentice Hall, 1993.

[8] Hinton, G., Deng, L., Yu, D., Dahl, G. E., Mohamed, A. R., Jaitly, N., ... & Kingsbury, B. (2012). Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. IEEE Signal Processing Magazine, 29(6), 82-97.

[9] "Time and Frequency Domains," *elvers.us*. https://elvers.us/perception/soundWave/

[10] D. Griffin and Jae Lim, "Signal estimation from modified short-time Fourier transform," in IEEE Transactions on Acoustics, Speech, and Signal Processing, vol. 32, no. 2, pp. 236-243, April 1984, doi: 10.1109/TASSP.1984.1164317.

[11] Wikipedia Contributors, "Spectrogram," *Wikipedia*, Oct. 17, 2019. https://en.wikipedia.org/wiki/Spectrogram

[12] M. Massoudi, S. Verma, and R. Jain, "Urban Sound Classification using CNN," IEEE Xplore, Jan. 01, 2021. https://ieeexplore.ieee.org/document/9358621/

[13] Janiesch, C., Zschech, P., & Heinrich, K. (2021). Machine learning and deep learning. Electronic Markets, 31(3), 685–695. https://doi.org/10.1007/s12525-021-00475-2

[14] C. M. Bishop, Pattern Recognition and Machine Learning. Springer, 2016.

[15] D. Silver *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016, doi: https://doi.org/10.1038/nature16961.

[16] V. R. Joseph, "Optimal ratio for data splitting," Statistical Analysis and Data Mining: The ASA Data Science Journal, vol. 15, no. 4, pp. 531–538, Apr. 2022, doi: https://doi.org/10.1002/sam.11583.

[17] D. Yu and L. Deng, *Automatic Speech Recognition*. London: Springer London, 2015. doi: https://doi.org/10.1007/978-1-4471-5779-3.

[18] H. Liu and M. Cocea, *Granular computing based machine learning*. Cham: Springer, 2018.

[19] J. Singh and R. Banerjee, "A Study on Single and Multi-layer Perceptron Neural Network," 2019 3rd International Conference on Computing Methodologies and Communication (ICCMC), 2019, doi: https://doi.org/10.1109/ICCMC.2019.8819775.

[20] D. Hendrycks, "Gaussian Error Linear Units (GELUs)," *arXiv.org*, Jun. 27, 2016. https://arxiv.org/abs/1606.08415

[21] B. Hamdi, S. Limam, and T. Aguili, "UNIFORM AND CONCENTRIC CIRCULAR ANTENNA ARRAYS SYNTHESIS FOR SMART ANTENNA SYSTEMS USING ARTIFICIAL NEURAL NETWORK ALGORITHM," *Progress In Electromagnetics Research B*, vol. 67, pp. 91–105, 2016, doi: https://doi.org/10.2528/pierb16031508.

[22] Bre, F., Gimenez, J. M., & Fachinotti, V. D. (2018). Prediction of wind pressure coefficients on building surfaces using artificial neural networks. Energy and Buildings, 158, 1429–1441. https://doi.org/10.1016/j.enbuild.2017.11.045

[23] A. Vaswani et al., "Attention Is All You Need," *arXiv.org*, Jun. 12, 2017. https://arxiv.org/abs/1706.03762

[24] S. Lakew, M. Fondazione, B. Kessler, M. Federico, M. Srl, and T. Fondazione, “A Comparison of Transformer and Recurrent Neural Networks on Multilingual Neural Machine Translation,” 2018. Available: https://aclanthology.org/C18-1054.pdf

[25] A. Farizawani, M. Puteh, Y. Marina, and A. Rivaie, “A review of artificial neural network learning rule based on multiple variant of conjugate gradient approaches,” *Journal of Physics: Conference Series*, vol. 1529, p. 022040, Apr. 2020, doi: https://doi.org/10.1088/1742-6596/1529/2/022040.

[26] D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” *arXiv.org*, Dec. 22, 2014. https://arxiv.org/abs/1412.6980

[27] D. Povey et al., Eds., The Kaldi Speech Recognition Toolkit. IEEE Signal Processing Society, 2011. Available: https://infoscience.epfl.ch/record/192584.

[28] “Kaldi ASR, ASpIRE Chain Model” kaldi-asr.org. https://kaldi-asr.org/models/m1 .

[29] C. Cieri, D. Miller, and K. Walker, “The Fisher Corpus: a Resource for the Next Generations of Speech-to-Text.” Available: http://www.lrec-conf.org/proceedings/lrec2004/pdf/767.pdf.

[30] mozilla, “mozilla/DeepSpeech,” GitHub, Dec. 10, 2020. https://github.com/mozilla/DeepSpeech

[31] A. Hannun et al., “Deep Speech: Scaling up end-to-end speech recognition.” Available: https://arxiv.org/pdf/1412.5567.pdf

[32] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, “LIBRISPEECH: AN ASR CORPUS BASED ON PUBLIC DOMAIN AUDIO BOOKS.” Available: http://www.danielpovey.com/files/2015_icassp_librispeech.pdf

[33] “Common Voice by Mozilla,” commonvoice.mozilla.org. https://commonvoice.mozilla.org/en

[34] “SWITCHBOARD: telephone speech corpus for research and development,” ieeexplore.ieee.org. https://ieeexplore.ieee.org/document/225858 .

[35] W.-N. Hsu, B. Bolte, Y.-H. H. Tsai, K. Lakhotia, R. Salakhutdinov, and A. Mohamed, “HuBERT: Self-Supervised Speech Representation Learning by Masked Prediction of Hidden Units,” arXiv:2106.07447 [cs, eess], Jun. 2021, Available: https://arxiv.org/abs/2106.07447

[36] M. Caron, P. Bojanowski, A. Joulin, and M. Douze, “Deep Clustering for Unsupervised Learning of Visual Features,” arXiv:1807.05520 [cs], Mar. 2019, Available: https://arxiv.org/abs/1807.05520

[37] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,” arXiv.org, Oct. 11, 2018. https://arxiv.org/abs/1810.04805

[38] J. Kahn et al., “Libri-Light: A Benchmark for ASR with Limited or No Supervision,” HAL (Le Centre pour la Communication Scientifique Directe), May 2020, doi: https://doi.org/10.1109/icassp40776.2020.9052942.

[39] O. Kuchaiev et al., “NeMo: a toolkit for building AI applications using Neural Modules,” arXiv:1909.09577 [cs, eess], Sep. 2019, Available: https://arxiv.org/abs/1909.09577

[40] A. Gulati et al., “Conformer: Convolution-augmented Transformer for Speech Recognition,” arXiv:2005.08100 [cs, eess], May 2020, Available: https://arxiv.org/abs/2005.08100

[41] D. Galvez et al., “The People’s Speech: A Large-Scale Diverse English Speech Recognition Dataset for Commercial Usage.” Available: https://arxiv.org/pdf/2111.09344.pdf

[42] A. Radford, J. Kim, T. Xu, G. Brockman, C. Mcleavey, and I. Sutskever, “Robust Speech Recognition via Large-Scale Weak Supervision.” Available: https://cdn.openai.com/papers/whisper.pdf

[43] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural Language Processing (Almost) from Scratch," Journal of Machine Learning Research, vol. 12, pp. 2493-2537, 2011.

[44] D. Jurafsky and J. H. Martin, "Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition," Pearson, 2019.

[45] I. Mani and M. T. Maybury, "Automatic Text Summarization," The MIT Press, 1999.

[46] C. D. Manning and Hinrich Schütze, *Foundations of statistical natural language processing*. Cambridge, Mass.: Mit Press, 2000.

[47] "The Stanford Natural Language Processing Group," *nlp.stanford.edu*. https://nlp.stanford.edu/software/tokenizer.shtml.

[48] P. Koehn et al., "Moses: Open Source Toolkit for Statistical Machine Translation," ACLWeb, Jun. 01, 2007. https://aclanthology.org/P07-2045/

[49] "Statistical and Neural Machine Translation," statmt.org. https://statmt.org/

[50] F. J. Och and H. Ney, "A Systematic Comparison of Various Statistical Alignment Models," Computational Linguistics, vol. 29, no. 1, pp. 19–51, Mar. 2003, doi: https://doi.org/10.1162/089120103321337421.

[51] A. Fan et al., "Beyond English-Centric Multilingual Machine Translation," arXiv:2010.11125 [cs], Oct. 2020, Available: https://arxiv.org/abs/2010.11125

[52] H. Schwenk, G. Wenzek, S. Edunov, E. Grave, and A. Joulin, "CCMatrix: Mining Billions of High-Quality Parallel Sentences on the WEB," arXiv.org, May 01, 2020. https://arxiv.org/abs/1911.04944 .

[53] A. El-Kishky, V. Chaudhary, F. Guzman, and P. Koehn, "CCAligned: A Massive Collection of Cross-Lingual Web-Document Pairs," arXiv.org, Oct. 11, 2020. https://arxiv.org/abs/1911.06154 .

[54] NLLB Team et al., "No Language Left Behind: Scaling Human-Centered Machine Translation," arXiv:2207.04672 [cs], Aug. 2022, Available: https://arxiv.org/abs/2207.04672

[55] J. Tiedemann and S. Thottingal, "OPUS-MT -Building open translation services for the World." Available: https://aclanthology.org/2020.eamt-1.61.pdf

[56] "OPUS - an open source parallel corpus," opus.nlpl.eu. https://opus.nlpl.eu/

[57] A. Veysov, "Silero VAD," *GitHub*, Mar. 04, 2022. https://github.com/snakers4/silero-vad

[58] "One Voice Detector to Rule Them All," *The Gradient*, Feb. 19, 2022. https://thegradient.pub/one-voice-detector-to-rule-them-all/

[59] K. Yang, V. Yao, J. Denero, and D. Klein, "A Streaming Approach for Efficient Batched Beam Search." Available: https://arxiv.org/pdf/2010.02164.pdf

[60] W. Kool, H. Van Hoof, and M. Welling, “Stochastic Beams and Where to Find Them: The Gumbel-Top-k Trick for Sampling Sequences Without Replacement.”. Available: https://arxiv.org/pdf/1903.06059.pdf

[61] F. Hernandez, V. Nguyen, S. Ghannay, N. Tomashenko, and Y. Estève, “TED-LIUM 3: twice as much data and corpus repartition for experiments on speaker adaptation,” *arXiv:1805.04699 [cs]*, vol. 11096, pp. 198–208, 2018, doi: https://doi.org/10.1007/978-3-319-99579-3_21.

[62] “AMI Corpus,” *groups.inf.ed.ac.uk*. https://groups.inf.ed.ac.uk/ami/corpus/

[63] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, “Dive into Deep Learning,” arXiv:2106.11342 [cs], Jul. 2021, Available: https://arxiv.org/abs/2106.11342

[64] T. Kudo and J. Richardson, “SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing,” arXiv:1808.06226 [cs], Aug. 2018, Available: https://arxiv.org/abs/1808.06226

[65] W. Chen, E. Matusov, S. Khadivi, and J.-T. Peter, “Guided Alignment Training for Topic-Aware Neural Machine Translation,” *arXiv.org*, Jul. 06, 2016. https://arxiv.org/abs/1607.01628

[66] R. Östling and J. Tiedemann, “Efficient Word Alignment with Markov Chain Monte Carlo,” *The Prague Bulletin of Mathematical Linguistics*, vol. 106, no. 1, pp. 125–146, Oct. 2016, doi: https://doi.org/10.1515/pralin-2016-0013

[67] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, “BLEU: a Method for Automatic Evaluation of Machine Translation,” Jul. 2002. Available: https://aclanthology.org/P02-1040.pdf

[68] A. Anastasopoulos et al., “TICO-19: the Translation Initiative for COvid-19,” ACLWeb, Dec. 01, 2020. https://aclanthology.org/2020.nlpcovid19-2.5/ .

[69] J. Tiedemann, “The Tatoeba Translation Challenge – Realistic Data Sets for Low Resource and Multilingual MT,” ACLWeb, Nov. 01, 2020. https://aclanthology.org/2020.wmt-1.139/ .

[70] N. Goyal et al., “The FLORES-101 Evaluation Benchmark for Low-Resource and Multilingual Machine Translation,” arXiv.org, Jun. 06, 2021. https://arxiv.org/abs/2106.03193 .

[71] “WMT-News,” opus.nlpl.eu. https://opus.nlpl.eu/WMT-News.php .

[72] D. Elliott, S. Frank, L. Barrault, F. Bougares, and L. Specia, “Findings of the Second Shared Task on Multimodal Machine Translation and Multilingual Image Description,” ACLWeb, Sep. 01, 2017. https://aclanthology.org/W17-4718/ .

[73] A. Bhandare et al., “Efficient 8-Bit Quantization of Transformer Neural Machine Language Translation Model.” Available: https://arxiv.org/pdf/1906.00532.pdf

[74] Y. Wu *et al.*, “Google’s Neural Machine Translation System: Bridging the Gap between Human and Machine Translation.” Available: https://arxiv.org/pdf/1609.08144.pdf

[75] Microsoft, “Visual Studio Code,” *Visualstudio.com*, Apr. 14, 2016. https://code.visualstudio.com/

[76] Anaconda, “Anaconda,” *Anaconda*. https://www.anaconda.com/

[77] Jupyter, “Project Jupyter,” *Jupyter.org*. https://jupyter.org/

[78] TensorFlow, “TensorBoard TensorFlow,” https://www.tensorflow.org/tensorboard

[79] “The Fast Remote Desktop Application – AnyDesk,” *Anydesk.com*, 2019. https://anydesk.com/en

[80] JetBrains, “PyCharm,” *JetBrains*. https://www.jetbrains.com/pycharm/

[81] “CTranslate2,” *GitHub* .https://github.com/OpenNMT/CTranslate2

[82] G. Klein, “Faster Whisper transcription with CTranslate2” GitHub. https://github.com/guillaumekln/faster-whisper

[83] “huggingface/transformers,” *GitHub*. https://github.com/huggingface/transformer

[84] “JiWER: Similarity measures for automatic speech recognition evaluation,” GitHub https://github.com/jitsi/jiwer

[85] “PyAudio: Cross-platform audio I/O for Python, with PortAudio,” *people.csail.mit.edu*. https://people.csail.mit.edu/hubert/pyaudio/

[86] Numpy, “NumPy,” *Numpy.org*, 2009. https://numpy.org/

[87] Pandas, “Python Data Analysis Library — pandas: Python Data Analysis Library,” Pydata.org. https://pandas.pydata.org/

[88] “The fastest way to build Flutter apps in Python | Flet,” flet.dev. https://flet.dev/

[89] “SentencePiece,” *GitHub*. https://github.com/google/sentencepiece

[90] Matplotlib, “Matplotlib: Python plotting” *Matplotlib.org*, 2012.
https://matplotlib.org/

[91] O. R. developers, “ONNX Runtime,” *GitHub*.
https://github.com/microsoft/onnxruntime.

[92] B. McFee et al., “librosa/librosa: 0.10.1,” Zenodo,
https://zenodo.org/record/8252662

[93] “One of The Greatest Speeches Ever by President Obama,” www.youtube.com.
https://www.youtube.com/watch?v=NaaSpRMBHjg

[94] “Texas, Export Shipping to Mexico,” Dictionary of American Regional English.
https://dare.wisc.edu/audio/texas-export-shipping-to-mexico/

[95] “The World Needs More of You | 1 Minute Motivational Speech,”
www.youtube.com. https://www.youtube.com/watch?v=4FDud9Lj5HY