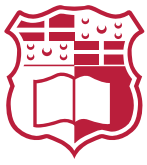
Machine Learning Noise Cancellation System

Graham Pellegrini

Supervisor: Dr Ing. Trevor Spiteri

June 2025

Submitted in partial fulfilment of the requirements
for the degree of Bachelor of Science (Honours) (Computer Engineering).



L-Università ta' Malta
Faculty of Information &
Communication Technology

Abstract

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Mauris sed ipsum risus. Nulla aliquet quis quam sed eleifend. Donec rutrum, dolor id vulputate pharetra, nulla tortor laoreet nisl, pellentesque dapibus velit dolor suscipit purus. Phasellus vitae eleifend sem. Integer ultricies ex in neque pellentesque, vitae facilisis orci aliquam. In pellentesque mollis turpis, eu tristique lacus eleifend nec. Vestibulum orci neque, rhoncus vitae convallis eu, suscipit quis dui. Nulla libero elit, porta sit amet sagittis vel, placerat sit amet tortor. Aliquam hendrerit dolor sit amet sollicitudin ornare. Aliquam placerat sodales est, in vestibulum nisl efficitur in. Nulla venenatis aliquam sem, at volutpat nisl pellentesque eleifend. Praesent vitae euismod nulla, eget vehicula turpis. Duis quis tellus vitae nisi tempus tincidunt.

Nam quis aliquet nisi, non pharetra ligula. Phasellus pulvinar mattis neque, nec interdum justo condimentum hendrerit. Mauris fermentum venenatis faucibus. Pellentesque egestas eleifend libero, quis placerat ante fermentum et. Suspendisse accumsan gravida rhoncus. Vestibulum auctor sodales vehicula. Pellentesque a urna et elit placerat laoreet a quis turpis. Morbi ut sem at nunc posuere malesuada vel nec libero. Nam et urna suscipit, bibendum diam sed, aliquet leo. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut pretium mauris et nulla malesuada, mattis tristique lectus molestie. Aenean accumsan iaculis quam, eget varius libero placerat eu. Fusce mauris justo, vulputate a sollicitudin a, malesuada at est. Sed ac augue elit. Maecenas massa lorem, tincidunt vitae neque et, maximus dictum tellus.

Acknowledgements

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Mauris sed ipsum risus. Nulla aliquet quis quam sed eleifend. Donec rutrum, dolor id vulputate pharetra, nulla tortor laoreet nisl, pellentesque dapibus velit dolor suscipit purus. Phasellus vitae eleifend sem. Integer ultricies ex in neque pellentesque, vitae facilisis orci aliquam. In pellentesque mollis turpis, eu tristique lacus eleifend nec. Vestibulum orci neque, rhoncus vitae convallis eu, suscipit quis dui. Nulla libero elit, porta sit amet sagittis vel, placerat sit amet tortor. Aliquam hendrerit dolor sit amet sollicitudin ornare. Aliquam placerat sodales est, in vestibulum nisl efficitur in. Nulla venenatis aliquam sem, at volutpat nisl pellentesque eleifend. Praesent vitae euismod nulla, eget vehicula turpis. Duis quis tellus vitae nisi tempus tincidunt.

Nam quis aliquet nisi, non pharetra ligula. Phasellus pulvinar mattis neque, nec interdum justo condimentum hendrerit. Mauris fermentum venenatis faucibus. Pellentesque egestas eleifend libero, quis placerat ante fermentum et. Suspendisse accumsan gravida rhoncus. Vestibulum auctor sodales vehicula. Pellentesque a urna et elit placerat laoreet a quis turpis. Morbi ut sem at nunc posuere malesuada vel nec libero. Nam et urna suscipit, bibendum diam sed, aliquet leo. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut pretium mauris et nulla malesuada, mattis tristique lectus molestie. Aenean accumsan iaculis quam, eget varius libero placerat eu. Fusce mauris justo, vulputate a sollicitudin a, malesuada at est. Sed ac augue elit. Maecenas massa lorem, tincidunt vitae neque et, maximus dictum tellus.

Contents

List of Figures

List of Tables

List of Abbreviations

AI Artificial Intelligence.

ANC Active Noise Cancellation.

CED Convolutional Encoder-Decoder.

CNN Convolutional Neural Network.

Conv-TasNet Convolutional Time-domain Audio Separation Network.

DSP Digital Signal Processing.

EER Equal Error Rate.

FCN Fully Convolutional Network.

FP16 16-bit Floating Point.

FP32 32-bit Floating Point.

GC Garbage Collection.

ISTFT Inverse Short-Time Fourier Transform.

LSD Log-Spectral Distance.

ML Machine Learning.

MMSE-LSA Minimum Mean-Square Error Log-Spectral Amplitude.

MSE Mean Squared Error.

OOM Out-Of-Memory.

PESQ Perceptual Evaluation of Speech Quality.

PReLU Parametric Rectified Linear Unit.

PTO Padding-Truncation Output-Truncation.

R-CED Residual Convolutional Encoder-Decoder.

ReLU Rectified Linear Unit.

SNR Signal-to-Noise Ratio.

SS Spectral Subtraction.

SSH Secure Shell.

STFT Short-Time Fourier Transform.

STOI Short-Time Objective Intelligibility.

TCN Temporal Convolutional Network.

UNet U-shaped Convolutional Neural Network.

VSCode Visual Studio Code.

WF Wiener Filtering.

Glossary of Symbols

$|D(\omega)|$ Estimated magnitude spectrum of the noise.

$H(\omega)$ Frequency-dependent Wiener gain.

$\hat{x}(n)$ Estimated enhanced speech signal at sample n.

N Total number of samples in the signal.

P_n Power of the noise (or error) signal.

P_s Power of the clean speech signal.

$S(f,t)$ Clean magnitude spectrum at frequency bin f, time frame t.

$\hat{S}(f,t)$ Enhanced magnitude spectrum at frequency bin f, time frame t.

$|X(\omega)|$ Magnitude spectrum of the clean signal.

$x(n)$ Clean reference speech signal at sample n.

$|Y(\omega)|$ Magnitude spectrum of the noisy speech.

1 Introduction

Machine Learning (ML), commonly referred to as Artificial Intelligence (AI), has seen rapid advancements over the past decade. Largely driven by increasing computational power and the availability of large datasets. These techniques are now widely applied across various domains to solve complex problems, including speech processing and noise cancellation. However, the adoption of AI in such areas often lacks thorough evaluation, with the justifications for their computational demands not always clearly addressed.

This project focuses on noise cancellation in audio signals for speech enhancement. An area with growing importance in telecommunications, assistive technologies, and real-time communication systems. Background noise can significantly degrade the quality of speech, making it difficult to interpret or process spoken information. Such noise is generally categorized into two types: stationary and non-stationary [1].

- **Stationary noise** has relatively constant properties over time, such as white noise or the hum of an appliance.
- **Non-stationary noise** fluctuates unpredictably, such as traffic, crowd chatter, or sudden environmental sounds.

Speech enhancement aims to suppress these unwanted noise components while preserving speech intelligibility. Classical noise cancellation techniques, such as the Wiener filter, work by estimating and subtracting noise from the signal, but they often assume that the noise is stationary. This assumption limits their effectiveness in real-world conditions.

ML approaches, on the other hand, have emerged as powerful alternatives. Capable of learning complex, data-driven patterns to separate speech from noise without relying on such rigid assumptions.

1.1 Project Goals and Implementation

The primary goal of the project is to explore both classical and ML based approaches to noise cancellation, evaluating their effectiveness and feasibility in real-world applications. The project will not only examine established noise reduction techniques but also develop a ML based model, comparing its performance against classical methods.

The approach involves designing and implementing a noise cancellation system that operates under the assumption of a single speaker scenario in a noisy background

environment, where the system must remove both stationary and non-stationary noise without access to a clean reference signal. Unlike established pre-trained models that required extensive datasets and resources, the model developed in this project will not be pre-trained. Allowing for a demonstration of the ease of developing and training in practical settings. Pre-trained models will be used as part of the evaluation process, but the focus will be on the model developed in this project.

The project will be implemented in Python using a modular and reproducible structure, ensuring that the framework can be extended or modified for further improvements. The project is structured into two core phases training and denoising:

- Training phase: The most computationally intensive stage, where a sourced dataset of clean and noisy speech samples will be formatted and fed into the model to learn respective mapping.
- Denoising phase: Involves loading the trained model and applying it to a noisy speech signal to generate a cleaned output. Here evaluation metrics will also be producible to assess the performance of one methodology against another.

It would be good to give a short overview of the remaining chapters.

2 Background

In this chapter, background information on established classical noise reduction methods is provided. It explores Spectral Subtraction, Wiener Filtering, and MMSE-LSA estimation, which does not require reference signals.The basis of ML models, autoencoders, is introduced. Additionally, the chapter introduces the key evaluation metrics used to assess speech enhancement performance. Finally, a brief section outlines the progression of the project and relevant foundational knowledge.

2.1 Spectral Subtraction

SS is one of the most widely used techniques for single-channel speech enhancement. The core idea is to estimate the noise spectrum from a noisy speech signal and subtract it from the observed spectrum to obtain a cleaner signal [1]. It assumes that noise remains relatively stationary over short time frames, while speech is a dynamic, non-stationary signal.

The analog speech signal $x(t)$ must be digitised into $x(n)$, a discrete-time signal. This digitisation must satisfy the Nyquist criterion, sampling at a rate at least twice the maximum frequency for accurate representation. However, sampling introduces limitations, such as quantisation noise and aliasing. If the sampling rate is too low, high-frequency components may fold into lower frequencies, misrepresenting the signal content [1].

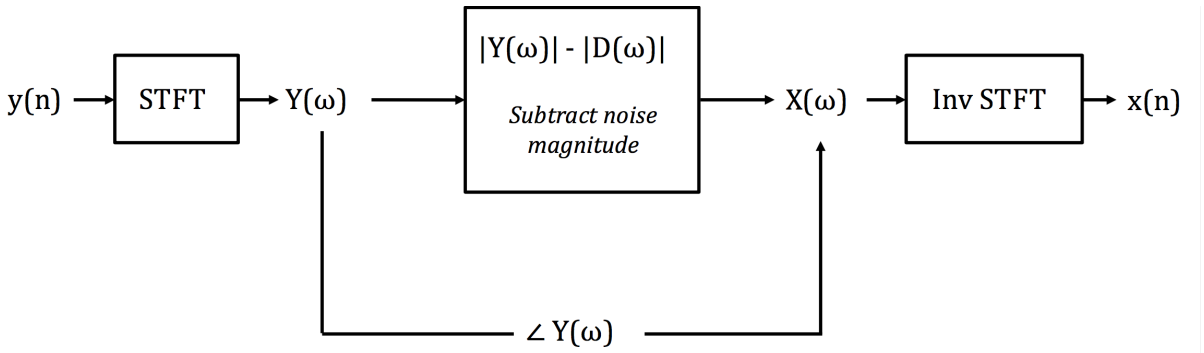


Figure 2.1 Block diagram of the SS method [2].

Before applying SS, the signal is pre-processed into a suitable format using the Short-Time Fourier Transform (STFT). The STFT segments the time-domain signal into overlapping frames using a windowing function and then converts each segment into the frequency domain. Windowing reduces spectral leakage and preserves continuity in the frequency domain. The magnitude spectrum of the noisy signal is obtained by taking the absolute value of the STFT [2]. The noise spectrum is estimated by

averaging the spectral magnitudes of non-speech segments over time. Once this estimate is obtained, it is subtracted from the noisy spectrum to yield an estimate of the clean speech. The final step applies the Inverse Short-Time Fourier Transform (ISTFT) to reconstruct the enhanced signal in the time domain by summing the overlapping frames.

This process can be mathematically described as:

Why double vertical bars?

$$||X(\omega)|| = ||Y(\omega)|| - ||D(\omega)|| \tag{2.1}$$

SS is widely adopted due to its simplicity and low computational cost. It requires only a single input channel and is straightforward to implement. However, it assumes stationary noise, a condition often unmet in practice. It may also introduce musical noise, a type of distortion perceived as tonal artifacts. Excessive subtraction may distort speech, while insufficient subtraction can leave residual noise.

Despite these challenges, SS remains a valuable baseline for evaluating more advanced noise suppression techniques. Methods such as Wiener filtering and deep learning-based approaches have been developed to overcome these shortcomings through more adaptive or data-driven mechanisms [1].

2.2 Wiener Filtering

WF is a statistically grounded method for speech enhancement that aims to minimise Mean Squared Error (MSE) between the estimated clean signal and the true clean signal [1]. Originally developed for linear time-invariant systems, the Wiener filter assumes that both the signal and noise are stationary stochastic processes and seeks the optimal linear estimate of the clean speech given the noisy observation [2].

Though usable in time or frequency domains, the frequency version is preferred for efficiency and STFT compatibility. The key idea is that if the power spectral densities (PSDs) of the clean speech P_s and the noise P_n are known or can be estimated, the optimal filter $H(\omega)$ is derived as:

$$H(\omega) = \frac{P_s}{P_s + P_n} \tag{2.2}$$

This filter acts as a frequency-dependent gain function. Frequencies where the clean speech dominates ($P_S \gg P_N$) are preserved, while noise dominant frequencies ($P_N \gg P_S$) are attenuated. The enhanced speech spectrum is obtained by multiplying the noisy spectrum $Y(\omega)$ with the Wiener gain:

$\hat{x}(\omega)$?

$$\hat{x}(n) = H(\omega)Y(\omega) \tag{2.3}$$

time domain frequency domain

don't mix

The enhanced signal is then reconstructed using the ISTFT.

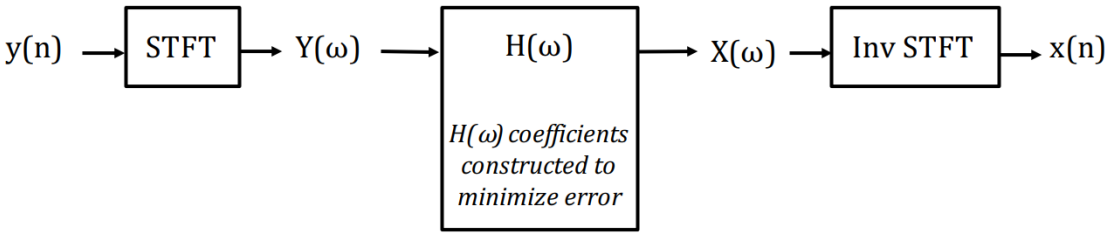


Figure 2.2 Block diagram of the WF process [2].

WF is a statistically optimal method that minimizes the mean square error (MMSE) between the estimated and clean speech signals. It performs well when noise characteristics can be reliably estimated, such as during silent speech segments, and is effective at preserving the spectral shape of speech components. However, the method assumes stationary, zero-mean noise uncorrelated with the speech signal. These assumptions often fail in real-world, dynamic environments. This can lead to inaccurate power spectral density (PSD) estimates, resulting in performance degradation or over-smoothing that reduces intelligibility.

Despite these limitations, WF remains a foundational approach in speech enhancement and is frequently integrated into more advanced or hybrid denoising pipelines [1, 2].

2.3 Minimum Mean Square Error Log-Spectral Amplitude Estimation

The MMSE-LSA estimator is a statistically grounded single-channel speech enhancement technique first proposed by Ephraim and Malah in 1984 [3]. Unlike SS and WF, which operate on magnitude or power spectra. MMSE-LSA aims to minimise the mean-square error between the logarithm of the spectral amplitudes of the clean and enhanced speech signals.

The algorithm operates in the time/frequency domain, assuming additive noise and using the STFT to decompose the noisy signal. It relies on estimates of the a priori and a posteriori signal-to-noise ratios (SNRs) to construct a gain function that enhances speech-dominant regions while suppressing noise. The enhanced spectral amplitude $\hat{X}^{\text{LSA}}(\omega)$ is computed as:

$$\hat{X}^{\text{LSA}}(\omega) = \exp\left(\mathrm{E}\left[\log ||X(\omega)|| \,|\, |Y(\omega)|\right]\right) \tag{2.4}$$

where $|Y(\omega)|$ is the noisy observation and the expectation is taken over the distribution of the clean signal conditioned on the observed noisy spectrum.

A block diagram of the MMSE-LSA estimation process is shown in Figure 2.3. It begins with an STFT of the input signal, followed by separation into magnitude and phase. The magnitude spectrum is used to estimate the noise power spectral density (PSD), from which the a priori and a posteriori SNRs are derived. These estimates are then passed into the MMSE-LSA gain function, which produces a gain mask applied to the magnitude spectrum. The enhanced signal is reconstructed by combining the modified magnitude with the original phase, followed by the inverse STFT.

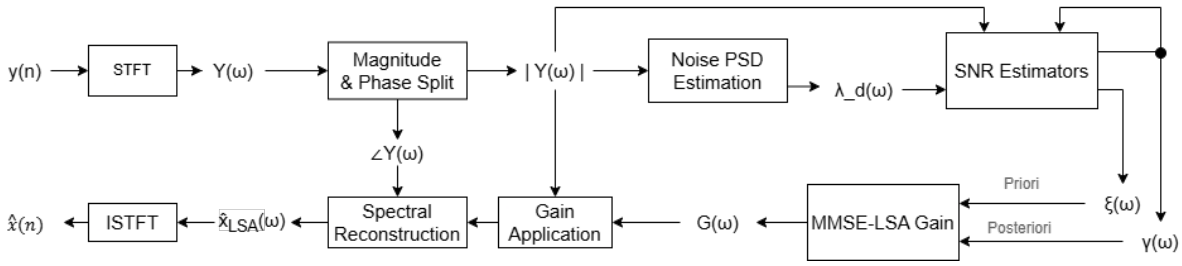


Figure 2.3 Block diagram of the MMSE-LSA method

The MMSE-LSA estimator incorporates a parametric model of the speech and noise distributions and uses recursive averaging to smooth Signal-to-Noise Ratio (SNR) estimates across time frames. This results in a more adaptive and perceptually aligned gain function than traditional methods. This log-domain formulation better aligns with human auditory perception.

Despite its original formulation in the 1980s, MMSE-LSA continues to be a highly relevant technique. Numerous enhancements and application-specific variants have been developed in the 2010s and 2020s, including integrations with wavelet transforms [4], improved noise tracking, and hybrid neural-Digital Signal Processing (DSP) pipelines. This enduring relevance highlights MMSE-LSA's foundational robustness and its suitability for modern speech enhancement systems.

2.4 Autoencoders

ML, a subset of AI, focuses on developing algorithms that enable systems to learn patterns from data and make decisions without being explicitly programmed. In the context of speech enhancement, ML techniques have enabled the development of specialised neural network architectures capable of recovering clean speech from noisy inputs. One widely used architecture is the *autoencoder*, which learns to reconstruct its input through a compressed latent representation [5].

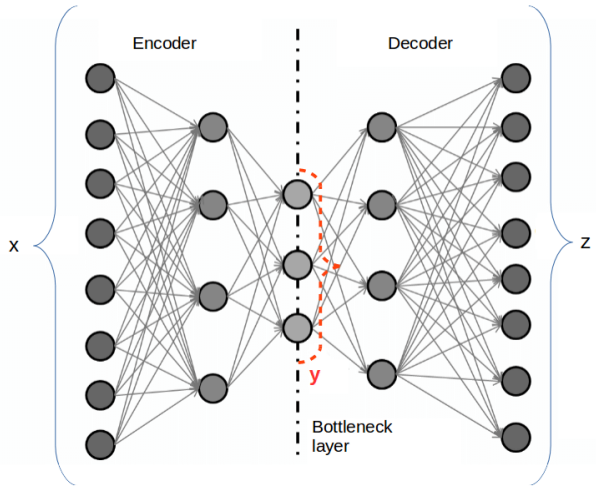


Figure 2.4 Block diagram of an autoencoder architecture [6].

An autoencoder consists of two main components: an encoder and a decoder. The encoder compresses the input signal x into a low-dimensional latent space using a series of convolutional, pooling, or fully connected layers. A central *bottleneck* layer enforces this compression, encouraging the model to retain only the most salient features (speech structure) while discarding irrelevant noise.

The decoder reconstructs the signal y using upsampling or mirrored layers. The objective is to recover the clean speech signal with minimal reconstruction error. Autoencoders are trained using paired datasets of noisy and clean speech. During training, the noisy input is passed through the network, and the output is compared against the clean reference. The model is optimised to minimise the difference, typically using a loss function such as MSE or Mean Absolute Error (MAE).

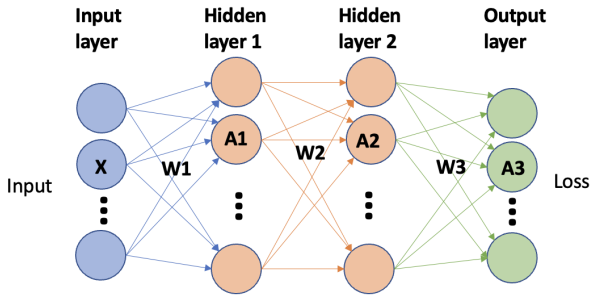


Figure 2.5 Loss and weight update process in an autoencoder [7].

Training proceeds iteratively. A forward pass generates an output, and the loss function quantifies the error. This error is then propagated backward through the network via backpropagation to compute gradients, which are used to update the model's weights using optimisers such as Stochastic Gradient Descent (SGD) or Adam. This process is repeated across many epochs until convergence.

* Are denoising autoencoders relevant to your work?
(Just curios, not a suggestion)

2.5 Evaluation Metrics

To evaluate the performance of noise reduction algorithms, several objective metrics are commonly used. These metrics provide quantitative measures of the quality and intelligibility of enhanced speech signals when compared to their clean references. The following subsections describe the most widely adopted metrics in speech enhancement research, all of which are utilised in this project.

2.5.1 Signal-to-Noise Ratio

The SNR is a fundamental metric that quantifies the relative strength of the desired speech signal compared to the background noise. A higher SNR indicates better separation between speech and noise components, which typically translates to improved intelligibility and perceived quality.

SNR is defined as:

$$\text{SNR} = 10 \log_{10} \left(\frac{P_s}{P_n} \right) \tag{2.5}$$

where P_s is the power of the clean speech signal and P_n is the power of the noise (or error) component. In practice, SNR can be computed in both time and frequency domains and is commonly used as a baseline performance indicator for denoising systems.

2.5.2 Mean Squared Error

The MSE measures the average squared difference between the enhanced and reference clean signals. It reflects the overall fidelity of the enhancement process but does not necessarily correlate with perceptual quality.

MSE is given by:

$$\text{MSE} = \frac{1}{N} \sum_{n=1}^{N} (x(n) - \hat{x}(n))^2 \tag{2.6}$$

where N is the number of signal samples, $x(n)$ is the clean signal, and $\hat{x}(n)$ is the enhanced signal. Lower MSE values indicate better approximation of the clean signal.

2.5.3 Perceptual Evaluation of Speech Quality

The Perceptual Evaluation of Speech Quality (PESQ) is an objective metric designed to estimate the perceptual quality of speech, taking into account human auditory

perception. Standardised [8], PESQ is widely used in speech enhancement and telecommunication systems.

PESQ simulates the auditory perception process and compares the clean and enhanced signals using psychoacoustic models. The PESQ score ranges from -0.5 to 4.5, with higher scores indicating better perceptual quality. A score above 3.0 is generally considered acceptable for most applications.

2.5.4 Short-Time Objective Intelligibility

citation

The Short-Time Objective Intelligibility (STOI) metric is designed to assess the intelligibility of speech, especially in noisy or distorted conditions. It works by comparing the short-time spectral envelopes of clean and enhanced speech segments.

The STOI score lies between 0 and 1, where higher values indicate better intelligibility. Scores above 0.5 typically suggest acceptable levels of speech understanding. STOI is particularly useful when intelligibility, rather than perceptual quality alone, is of primary concern.

2.5.5 Log Spectral Distance

citation

The Log-Spectral Distance (LSD) metric quantifies the spectral distortion introduced by a denoising algorithm. It measures the average distance between the logarithmic power spectra of the clean and enhanced signals across time and frequency.

LSD is defined as:

$$\text{LSD} = \frac{1}{F}\sum_{f=1}^{F}\sqrt{\frac{1}{T}\sum_{t=1}^{T}\left(\log S(f,t) - \log \hat{S}(f,t)\right)^2} \tag{2.7}$$

where $S(f,t)$ and $\hat{S}(f,t)$ are the clean and enhanced magnitude spectra at frequency bin f and time frame t, respectively. Lower LSD values indicate better preservation of spectral characteristics and less distortion.

Individually, these metrics fail to provide a complete and meaningful evaluation of the performance of noise reduction algorithms. For example, SNR and MSE are sensitive to the absolute power levels of the signals, while PESQ and STOI focus on perceptual aspects. Thus, multiple metrics are needed for a well rounded assessment.

2.6 Project Progression

The initial objective of this project was to enhance speech signals corrupted by noise using traditional DSP techniques. The original scope involved implementing classical noise reduction methods, such as SS and WF, with the intention of deploying them on an MSP432 microcontroller. These techniques were selected due to their low computational complexity and suitability for resource-constrained embedded systems.

However, early prototyping in Python revealed inherent limitations of classical DSP methods. Particularly problematic were their poor performance in non-stationary noise and limited adaptability to diverse real-world conditions. While methods like SS offer simplicity and efficiency, they often introduce musical noise artifacts and reduce intelligibility. Similarly, WF, though more rigorous, relies on assumptions about noise characteristics that may not hold in practice.

Meanwhile, data-driven approaches emerged as a compelling alternative. Real-time systems such as *Krisp*, *NVIDIA RTX Voice*, and *RNNoise* demonstrate how ML-based models can robustly suppress noise by learning complex nonlinear mappings from large-scale clean/noisy speech datasets. Recognising these advantages, the project evolved from solely implementing classical methods to exploring the deployment of lightweight ML-based speech enhancement models on more capable embedded hardware platforms. This shift aligned the project with modern trends while maintaining a focus on real-time deployment. It also enabled analysis of trade-offs between classical and data-driven methods in terms of performance, memory, and computation.

The revised project goal became benchmarking ML-based noise suppression against classical DSP methods using objective metrics. The project title was thus updated from *DSP Based Noise Cancellation System* to *Machine Learning Noise Cancellation System* to reflect this shift in focus.

3 Literature Review

This chapter reviews the key literature that directly informed the design and implementation of the project. While many aspects of the system drew on established methods and architectures, the following discussion focuses on the most influential and relevant papers. Three core areas are covered: handling variable-length sequences in datasets, the application of Fully Convolutional Network (FCN)s for speech enhancement, and the Conv-TasNet architecture for end-to-end speech separation.

3.1 Distortion Free Variable Length Handling

A significant challenge addressed in Section 5.1 is the handling of variable-length audio sequences within the dataset. To identify suitable strategies, a review of relevant research was conducted, with the method proposed by Yoon and Yu [9] emerging as particularly applicable.

Their work focuses on minimizing the information distortion introduced when sequences of varying lengths are batched for training. They explain the shortcomings of traditional padding and truncation methods, as well as more advanced implementations such as `PackedSequence` in PyTorch. While these methods enable batch processing, they inevitably introduce either loss of important information (via truncation) or addition of irrelevant information (via padding).

To address this, Yoon and Yu proposed the PTO method, a distortion-free technique that preserves the original sequence structure. In PTO, sequences are padded within a batch to match the longest input, but after processing, outputs are truncated back to their respective original lengths.

Experimental evaluation on the RSR2015 speaker verification dataset demonstrated that the PTO method consistently outperformed conventional approaches in terms of Equal Error Rate (EER), achieving up to a 27% relative improvement over baseline methods, while incurring only a modest computational overhead.

Given its distortion-free nature and suitability for real-time speech enhancement, PTO was adopted in this project as one of the three primary methods for handling variable-length sequences. It is further discussed in Section 6.1.3 and evaluated in Section 7.1.

3.2 Fully Convolutional Neural Network Adaptations

When considering the model architectures for the speech enhancement system, extensive research was conducted into models specifically designed for denoising tasks. Beyond standard CNN and UNet architectures already established in the field, the focus was placed on identifying designs that innovated.

One particularly relevant work is the study by Park and Lee [10], which proposed the use of FCNs for speech enhancement. Two architectures are displayed here, the CED and its improved variant, the R-CED, both specifically designed for the task of speech enhancement and operating directly on spectrogram data.

The CED model follows a symmetric encoder-decoder structure without an explicit bottleneck and is optimized for temporal feature extraction using frequency-preserving convolutional kernels. The real and imaginary components of the spectrogram are concatenated to form a two-channel input. The network compresses input features along the encoder and reconstructs them along the decoder, using pooling and upsampling operations to manage feature dimensions.

However, the paper notes a drawback of the CED architecture. Aggressive compression and upsampling operations can lead to information loss, particularly in the context of speech signals where temporal resolution is critical. To address this, the authors proposed the R-CED model. It removes pooling and upsampling layers entirely, instead employing redundant convolutional layers to augment feature representations. This design maintains full resolution throughout the network, enabling more effective denoising without sacrificing temporal fidelity.

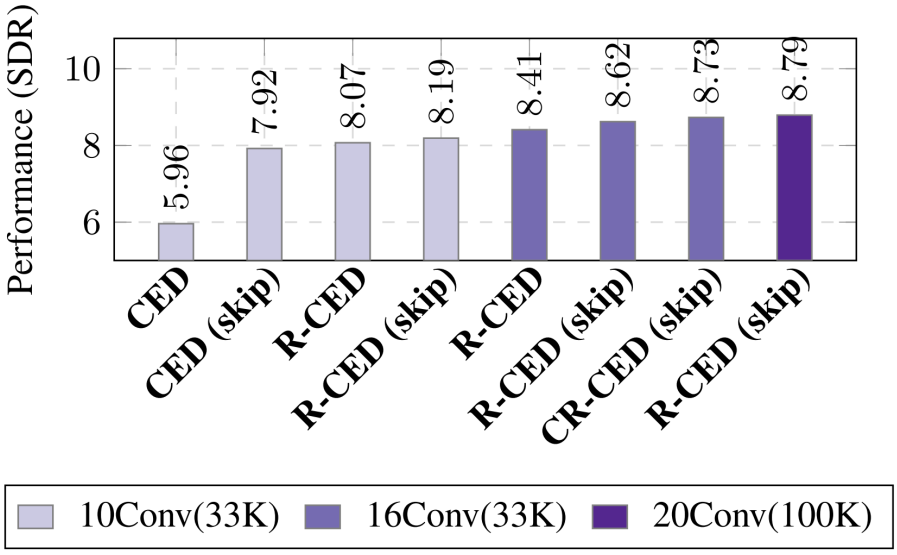


Figure 3.1 A Comparison of Denoising Performance and the Model Size for different CNN Architectures [10].

The results of the study demonstrated that R-CED consistently outperformed the conventional CED when evaluated with 10 convolutional layers and an equal parameter budget. Furthermore, performance improvements were observed by incorporating skip connections between encoder and decoder layers, which allow low-level feature information to be preserved and passed through the network. As shown in Figure 3.1, the addition of skip connections further enhanced the denoising capability of both the CED and R-CED architectures, with the R-CED (skip) configuration achieving the highest overall performance among the tested models.

3.3 Conv-TasNet for End-to-End Speech Separation

fix citation

Conv-TasNet is a convolutional neural network architecture proposed by Luo and Mesgarani [**luo2019conv**], representing a significant advancement in the field of speech enhancement. Unlike traditional approaches that operate in the time-frequency domain, Conv-TasNet processes raw audio waveforms directly using a Temporal Convolutional Network (TCN). The TCN replaces recurrent layers by employing stacked dilated convolutions and residual connections, enabling efficient modelling of long-range temporal dependencies with fewer layers.

Each TCN block consists of a depthwise separable convolution, non-linear activation Parametric Rectified Linear Unit (PReLU), normalisation (group normalisation), and a residual connection. Dilation factors increase exponentially across layers (e.g., 1, 2, 4, 8), allowing the model to expand its receptive field efficiently while maintaining computational stability. By stacking multiple TCN blocks, Conv-TasNet captures broad temporal contexts without the sequential limitations of recurrent models.

Conv-TasNet achieved state-of-the-art performance in speech separation tasks, surpassing conventional magnitude masking methods. The paper demonstrated the model's ability to separate multiple overlapping speakers from a single-channel recording under challenging conditions, highlighting the effectiveness of its end-to-end training approach in learning optimal source representations.

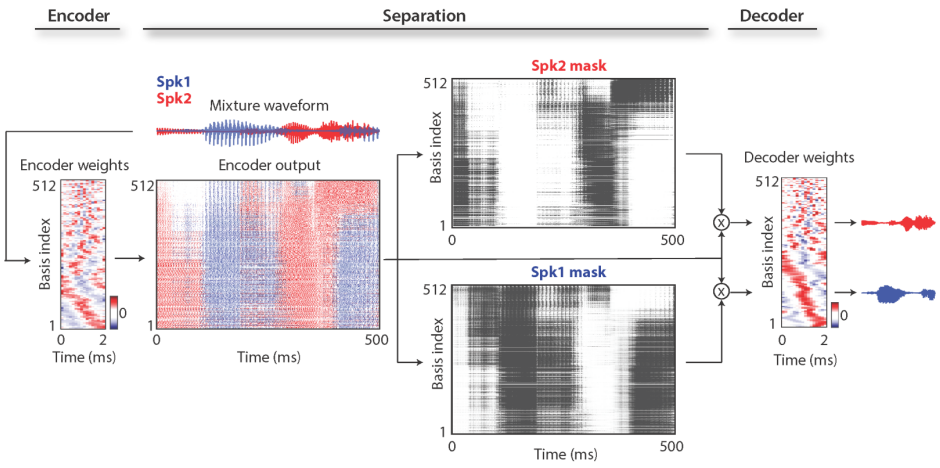


Figure 3.2 Source masks for a sample two-speaker mixture [**luo2019conv**].

In this project, the Conv-TasNet framework is adapted to operate on complex-valued spectrogram inputs for the task of speech denoising, while preserving the TCN structure for modelling long-range dependencies.

Refer to Figure 3.2 in text

4 Specification

This chapter outlines the core setup for the project. It begins by examining the dataset's structure and characteristics, which influence how data is preprocessed and formatted for input to the model. This is followed by a summary of the system-level requirements, including the development environment, training hardware, and memory constraints. Together, these define the context for the design and implementation decisions presented in subsequent chapters.

4.1 Dataset Exploration

A key aspect of defining the system specifications is understanding the dataset used. This project employs an open-source dataset titled *"Noisy speech database for training speech enhancement algorithms and TTS models"*, made available by the University of Edinburgh through its *DataShare* repository [11]. The dataset is released under the Creative Commons Attribution 4.0 International License [12].

`https://datashare.ed.ac.uk/handle/10283/2791`

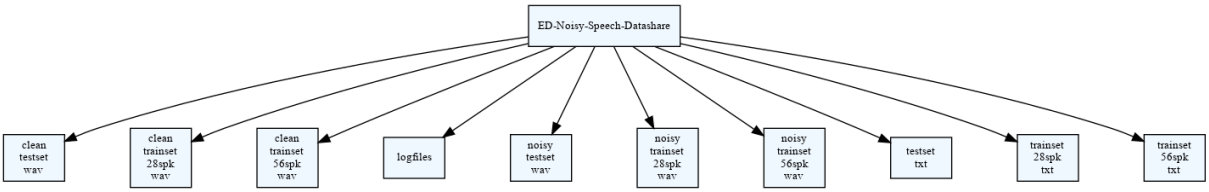
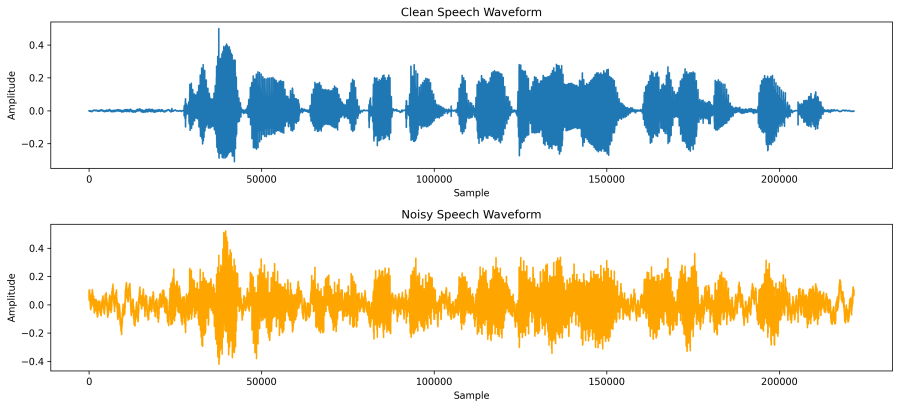


Figure 4.1 Dataset Tree Structure

As shown in Figure 4.1, the dataset is organized into three main subsets: a test set, a 28-speaker training set, and a 56-speaker training set. Each subset includes folders for transcripts, clean speech, and noisy speech. Files are aligned by index across these folders, ensuring parallelism between clean and noisy samples. However, the file numbering is not continuous, with gaps in the sequence.



Transcript: "To the Hebrews it was a token that there would be no more universal floods."

Figure 4.2 Example of a Clean, Noisy Speech Sample and its Transcript

All audio files are in `.wav` format and sampled at 48 kHz. Clip lengths vary, which significantly impacts how data is batched and fed into the model. The dataset contains no corrupted files, and the total download size was approximately 15 GB.

4.2 System Requirements

Given the computational demands of training neural networks, this project was developed and executed remotely on the University of Malta's Interfacing Lab servers, accessed via Secure Shell (SSH). These machines are equipped with NVIDIA GeForce RTX 3060 GPUs, which are suitable for deep learning workloads. However, GPU memory limitations required careful control of training parameters, including batch size, model depth, and the number of epochs.

The development environment was based on Visual Studio Code (VSCode), with assistance from GitHub Copilot for improved coding efficiency. Early experiments and module testing were conducted in Jupyter notebooks, which were later consolidated into a single modular Python application. This modular structure allows for separation of data preprocessing, model training, evaluation, and inference, improving maintainability and extensibility.

The final output of the system is a trained model checkpoint, which can be used for inference on new audio data. The model is compatible with industry-standard frameworks such as TensorFlow and PyTorch and is designed to be portable across machines. This enables future deployment on embedded boards or other constrained platforms for real-time speech enhancement.

5 Design

This chapter outlines the core design components that form the foundation of the speech enhancement system. It begins by addressing the challenge of handling variable length audio inputs, essential for batch-based model training. The latter and more substantial section focuses on the design of the ML architectures used in this project. Detailing each model's structure and the reasoning behind their implementation for speech enhancement.

5.1 Variable Length Handling

For this project, only the clean and noisy pairs of audio files from the dataset are required. The transcript text files are ignored, as they are not relevant to the task. However, such transcripts are valuable in tasks like speech recognition or text-to-speech. As highlighted in the dataset analysis in Section 4.1, the audio files vary in length. This poses a challenge for model training, as batch processing requires input tensors to have consistent dimensions.

To address this, several algorithms for handling variable-length audio inputs were explored. The most basic approach involves padding each audio file to match the length of the longest sample in the batch, typically by appending zeros to the end of shorter files. While this method is simple, it has significant drawbacks. Excessive padding introduces unnecessary data that may act as noise during training, making it harder for the model to learn effectively. The greater the variation in input lengths, the more padding is required, which can negatively impact overall training performance.

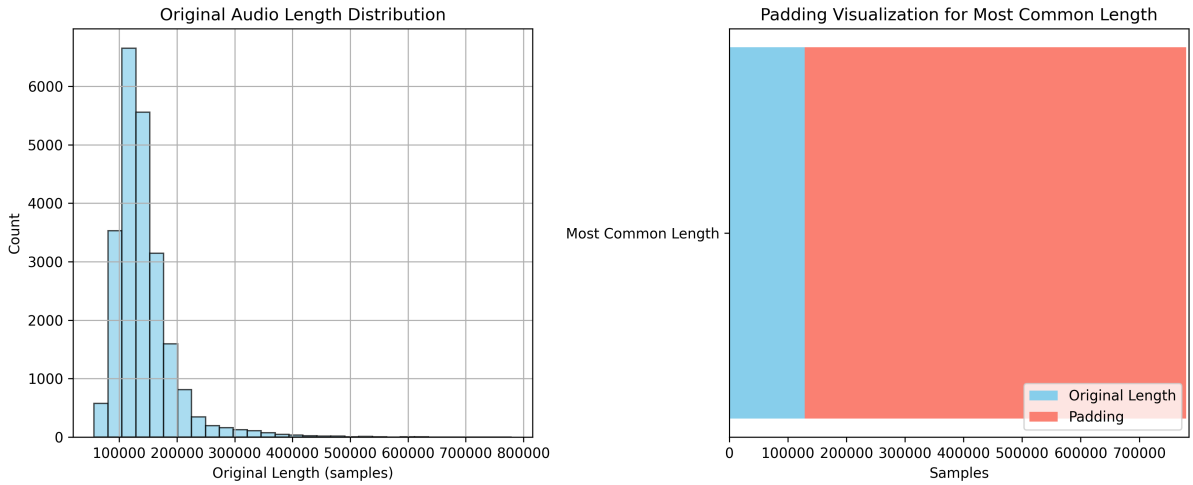


Figure 5.1 Illustration of maximum-length padding.

As shown in Figure 5.1, the most common audio length is padded so heavily

that the padding exceeds the actual content. This is far from ideal. To mitigate this issue, three different padding strategies were used, each aimed at reducing excessive padding.

The first method is *Static Bucketing*, which essentially groups audio files into predefined fixed-length buckets, allowing the grouped files to require less unnecessary padding. The second method, *Dynamic Bucketing*, builds on this by creating buckets dynamically based on the distribution of audio lengths, offering a more adaptive grouping approach. The third and final method introduced in Section 3.1. PTO is a more sophisticated approach that combines padding and truncating to ensure that the minimum distortion is introduced to the data.

All three methods were implemented for testing and evaluation. Their role in the system design is critical, as they help ensure that the model can learn effectively without being hindered by dimensional mismatches or excessive zero-padding. Further details on their implementations are provided in Chapter 6, and their impact on model performance is discussed in Chapter 7.

5.2 Model Architecture

The model architecture is central to system design, defining how inputs are processed, features transformed, and outputs reconstructed. The project's modular structure supports exploring various neural networks, starting from a simple baseline and extending to more advanced models. As introduced in Section 2.4, all models follow the autoencoder paradigm and operate on spectrograms, with real and imaginary components concatenated into a two-channel input and producing a similar two-channel output.

Initially, a `Tanh` activation was used at the output layer to constrain values within $[-1, 1]$, aiming for numerical stability during inverse STFT. However, evaluations showed this suppressed amplitude dynamics and degraded denoising performance. The `Tanh` was therefore removed from all model outputs—an improvement detailed in Appendix B.

5.2.1 Convolutional Neural Network

The CNN model implemented in this project serves as the baseline architecture. Its design follows a widely used encoder-decoder structure. The model operates on concatenated real and imaginary components of the spectrogram, processed as a two-channel input. Its straightforward structure makes it suitable for benchmarking and for validating the core pipeline before exploring more advanced architectures.

The first entry point in the forward pass of the model is the encoder. The encoder consists of three convolutional layers, each followed by a PReLU activation function. Each convolution uses a 3×3 kernel. The network begins with 2 input channels and progressively increases to 128 channels. The encoder's role is to reduce the spatial dimensions of the input while increasing the number of feature channels.

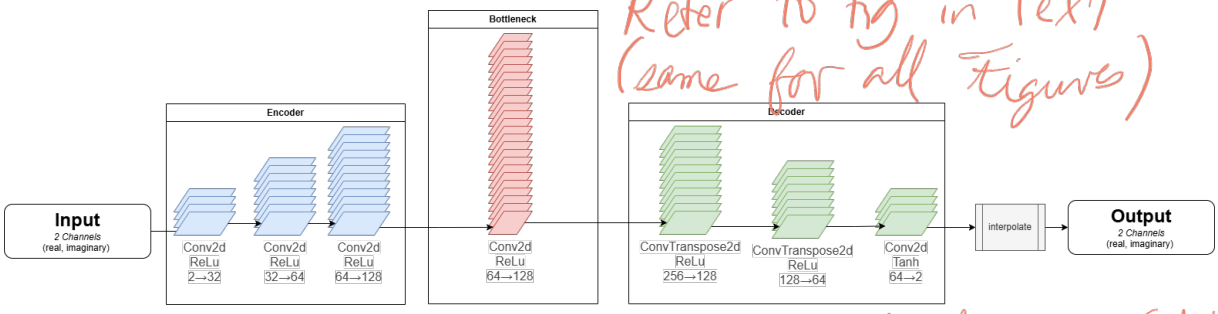


Figure 5.2 Basic CNN architecture.

The output of the encoder is passed to a bottleneck layer, which further transforms the feature space without altering its spatial resolution. In this implementation, the bottleneck consists of a single 3×3 convolutional layer that increases the number of channels from 128 to 256, followed by a Rectified Linear Unit (ReLU) activation. This design deepens the network and allows for more complex feature extraction. Since the bottleneck is not part of the downsampling or upsampling path, it also encourages the decoder to learn a more challenging mapping from latent space back to the input resolution.

The decoder reconstructs the spectrogram from the transformed features using a three-stage transposed convolutional block. It includes two transposed convolutional layers that upsample the spatial dimensions, reversing the compression performed by the encoder. Each is followed by a ReLU activation. A final standard 3×3 convolutional layer reduces the channel count from 64 to 2, corresponding to the real and imaginary parts of the output spectrogram. To ensure output consistency, bilinear interpolation is applied to match the original input resolution prior to splitting the output into its two channels.

While simple, this model plays a critical role in establishing a baseline performance level. It validates system functionality and serves as a reference for later architectures. The CNN model is straightforward to implement and interpret, making it a suitable starting point for benchmarking and guiding the development of more sophisticated architectures introduced in subsequent sections.

5.2.2 Convolutional Encoder Decoder

The CED architecture implemented in this project is based on the design proposed by Park and Lee [10], as reviewed in Section 3.2. The model follows a symmetric encoder-decoder structure, operating on complex spectrograms represented as two-channel inputs (real and imaginary concatenated along the channel dimension).

The encoder is composed of five convolutional blocks. Each block consists of a convolutional layer with a tall vertical kernel, batch normalisation, a ReLU activation function, and a 2×1 max pooling operation. The vertical kernel sizes decrease from 13×1 to 5×1 across layers, preserving frequency resolution while downsampling the temporal dimension. The number of channels increases progressively from 12 to 32, enabling richer feature representations as the input is compressed.

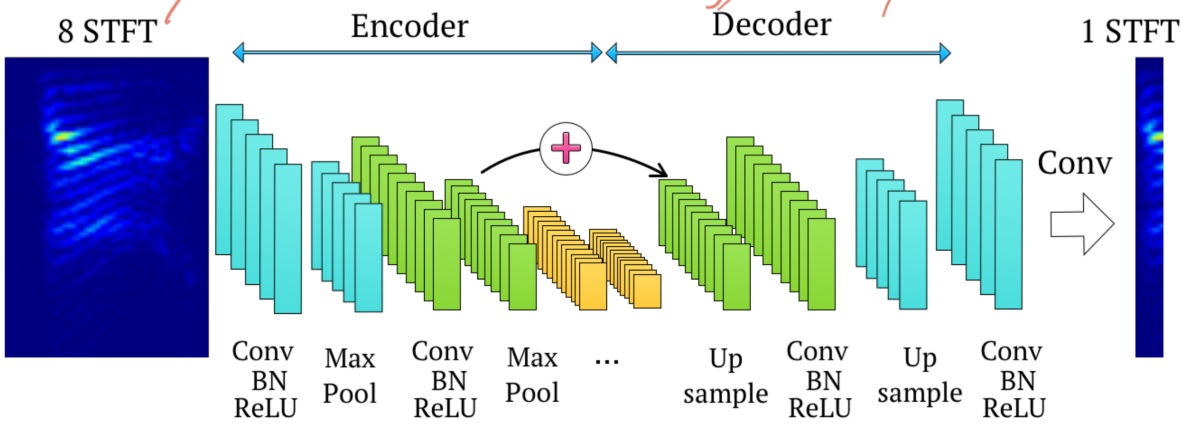


Figure 5.3 CED Network [10].

Unlike the CNN baseline, which uses a dedicated bottleneck for feature transformation, the CED model omits this step. Its encoder and decoder are fully connected through a continuous intermediate representation, avoiding abrupt compression. This enables smoother feature transitions across the network. Beneficial for speech signals where such disruptions can degrade temporal patterns.

The decoder mirrors the encoder in structure, using four stages of upsampling followed by convolution, batch normalisation, and ReLU activation. Each upsampling layer doubles the temporal dimension using a scale factor of $(2, 1)$, restoring the resolution reduced in the encoder. The decoder reverses the channel progression, reducing from 32 back down to 12. A final convolutional layer with a large vertical kernel of 129×1 is used to project the output to two channels, corresponding to the real and imaginary parts of the denoised spectrogram. A bilinear interpolation step is applied at the end to ensure the output matches the original input resolution.

By eliminating the bottleneck and employing deep, temporally aware

convolutional transformations, the CED model enables a smooth and information preserving mapping from noisy to clean spectrograms. Its symmetric structure, combined with frequency preserving vertical kernels and temporal downsampling, makes it well suited for capturing sequential dependencies without sacrificing spectral resolution. In this project, the CED architecture is used as a strong benchmark for assessing the effectiveness of FCN encoder-decoder, particularly in contrast to the more compressed representations used in CNN-based designs.

5.2.3 Redundant Convolutional Encoder Decoder

The R-CED architecture implemented in this project builds upon the CED framework introduced by Park and Lee [10] and discussed in Section 3.2. Like the CED, R-CED operates on complex spectrograms represented as two-channel inputs (real and imaginary). However, the design omits all pooling and upsampling operations to preserve full temporal and spectral resolution throughout the network.

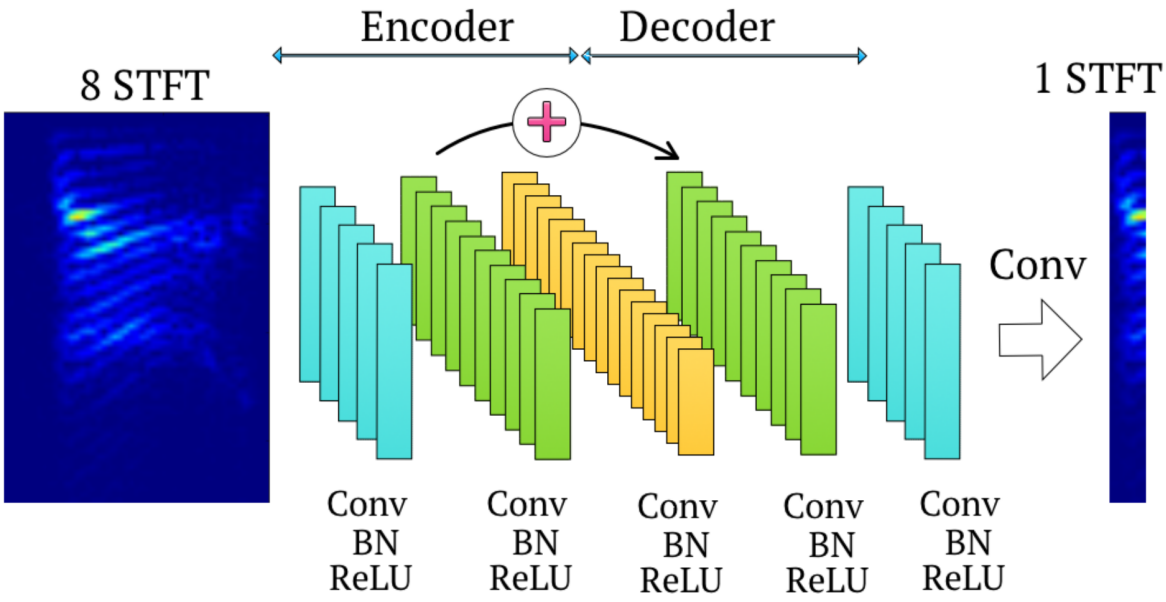


Figure 5.4 R-CED architecture [10].

The network is composed of nine sequential convolutional layers. The first eight layers use tall vertical kernels ranging from 13×1 to 7×1, symmetrically arranged around the network's midpoint. Each layer is followed by a ReLU activation and batch normalisation. The number of channels increases from 2 to 32 in the first half and then mirrors back to 12 in the second half, following the filter progression: 12, 16, 20, 24, 32, 24, 20, 16, 12. This symmetric arrangement enables the model to gradually build and refine feature representations without disrupting the resolution of the input signal.

Unlike typical encoder-decoder models, the CED maintains a constant spatial

resolution throughout the network. Instead of pooling or upsampling layers, representational depth is increased by stacking more convolutional layers. This redundancy increases the model's expressive capacity while preserving time-frequency alignment, crucial for speech enhancement.

The final convolutional layer uses a 129×1 kernel, consistent with the original paper, to project the output down to two channels representing the denoised real and imaginary components. Bilinear interpolation is applied at the end to match the input resolution before output splitting.

By eliminating dimensionality altering operations and relying on a deeper sequence of convolutional blocks, the CED model is able to maintain high resolution feature representations across all layers. This architecture is especially useful in applications where preserving the fine structure of spectrograms is essential. In this project, CED is evaluated as a resolution preserving alternative to encoder-decoder architectures.

5.2.4 U-Net

The UNet model implemented in this project is adapted from the widely known UNet architecture originally proposed for biomedical image segmentation [13]. In this work, it is repurposed for the task of speech enhancement using complex spectrograms. The model follows a symmetric encoder-decoder structure with skip connections and is designed to recover clean speech signals by operating directly on the concatenated real and imaginary spectrogram components.

The encoder consists of five convolutional blocks, each followed by instance normalisation and a PReLU activation. Downsampling is achieved by using a stride of 2 in all but the first convolution block. The channel depth increases progressively from 2 to 1024 across the five stages: $2 \rightarrow 64 \rightarrow 128 \rightarrow 256 \rightarrow 512 \rightarrow 1024$. Instance normalisation is preferred over batch normalisation for improved stability with variable-length or low-batch-size audio inputs.

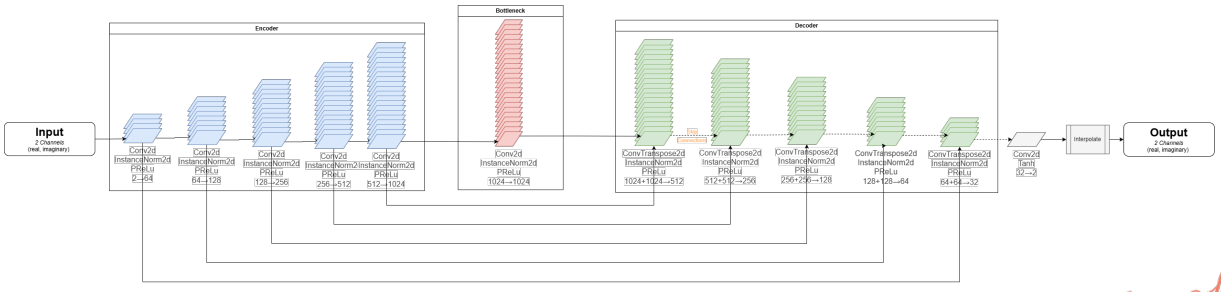


Figure 5.5 UNet Architecture.

At the centre of the network is a bottleneck block that maintains the 1024-channel depth while applying an additional convolution, instance normalisation,

and PReLU activation. Unlike traditional bottlenecks that compress the latent representation, this layer serves as a non-linear transformation hub that enhances the feature space before decoding.

The decoder mirrors the encoder with five transposed convolutional blocks that upsample feature maps and reduce channel depth through the sequence $1024 \rightarrow 512 \rightarrow 256 \rightarrow 128 \rightarrow 64 \rightarrow 32$. At each stage, features from the corresponding encoder block are concatenated with the upsampled output via skip connections. Since downsampling alters spatial resolution, bilinear interpolation is applied to align encoder features before concatenation and again after the final 3×3 convolution, which projects the output to 2 channels corresponding to the real and imaginary components of the denoised spectrogram.

This UNet implementation preserves the architectural strengths of the original model while tailoring its structure for speech enhancement. The use of skip connections, instance normalisation, and a deep encoder-decoder hierarchy enhances the model's ability to recover spectrogram-temporal structure, making it a powerful architecture for complex denoising tasks.

5.2.5 Conv-TasNet

The Conv-TasNet model implemented in this project is based on the architecture proposed by Luo and Mesgarani [**luo2019conv**], as discussed in Section 3.3. Although the core structure is preserved, several adaptations were made to align the model with the spectrogram-based input framework used throughout this system.

fix citation

The original Conv-TasNet operates directly on time-domain audio signals. In contrast, this implementation modifies the input format to handle complex-valued spectrograms. The real and imaginary components are concatenated along the channel dimension, forming a two-channel input that is passed through a 3×3 convolutional encoder. The encoder projects the input to 128 channels using dynamic padding to preserve resolution.

The encoded features are passed to a TCN, which serves as the core separation module. The TCN is composed of two stacks, each containing four residual blocks, for a total of eight layers. Each residual block consists of a dilated convolution with exponentially increasing dilation factors (e.g., 1, 2, 4, 8), followed by group normalisation (with eight groups), a PReLU activation, and a second projection convolution. A residual connection is used to stabilize training and preserve input information. The dilation strategy allows the TCN to model long-range temporal dependencies efficiently while maintaining a fixed receptive field size.

Unlike the original Conv-TasNet, which applies explicit time-domain masking for source separation, this implementation omits masking altogether. Instead, the TCN

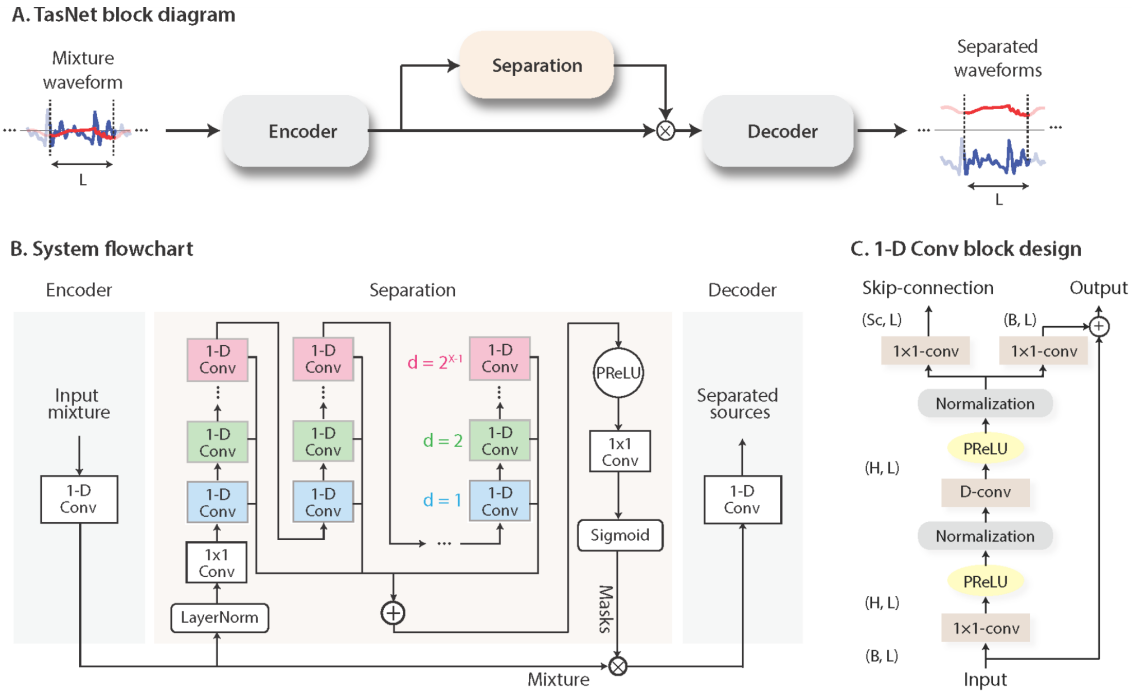


Figure 5.6 Conv-TasNet architecture overview [**luo2019conv**].

directly outputs an enhanced latent representation, which is then decoded into denoised real and imaginary spectrogram components via a final 3×3 convolutional layer. This simplifies the architecture, as masking yielded marginal gains.

The output of the decoder is aligned with the original input size by cropping along both frequency and time axes to correct for minor size mismatches introduced by convolutional operations. This ensures a consistent dimensionality before inverse STFT reconstruction.

By preserving the temporal modelling strength of the original TCN structure and adapting it to operate in the complex spectrogram domain, this implementation of Conv-TasNet offers a powerful yet flexible architecture for speech denoising. It combines deep receptive fields, efficient convolutional modelling, and skip connections within residual blocks to robustly recover clean speech across variable-length inputs.

This project explores several neural network architectures for speech enhancement, starting with a simple CNN-based autoencoder. It then advances to FCN models like CED and R-CED[]park2017acoustic, followed by UNet with skip connections and deeper encoding. Finally, Conv-TasNet [**luo2019conv**], originally designed for time-domain speech separation, is adapted for spectrogram-based denoising using its TCN. These models provide a diverse and comparative framework, highlighting trade-offs in design and establishing strong baselines against classical signal processing techniques.

6 Implementation

This chapter presents the key implementation details that enable the speech enhancement system to function effectively under practical constraints. It describes the core components developed from scratch. Focus is drawn to the variable-length handling in the custom dataset classes. As well as the Out-Of-Memory (OOM) handling strategies that were implemented to ensure robust and efficient training of deep models.

6.1 Datasets

Following Section 5.1, this section describes the dataset handling mechanisms implemented to accommodate variable-length audio signals. The custom dataset classes developed for this project inherit from PyTorch's `Dataset` class and are used in both training and denoising pipelines. These implementations are central to managing waveform padding, STFT conversion, and batch preparation for ML models.

Each class implements three essential methods that are all similarly structured but differ in their approach to handling variable-length data. The `__init__` method initializes key parameters, constructs paths for clean and noisy speech files, and prepares required caches. Here, the amount of parameters varies based on the approach as some methods require more configuration than others. The `__len__` method is a simple return function that provides the total number of samples in the dataset. The `__getitem__` method applies necessary transformations to the audio files such as mono conversion, resampling, and STFT. It also handles the padding and truncation of audio samples to ensure consistent input dimensions across batches. Real and imaginary STFT components are used as input-output pairs, ensuring a consistent format across all datasets. A critical motivation for this implementation is real-time deployment. The STFT inherently segments audio into short overlapping frames, simulating real-time input. This makes training models on STFT frames suitable for later real-time inference.

6.1.1 Static Bucketing

The first dataset implementation explored to address the variable-length issue and limitations of fixed-length padding is the static bucketing method. This approach uses a predefined set of bucket sizes, each corresponding to a fixed number of audio samples. In this project, sizes were derived by multiplying the sampling rate (48 kHz) by target durations (e.g., 1s, 2s, 3s), resulting in discrete, fixed-length buckets.

Each audio file is assigned to the first bucket that can fully contain it. For example, an audio clip of 2.5 seconds is assigned to the 3s bucket, since the 2s bucket is insufficient, and the 3s bucket is the next available option.

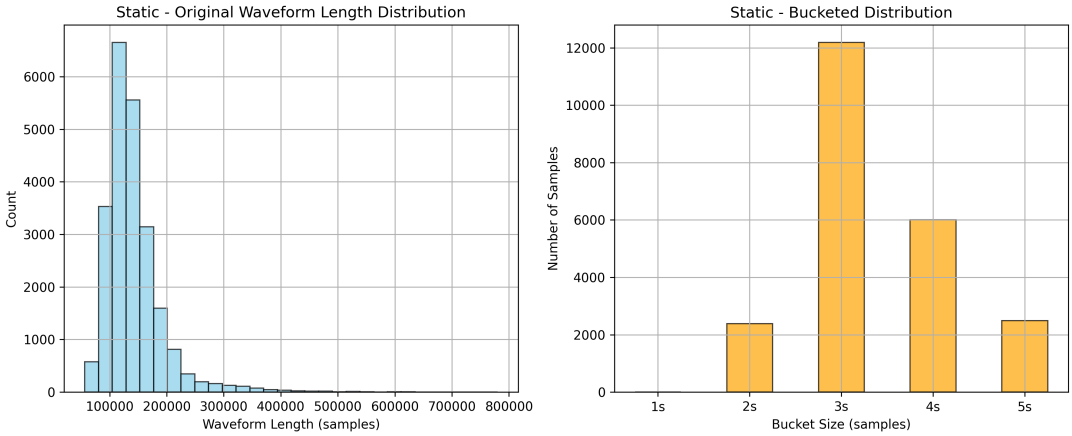


Figure 6.1 Static bucketing histogram and bucket allocation

The `bucket_handler` method iterates over all clean files, assigns them to buckets, and caches these assignments to avoid redundant processing in future runs. During training, a custom `collate` function pads or truncates audio waveforms to match their bucket's target length. This guarantees consistent input dimensions within each batch, which is essential for batch-based processing in neural networks.

While static bucketing does not eliminate padding altogether, it significantly reduces the amount of excess padding compared to a naive fixed-length strategy. As shown in Figure 6.1, most samples naturally group into the smaller buckets (3s), improving efficiency and reducing unnecessary computations.

6.1.2 Dynamic Bucketing

The dynamic bucketing method improves upon static bucketing by adapting the bucket sizes to the actual distribution of audio file lengths. Instead of predefined durations, the method uses K-Means clustering on the waveform lengths to compute optimal bucket centres. This enables buckets that reflect the natural variance in the dataset.

During initialization, the method measures the length of each clean file and applies K-Means with a specified number of clusters (`num_buckets`). These cluster centers become the bucket sizes, and each sample is assigned to the closest one. Like static bucketing, this assignment is cached.

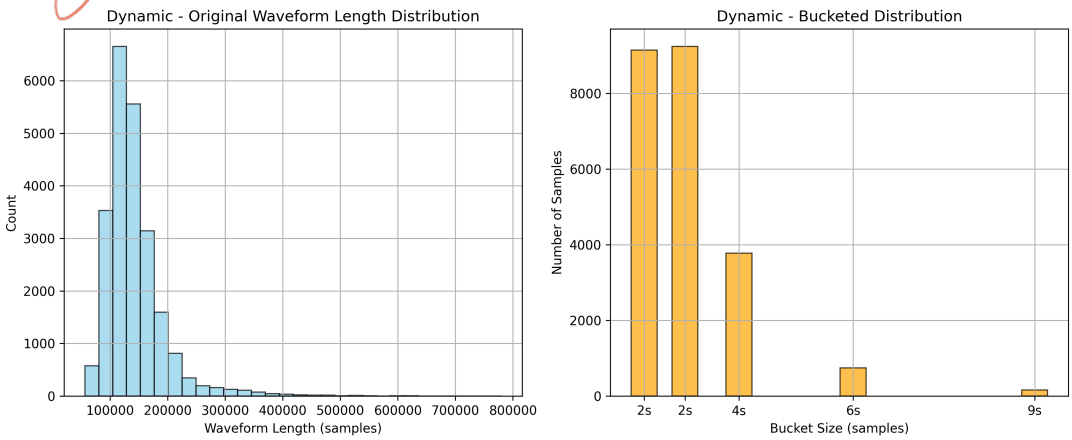


Figure 6.2 Dynamic bucketing histogram and bucket allocation

The `collate` method then pads or truncates each waveform to its bucket's target size. Compared to static bucketing, dynamic bucketing leads to tighter groupings, minimizing the average amount of padding per batch. This results in better memory utilization and more efficient training.

Dynamic bucketing combines flexibility with performance, adapting to the dataset rather than imposing fixed constraints. It is particularly effective when the data contains a wide or uneven distribution of sequence lengths. When comparing dynamic to static bucketing, we observe that the dynamic method identifies two buckets around the 2 second mark that are heavily populated. Clearly illustrating its ability to adapt to the underlying structure of the dataset more effectively than the fixed approach.

6.1.3 Padding-Truncation Output-Truncation

The PTO method, introduced by Yoon and Yu [9] and discussed in Section 3.1, was implemented in this project following the original proposal.

To facilitate the integration of PTO into the data pipeline, a custom `pto_collate` function was developed. This function aligns all spectrograms along the frequency axis using bilinear interpolation, then pads the time axis so that all samples in a batch match the length of the longest sample. Additionally, the original (pre-padding) lengths of each input are recorded and returned alongside the padded tensors.

During training, batches consist of spectrograms with uniform dimensions, while preserving the original input lengths for later use. After inference, model outputs are cropped back to their corresponding original lengths, ensuring that the final enhanced outputs remain distortion-free and aligned with the original input signals.

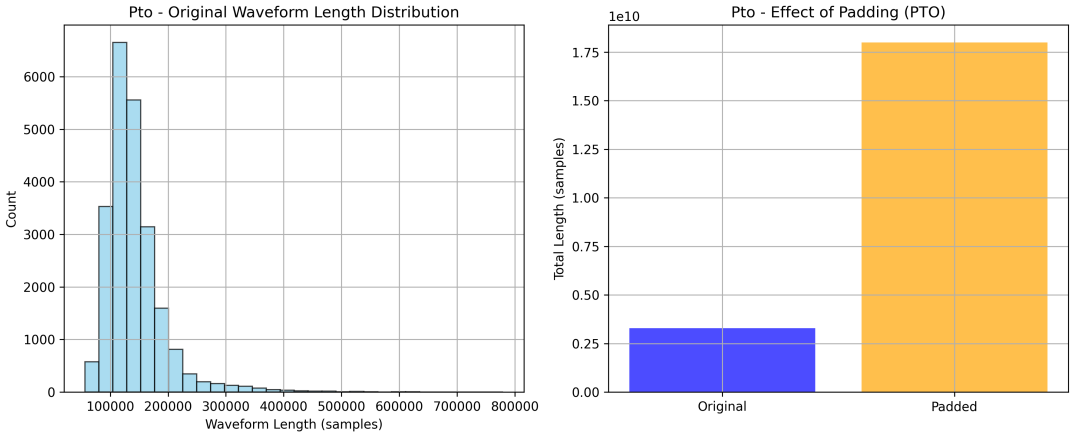


Figure 6.3 Padding effect using the PTO method.

The PTO method simulates real-time signal processing by operating on frame-aligned STFT inputs, making it highly suitable for deployment in low-latency scenarios. Unlike bucketing methods, it does not require predefined groups or clustering of sequence lengths, offering greater adaptability to varied datasets.

However, it introduces computational overhead, as padding is still necessary during training and output truncation adds an extra processing step at inference. This overhead was addressed by implementing additional handling within the training pipeline. Firstly, to correctly manage the extra PTO parameter containing the original (unpadded) lengths. Secondly, to perform output truncation as a post-processing step, cropping model outputs back to their original lengths after inference.

It is also noted that, although padding is mitigated post-inference, the model remains exposed to padded inputs during training, meaning model weights are still updated based on padded data.

6.1.4 Helper Functions

A few support functions are implemented to ensure that batching and visualization are handled effectively across dataset variants:

- `pto_collate`: A custom collate function used by the PTO dataset. It aligns all spectrograms along the frequency axis and pads them along the time axis, whilst retaining the original lengths for post-processing.
- `BucketSampler`: Used with static and dynamic bucketing. It ensures that batches contain only samples from the same bucket, preventing dimension mismatch during training. It randomizes batches within buckets to improve generalization.
- `visualize_dataset_padding`: A diagnostic utility that plots the distribution of

original lengths and the effects of padding for each method. It provides insights into the efficiency and overhead introduced by each strategy.

6.2 Out-Of-Memory Handling

OOM errors were encountered while training deeper model architectures (UNet and Conv-TasNet mainly), even with relatively small batch sizes (e.g., 4). These errors are common in GPU constrained environments due to the increased number of trainable parameters and the large size of spectrogram tensors derived from high-resolution audio. Figure 6.4 shows a representative OOM error message encountered during training.

```
torch.OutOfMemoryError: CUDA out of memory. Tried to allocate 776.00 MiB. GPU 0
has a total capacity of 7.79 GiB of which 39.75 MiB is free. Including
non-PyTorch memory, this process has 7.73 GiB memory in use. Of the allocated
memory 6.17 GiB is allocated by PyTorch, and 1.40 GiB is reserved by PyTorch
but unallocated. If reserved but unallocated memory is large try setting
PYTORCH_CUDA_ALLOC_CONF=expandable_segments:True to avoid fragmentation.  See
documentation for Memory Management  (https://pytorch.org/docs/stable/notes/
cuda.html#environment-variables)
```

Figure 6.4 PyTorch CUDA out-of-memory error message.

To address these limitations, several memory management strategies were implemented to stabilize training and enable deeper models to run within the available GPU resources. These methods also allowed the system to simulate larger batch sizes and achieve better convergence. The following techniques were used to mitigate OOM errors:

- **Expandable CUDA Segments:** Before initiating training, the environment variable `PYTORCH_CUDA_ALLOC_CONF=expandable_segments:True` is set. This enables PyTorch's new memory allocator with segment expansion, which helps alleviate fragmentation issues and allows the allocator to resize memory segments dynamically. This is the first suggestion from the PyTorch documentation for OOM errors.
- **Mixed Precision Training:** Automatic mixed precision (AMP) was used via PyTorch's `autocast()` and `GradScaler`. This allows intermediate tensors to be stored in FP16 format, reducing memory usage and increasing throughput, while maintaining model weights and gradient computations in FP32 for stability. Figure 6.5 illustrates the reduced bit-width and structure of FP16 compared to FP32, highlighting the potential for nearly half the memory usage in forward and

backward passes [14]. This approach significantly extends the trainable capacity of a model under constrained GPU environments.

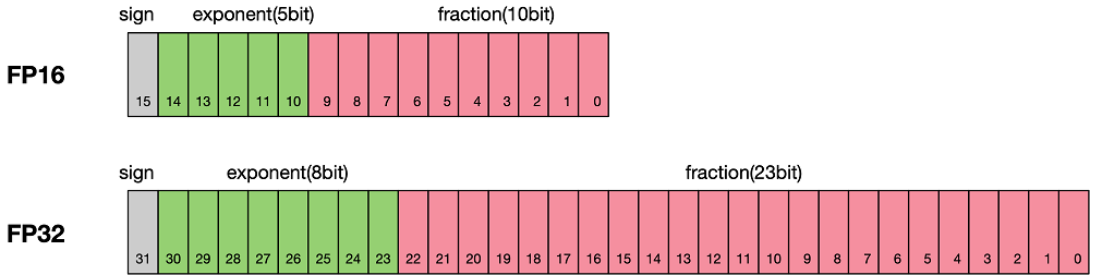


Figure 6.5 Comparison of memory allocation structure between FP16 and FP32 formats [14].

- **Gradient Accumulation:** Due to memory limitations that constrained the feasible batch size, gradient accumulation was employed to simulate a larger effective batch size. By accumulating gradients over multiple mini-batches and updating the model weights only every `accumulation_steps` iterations, the training process preserves the learning dynamics of a larger batch size without exceeding memory constraints. However, directly plotting the scaled loss can result in misleading visualizations, such as those shown in Figure 6.6. Where the logged values may appear artificially small or flat. To mitigate this, the raw (unscaled) loss is computed and stored separately before scaling. This ensures that both training and validation loss curves accurately reflect model performance trends, even when gradients are accumulated across steps.

Figure 6.6 Plot Result When Scaling Losses

- **Manual Memory Management:** To ensure memory is cleared between epochs and reduce fragmentation, memory cleanup routines are explicitly called at the start of every epoch: `gc.collect()`, `torch.cuda.empty_cache()`, and `torch.cuda.reset_peak_memory_stats()`. This manual intervention is particularly beneficial in a shared GPU setting, where residual allocations from other users or previous epochs may linger.
- **Gradient Clipping:** To prevent exploding gradients, which can rapidly consume memory, gradient norms were clipped using `torch.nn.utils.clip_grad_norm_()`. This maintains training stability and prevents sudden memory spikes during backpropagation, which is especially important when using high learning rates or unregularized architectures.
- **Learning Rate Scheduling:** An optional scheduler progressively lowers the learning rate using a step-decay approach. This helps reduce volatile updates later in training, smoothing out the optimization process and reducing the likelihood of erratic memory usage. The scheduler also saves on resources, especially in the cases when smaller models saturate at earlier epochs.
- **Memory Profiling:** At the end of each epoch, GPU memory statistics are logged using `torch.cuda.max_memory_allocated()` and `torch.cuda.max_memory_reserved()`. This aids in profiling memory behavior and catching potential leaks or inefficiencies across training epochs.

These combined methods provided the foundation for training deep models across all dataset variants with minimal interruptions. Even with GPU constraints, these optimizations enabled stable, high-performance training and evaluation of multiple architectures, supporting both experimentation and reproducibility in a limited computing environment.

7 Evaluation

This chapter presents a comprehensive evaluation of the implemented speech enhancement system, assessing both classical and deep learning approaches across several dimensions. The evaluation is divided into three key parts. First, the impact of dataset handling strategies is examined, comparing how the different dataset methods affect training efficiency and performance. Second, the effectiveness of OOM mitigation techniques is validated to ensure that memory-saving strategies do not degrade model quality. Finally, the core focus of this chapter is a comparative analysis of model architectures, benchmarking five ML models against three classical denoising methods.

Each section includes detailed quantitative assessments using respective metrics. The metrics within the first three sections are conducted in batch. That is the model is trained on the whole training set, the validation on the whole validation set, and the denoising on the whole test set. So any metrics are averaged across the whole dataset. Results are presented using clear, tabulated formats for ease of interpretation and cross-comparison.

7.1 Dataset Performance

This section examines the performance of the three dataset handling strategies: Static Bucketing, Dynamic Bucketing, and PTO. The goal is to assess how these strategies affect the overall efficiency of the training process, particularly in terms of dataset loading times, runtime overhead during training, and their influence on model performance. Each strategy was tested using the same model architecture, the CED, chosen for its simplicity and effectiveness in the spectrogram domain. Under two conditions:

- **Cold Run (Uncached):** In this scenario, all dataset operations are executed from scratch. Static and Dynamic Bucketing compute bucket assignments (with Dynamic Bucketing also requiring K-Means clustering), while PTO calculates and stores the original waveform lengths. This setup simulates a first-time deployment or training on a fresh system.
- **Warm Run (Cached):** This run utilises cached data generated during the cold run, significantly reducing load and pre-processing time. For Static and Dynamic Bucketing, bucket mappings and K-Means centresare reloaded. For PTO, the previously computed original sequence lengths are retrieved.

The configuration used for all CED runs is shown in Figure 7.1, with the only varying parameter being the `PAD_METHOD`.

```
# Dataset Parameters
DATASET_DIR = "../ED-Noisy-Speech-Datashare"  # Path to the dataset
SAMPLE_RATE = 48000  # Dataset sample rate
N_FFT = 1024  # Number of FFT points
HOP_LENGTH = 256  # Hop length for the STFT
BATCH_SIZE = 4  # Batch size for the model
ACCUMULATION_STEPS = 1  # Gradient accumulation steps
NUM_WORKERS = 4  # Number of workers for the DataLoader


# Padding Methods
PAD_METHOD = "dynamic" # OPTIONS: "dynamic", "static",  "pto"
VISUALIZE = False  # Visualize the padding method
NUM_BUCKET = 5  # Number of dynamic buckets (dynamic only)


# Model Parameters
MODEL = "CED"  # OPTIONS: "CNN", "CED", "RCED", "UNet", "ConvTasNet"
EPOCHS = 10  # Number of epochs to train the model
LEARNING_RATE = 1e-3  # Learning rate for the model
```

Figure 7.1 Configuration of the CED model used for dataset performance evaluation.

Both uncached and cached runs were executed, and the relevant timing metrics were collected from the output logs, as shown in Table 7.1.

Table 7.1 Dataset Training Overheads

Dataset	Uncached	Cached	Truncation Overhead
Static Bucketing	296.85s	0.83s	N/A
Dynamic Bucketing	581.57s	0.83s	N/A
PTO	288.47s	0.79s	68.62s

The results in Table 7.1 reveal key insights into the efficiency of each dataset handling strategy. In the uncached condition, both Static Bucketing and PTO exhibit similar loading times, as each only requires a single pass through the dataset. Either to assign buckets or compute original waveform lengths. Dynamic Bucketing, however, incurs nearly double the loading time due to the additional K-Means clustering step, which requires a second pass to calculate cluster centres for optimal bucket assignment. This overhead is particularly pronounced in the uncached scenario, where the dataset must be fully loaded and processed from scratch.

In contrast, the cached runs significantly reduce loading times across all methods. Once the dataset metadata has been computed and stored, subsequent runs

simply reload the cached mappings or original lengths, avoiding repeated preprocessing. PTO achieves the fastest cached time, as retrieving a list of original lengths is marginally quicker than reloading bucket assignments or cluster centres. However, a key distinction of PTO is its requirement for per-epoch output truncation during training. While this truncation overhead is relatively minor compared to the total training duration (typically on the order of hours). It can accumulate significantly across large datasets or extended training schedules. This trade-off must be carefully considered when assessing PTO's suitability for time-sensitive or resource-constrained training environments.

To evaluate the impact of dataset handling strategies on model learning, each method was tested using the best-performing checkpoint from its respective training run. Since the differences in performance between cached and uncached conditions were found to be trivial, the results reported in Table 7.2 reflect the mean values across runs, with corresponding margins of error. This provides a clearer picture of any underlying variation while confirming the consistency of results.

Table 7.2 Dataset Handling Strategies Training Metrics

Dataset	Train Loss	Val Loss	Val SNR
Static Bucketing	0.1805 ± 0.0013	0.1787 ± 0.0020	0.54 ± 0.045 dB
Dynamic Bucketing	0.1779 ± 0.0004	0.1850 ± 0.0063	0.56 ± 0.010 dB
PTO	0.1414 ± 0.0019	0.1458 ± 0.0033	0.53 ± 0.050 dB

The results in Table 7.2 show that while all three dataset handling strategies yield reasonable and consistent validation SNR values, there are notable differences in training and validation loss. In particular, the PTO method achieves lower training and validation loss compared to both Static and Dynamic Bucketing. This suggests that PTO facilitates more learning under the current loss formulation.

However, despite its lower loss, PTO does not yield a corresponding improvement in validation SNR. The SNR values across all methods remain within a narrow range, indicating that the perceptual or energy-based denoising quality is comparable. The improved loss values in PTO may be partly influenced by its output truncation mechanism. This evaluates only the central, non-padded regions of the spectrogram. Potentially leading to more favourable loss estimates even if the actual perceptual improvement is marginal.

Considering the trade-offs between implementation complexity, runtime overhead, and generality. Dynamic Bucketing remains the most balanced choice. It handles variable-length inputs with optimal bukceting and minimal padding. Avoids the per-epoch truncation overhead of PTO and maintains reliable performance across all

metrics. As such, Dynamic Bucketing is selected as the preferred dataset handling strategy for all subsequent model evaluations in this project.

7.2 OOM Validation

While Section 6.2 outlined several techniques to mitigate OOM errors during training, it is important to demonstrate that these strategies do not compromise model learning. The goal of this section is to validate that memory-saving methods do not lead to information loss or degraded performance. To assess this, the R-CED model was used, since the UNet and Conv-TasNet could not be trained without OOM handling techniques. Training was conducted using the dynamic bucketing strategy and compared across three different configurations:

1. **Clean Training (Control):** A standard training loop without any OOM handling logic, using a batch size of 4. This configuration serves as the control, with no memory management mechanisms.
2. **OOM Handling (Batch 4, Accum 1):** OOM handling techniques were enabled while maintaining a batch size of 4. These included FP16 and Garbage Collection (GC), allowing for a more memory-efficient training process.
3. **OOM + Accumulation (Batch 2, Accum 2):** The batch size was reduced to 2, with gradient accumulation set to 2, simulating an effective batch size of 4. All OOM handling techniques remained enabled. This configuration is designed to reduce memory usage while preserving gradient stability.

Each configuration was trained for the same number of epochs, using identical learning rates and optimizers. Table 7.3 summarizes the training and validation performance.

Table 7.3 OOM Configurations Training Metrics

Train Config	Train Loss	Val Loss	Val SNR	Training Time
Control	0.1169	0.1185	3.20 dB	4.44 h
OOM Handling	0.1217	0.1255	2.74 dB	3.08 h
OOM + Accumulation	0.1184	0.1263	2.95 dB	3.91 h

The results in Table 7.3 show that all three training configurations achieve comparable performance in terms of both loss and validation SNR. The control configuration, which does not use any OOM handling mechanisms, achieves the best

validation SNR of 3.20 dB. With the differences across configurations being within a 0.5 dB range. Suggesting that the inclusion of OOM handling strategies may introduce some degradation of the models training performance. The OOM Handling configurations, use of FP16 and GC could be seen as a trade-off between memory efficiency and model performance.

Each configuration was further evaluated in the full denoising pipeline. Table 7.4 presents the performance metrics across key evaluation criteria.

Table 7.4 OOM Configurations Denoising Metrics

Train Config	**↑SNR**	**↓MSE**	**↑PESQ**	**↑STOI**	**↓LSD**
Control	14.2843	0.000120	2.1096	0.8730	0.702311
OOM Handling	14.7731	0.000114	2.1791	0.8783	0.691936
OOM + Accumulation	14.8076	0.000117	2.1008	0.8748	0.692152

The results in Table 7.4 reveal that, contrary to initial expectations, the OOM handling configurations not only maintain performance but slightly outperform the control in several key denoising metrics. Both the **OOM Handling** and **OOM + Accumulation** setups show improvements in SNR, PESQ, and LSD, suggesting enhanced perceptual quality and spectral fidelity.

Specifically, **OOM + Accumulation** achieves the highest SNR at 14.81 dB, marginally outperforming both the control and the non-accumulated OOM variant. Additionally, the lowest LSD values are observed in the OOM handled models, indicating that their outputs more closely preserve the spectral characteristics of the clean reference signals. These findings suggest that the use of FP16 and memory aware strategies does not harm. Potentially even improving the model's ability to generalize.

The marginal gains in PESQ and STOI further support this conclusion. Although the differences are subtle, they point to a stable perceptual consistency across configurations. It is also possible that reduced numerical precision in FP16 introduces a form of implicit regularization, slightly mitigating overfitting and improving generalization performance.

In summary, while the training metrics in Table 7.3 showed modest differences across configurations. The downstream denoising evaluation shows no degradation and in some cases minor improvements. These findings fully justify the use of OOM handling techniques throughout this project. Even if future configurations exhibited performance regressions due to OOM mitigation. Their use would remain essential for enabling the training of memory intensive models such as *UNet* and *Conv-TasNet*, ensuring a fair architectural comparison across all model types evaluated in this work.

7.3 Model Performance

This section presents the most critical part of the evaluation and the central focus of the project. A comparative assessment of classical methods and five ML models for speech enhancement. Unlike earlier evaluations, which focused on dataset handling strategies and OOM mitigation techniques using fixed models to conduct the justification. The main scope of this project is to justify the use of ML models for speech enhancement and to evaluate their performance against classical methods.

The evaluation retains the previously established dynamic bucketing and OOM handling strategies to ensure consistency. To provide a meaningful frame of reference, a baseline set of metrics is also established in Section 7.3.1. This baseline follows the same evaluation pipeline as the classical methods but uses the raw noisy inputs directly compared against the clean reference signals. These values define a lower bound for performance and serve as a clear reference point for assessing the effectiveness of each denoising method.

7.3.1 Classical Methods

The classical methods evaluated include SS, WF, and the MMSE-LSA.These are all implemented in a single-channel setting. SS and WF are foundational methods in literature, whilst MMSE-LSA represents a more developed and perceptually motivated approach. Since these methods do not require training, the classical evaluation focuses solely on denoising performance using the same pipeline and evaluation metrics as the learning based models.

Table 7.5 Classical Denoised Metrics

Method	↑SNR	↓MSE	↑PESQ	↑STOI	↓LSD	Denoise Time
Baseline	-2.2774	0.005152	1.8451	0.8928	0.904223	56s
SS	3.0861	0.001525	1.4535	0.8457	0.767089	1.01 m
WF	0.4647	0.002875	2.0639	0.8889	0.753452	1.21 m
MMSE-LSA	-0.8619	0.003726	2.0238	0.8943	0.797070	1.39 m

The results in Table 7.5 highlight the differing trade-offs established by each classical denoising method. Importantly, the inclusion of baseline metrics provides a critical point of reference for evaluating both classical and learning-based models. Without it, the negative SNR value for MMSE-LSA would misleadingly suggest a failure in denoising—when in fact, it still reduces noise energy effectively.

SS achieves the best improvement in SNR and MSE, reflecting strong numerical

suppression of noise. However, this comes at the cost of perceptual quality, as evidenced by its relatively low PESQ and STOI scores. This is likely due to spectral over-subtraction and residual artefacts, which degrade perceived audio quality.

In contrast, WF and MMSE-LSA adopt more perceptually driven strategies. WF achieves the best PESQ (2.0639) and the lowest LSD (0.753), indicating better preservation of spectral fidelity. Its performance benefited from an adjustment in noise estimation. Where the noise PSD was computed using the mean of the first six STFT frames rather than the minimum—resulting in more stable and effective filtering. MMSE-LSA records the highest STOI (0.8943), suggesting superior intelligibility of speech content. While iterative tuning helped stabilize its behaviour, MMSE-LSA inherently prioritizes log-spectral distortion minimization. As a result, it achieves clear perceptual improvements but offers less numerical gain in SNR compared to SS or WF. This trade-off is consistent with its design focus on perceptual quality over raw energy preservation.

The negative SNR values for both the baseline and MMSE-LSA arise primarily from averaging across test files with highly variable and often low input SNRs. In such cases, even minor reconstruction errors can dominate energy calculations, pulling the average SNR below zero—despite the clear perceptual improvements reflected in PESQ and STOI.

All classical methods complete denoising in under two minutes, confirming their practicality for batch processing and their suitability for real-time deployment when computational efficiency is essential.

It would be helpful to state the duration in s of the processed dataset, to put denoising time in perspective

7.3.2 Machine Learning Models

Training Metrics

With the classical methods evaluated, the focus now shifts to the ML models. Each model defined in the ~~5.2~~ 5.2 section was trained using the same dataset handling strategies and OOM mitigation techniques previously established. The models followed a consistent training configuration: a batch size of 2, accumulation steps of 4, 25 training epochs, and a learning rate of $1 \times e^{-3}$. The table below summarizes the training performance for each model.

Table 7.6 Machine Learning Models Training Metrics

Model	Train Loss	Val Loss	Val SNR	Training Time
CNN	0.2855	0.4292	0.45 dB	4.01 h
CED	0.1599	0.1675	0.88 dB	6.32 h
R-CED	0.1136	0.1239	2.69 dB	7.43 h
R-CED	0.0477	0.0649	6.35 dB	17.44 h
Conv-TasNet	0.0438	0.0478	9.23 dB	1d 1.25h

The training outcomes summarized in Table 7.6 clearly demonstrate a consistently increamenting trend in performance as the models progress in complexity.

The CNN model was introduced as a simple foundation to validate the project's training and evaluation pipeline. It demonstrated reasonable training metrics, though not directly comparable to the classical methods, which do not involve a learning phase. The training plot in Figure 7.2 shows a moderate gap between training (0.2855) and validation (0.4292) loss, around a 50% increase indicating moderate overfitting. Additionally, the validation loss increases during the early epochs, likely due to the model initially overfitting to noise patterns in the training data before learning more generalizable features. Despite this, the validation SNR continues to improve steadily, suggesting that the model gradually captures meaningful structure in the data.

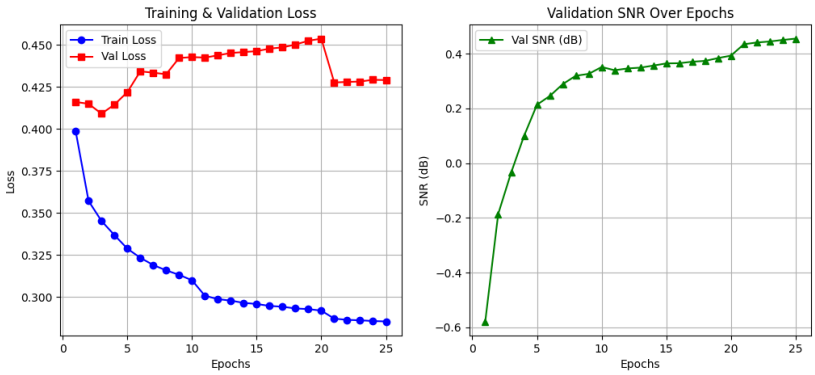


Figure 7.2 CNN training plot.

The CED model expands upon the CNN baseline by adopting a fully connected encoder-decoder architecture. It delivers significant performance gains for only a modest increase in model complexity and training time. As shown in Figure 7.3, both the training and validation loss curves converge smoothly and remain closely aligned throughout training, indicating stable learning and minimal overfitting.

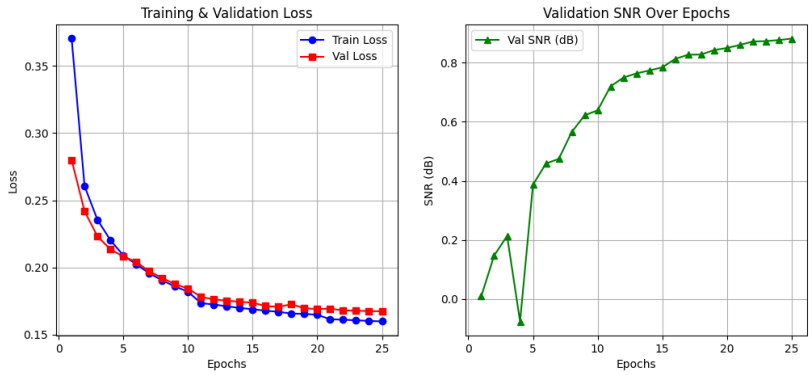


Figure 7.3 CED training plot.

The R-CED model extends the CED architecture by incorporating residual connections to improve information flow and gradient propagation. As shown in Table 7.6, R-CED outperforms CED across all training metrics, with validation SNR increasing from 0.88 dB to 2.69 dB and only a minor increase in training time.

The training plot in Figure 7.4 shows stable convergence in loss curves, though the validation SNR exhibits more fluctuation compared to CED. These oscillations likely stem from the model's increased sensitivity due to its added complexity with residual connections. Nevertheless, the upward SNR trend confirms that R-CED generalizes effectively and benefits from residual learning, supporting the findings of [10].

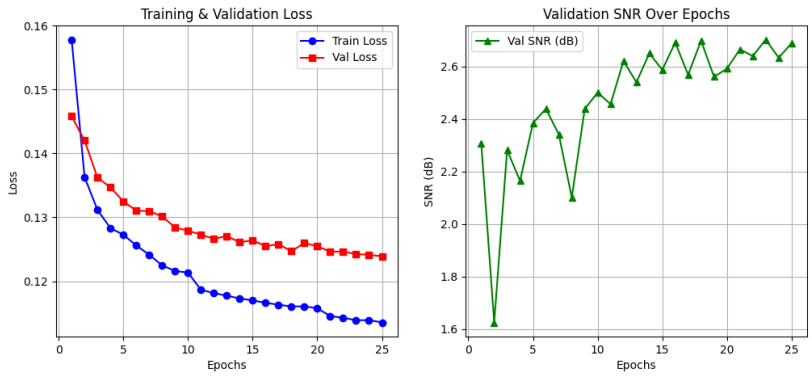


Figure 7.4 R-CED training plot.

The R-CED model achieved the best training and validation loss so far, with loss values of 0.0477 and 0.0649 respectively, and more than doubled the validation SNR to 6.35 dB compared to R-CED. This reflects the strength of its skip connected encoder-decoder architecture, which helps preserve fine grained spectral details during reconstruction.

However, this performance comes at a significant cost. R-CED required 17.44 hours to train, over twice the duration of R-CED. As shown in Figure 7.5, the model converges smoothly, and the validation SNR shows a consistent upward trend with

fewer fluctuations than R-CED. It is worth noting that all recorded training times are situational and can vary with configuration and system load. A full justification of these time performance trade offs is provided in Section 7.3.2.

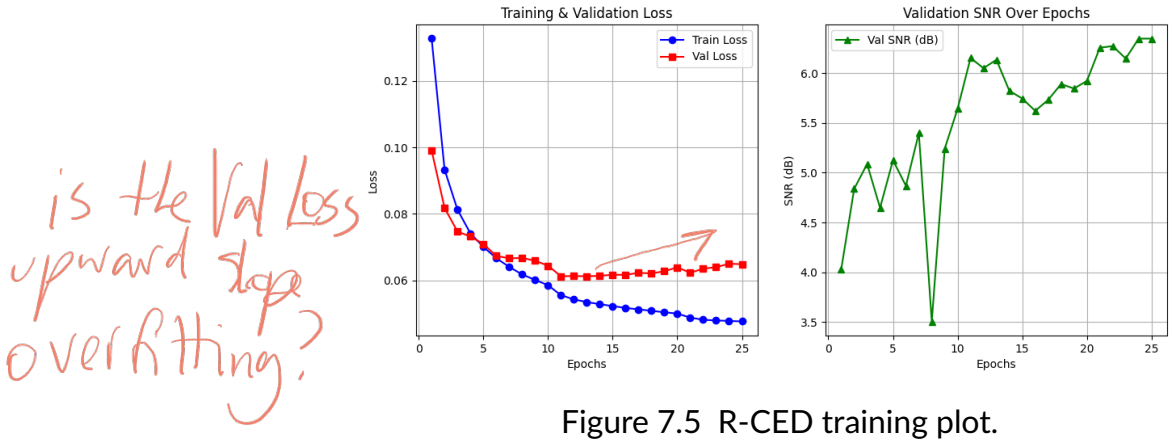


Figure 7.5 R-CED training plot.

The Conv-TasNet model yielded the most notable training results. It achieved the lowest loss values 0.0438 (train) and 0.0478 (validation). The highest validation SNR of 9.23 dB, a substantial improvement over UNet's 6.35 dB. As shown in Figure 7.6, the model demonstrates strong convergence, with the validation SNR stabilizing at a high level after initially fluctuating. This behaviour reflects Conv-TasNet's ability to model complex temporal dependencies more effectively than the previous architectures.

However, this performance came at a computational cost. Conv-TasNet required over 25 hours to train, marking the longest training time in this project. This trend of increasing training duration aligns with the growing complexity and capacity of the models. As architectures become more expressive, it becomes harder to extract further improvements in SNR, necessitating longer optimization cycles to refine smaller, higher level features.

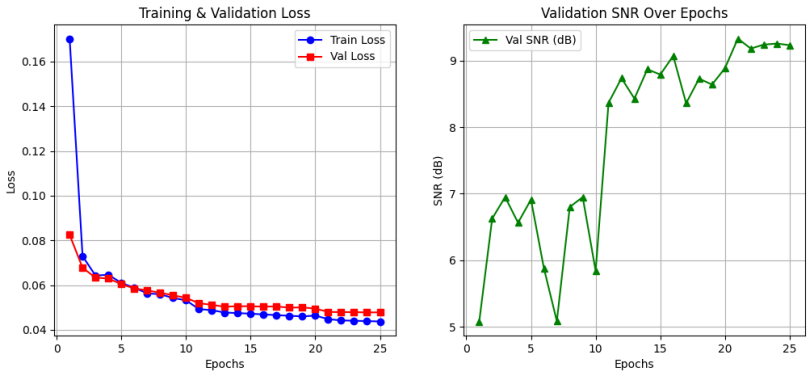


Figure 7.6 Conv-TasNet training plot.

In summary, the training phase across all models showed a clear upward trend in learning capacity and output quality. Each architectural refinement from CNN to Conv-TasNet yielded measurable improvements in loss and SNR, though at the expense of increased training time and complexity. This trade off sets the stage for a comprehensive evaluation in the next section, where these models are benchmarked on real denoising performance.

Denoising Metrics

Following the training phase, each ML model was evaluated using the same denoising pipeline and metrics applied to the classical methods. The results in Table 7.7 summarise performance across objective and perceptual criteria. Consistent with the training phase, the evaluation reveals the same trend of increasing model complexity correlating to improved denoising quality.

Table 7.7 Machine Learning Denoising Metrics

Model	↑SNR	↓MSE	↑PESQ	↑STOI	↓LSD	Denoise Time
Baseline	-2.2774	0.005152	1.8451	0.8928	0.904223	56s
CNN	4.6432	0.001344	1.7410	0.8073	0.795551	1.11 m
CED	13.1850	0.000161	1.6780	0.8386	0.765542	1.05 m
R-CED	14.5285	0.000117	2.0542	0.8677	0.647985	1.14 m
R-CED	16.9872	0.000069	2.1384	0.8940	0.707572	1.27 m
Conv-TasNet	18.0607	0.000063	2.4329	0.9112	0.674051	2.19 m

The CNN model, although introduced as a simple foundation, already outperforms all classical methods in numerical metrics. It achieves an SNR of 4.64 dB and an MSE of 0.001344, improving on SS's 3.08 dB and 0.001525, respectively. However, it underperforms in perceptual quality, with a PESQ of 1.74 and STOI of 0.81, falling short of both WF and MMSE-LSA. This suggests that while the CNN effectively suppresses noise energy, its simple architecture distorts perceptual features. Its LSD of 0.80 confirms moderate fidelity to the original spectral structure but still shows room for improvement.

The introduction of the fully connected encoder-decoder architecture in CED presents a trade off between numerical and perceptual metrics. CED achieves slightly lower PESQ (1.68) and STOI (0.84) scores than CNN, indicating a perceptual quality drop, despite the design's intended purpose of improving it. However, the milestone in SNR (13.19 dB) and MSE (0.000161) are substantial enough to outweigh this perceptual dip. This substantial improvement likely came from the fixed overfitting experienced in the CNN. Now that a large portion of the noise energy is successfully

suppressed, the models that follow must focus on enhancing perceptual quality to surpass the classical methods comprehensively.

Here, the results from [10] are confirmed are once again confirmed with the R-CED model. It delivers comprehensive improvements across all metrics, most notably in perceptual quality. Achieving a PESQ of 2.05 and a STOI of 0.87, values nearly on par with those of WF and MMSE-LSA. These gains justify the use of R-CED over CED, as the added residual connections lead to meaningful enhancements in both intelligibility and naturalness.

The R-CED model and introduction of skip connections is known to come at significant computational cost. However, the results in Table 7.7 show that this complexity is justified by the model's performance. R-CED achieves a significant SNR of 16.99 dB, a PESQ of 2.14, and a STOI of 0.89. This is the first model to pass all classical methods in all metrics. It's important to note that the closer we get to the clean signal, the harder it is to improve the results.

Finally, we have the Conv-TasNet model. The most complex of all the models and computationally expensive. Conv-TasNet achieves remarkable results across all metrics, with an SNR of 18.06 dB, a PESQ of 2.43, and a STOI of 0.91. These metrics are surely suitable for deployment standards in a small scale scenario. The LSE of 0.67 shows how we have moved closer to the clean signal whilst maintaining a good spectral representation. Overall, our model evalution ends with Conv-TasNet achieving the best performance across all metrics compared to both all the classical methods and the other ML models.

8 Future Work

While the project successfully met its primary goal of developing and evaluating a modular speech enhancement system, There remains considerable scope for future work. The pipeline was deliberately designed for modularity, allowing straightforward integration of emerging model architectures. Recent transformer based and diffusion based speech enhancement methods could be incorporated into the existing framework and evaluated using the same metrics. These models may offer gains in both perceptual and numerical performance.

For instance, the transformer based ScaleFormer model [15] is a very powerful architecture used in other domains. It was not implemented due to time constraints and the already high performance achieved, but its results indicate promising future potential. Effectively adapting such models to the spectrogram-based pipeline used here will be key to their success.

Another area for improvement is generalisation. As shown in Appendix C.2, both our best model and many pretrained systems struggled with unseen noise types. One outlier, `DeepFilterNet`, likely benefited from training on a more diverse dataset. This underscores the importance of exposing models to broader training data to ensure better real-world robustness. Since the pipeline was built from scratch. More advanced learning strategies such as reinforcement learning or self-supervised pretraining were not explored. These could accelerate convergence and improve generalisation, especially for larger models or more complex tasks.

Deployment-oriented enhancements also merit investigation. Real devices like headphones and smartphones often use microphone arrays, enabling spatial filtering. Future work could incorporate multichannel input and beamforming to better exploit these cues. Additionally, increased market interest and research into systems integrating playback time feedback or Active Noise Cancellation (ANC) has been seen. Its development does not reach too far from the current pipeline and evaluation could provide valuable insights.

Overall, the flexibility of the current system lays a strong foundation for continued experimentation and development. By extending its capabilities with newer architectures, more diverse data, and real-world constraints, the project can evolve into a deployable solution fit for modern audio enhancement demands.

9 Conclusion

This project set out to explore and compare classical and ML approaches for speech enhancement. A modular pipeline was implemented, supporting flexible experimentation through custom dataset loaders, real-time inference support, and unified evaluation metrics. Critical implementation flaws, such as the improper use of `Tanh` activations, were identified and resolved through iterative metric-based and visual analysis.

Evaluation began with classical methods: SS, WF, and MMSE-LSA, establishing a reliable performance baseline. The project then introduced and assessed five ML models—CNN, CED, R-CED, UNet, and Conv-TasNet. All trained from scratch in the spectrogram domain using the Edinburgh DataShare dataset. A key contribution was the design and evaluation of variable-length dataset handling strategies, including static and dynamic bucketing and padding-truncation approaches. These ensured efficient training across sequences of diverse lengths.

To address hardware limits, OOM mitigation strategies were implemented. While these slightly reduced training accuracy, they consistently improved denoising metrics and enabled deeper models like UNet and Conv-TasNet to be trained successfully. Conv-TasNet emerged as the top performer, achieving an SNR of 18.06 dB, PESQ of 2.43, and STOI of 0.91. Although training was computationally intensive, ConvTasNet took over 25 hours, these were one time offline costs. Inference remained efficient, with the entire test set denoised in under two minutes.

In conclusion, this project demonstrated the viability of training speech enhancement models from scratch using a well-engineered system. The final pipeline supports robust evaluation and real-time denoising, validating ML-based approaches as a practical and superior alternative to classical methods. This work lays the foundation for potential future developments Chapter 8.

Maybe highlight advantage of ML by also listing metrics of best-performing classical method.

References

[1] P. C. Loizou, *Speech Enhancement: Theory and Practice*, 2nd. CRC Press, 2013, ISBN: 9781466504219.

[2] M. Dubey, "Evaluation of signal processing methods for speech enhancement," Advisor: Paris Smaragdis, M.S. thesis, University of Illinois at Urbana-Champaign, 2016. [Online]. Available: http://hdl.handle.net/2142/90373.

[3] Y. Ephraim and D. Malah, "Speech enhancement using a minimum-mean square error short-time spectral amplitude estimator," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 32, no. 6, pp. 1109–1121, 1984.

[4] J. Wei, M. Wang, and H. Yang, "Mmse-lsa based wavelet threshold denoising algorithm for low snr speech," in *2016 IEEE International Conference on Digital Signal Processing (DSP)*, 2016, pp. 253–256. DOI: 10.1109/ICDSP.2016.7868556.

[5] A. Azarang and N. Kehtarnavaz, "A review of multi-objective deep learning speech denoising methods," *Speech Communication*, vol. 122, pp. 1–10, 2020, ISSN: 0167-6393. DOI: 10.1016/j.specom.2020.04.002. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167639319304686.

[6] B. Vachhani, C. Bhat, and B. Das, "Deep autoencoder based speech features for improved dysarthric speech recognition," in *Interspeech 2017*, 2017. DOI: 10.21437/Interspeech.2017-1318.

[7] M. Hobbhahn and J. Sevilla, *What's the backward-forward flop ratio for neural networks?* Accessed: 2025-03-30, 2021. [Online]. Available: https://epoch.ai/blog/backward-forward-FLOP-ratio.

[8] ITU-T Recommendation P.862, *Perceptual Evaluation of Speech Quality (PESQ): An objective method for end-to-end speech quality assessment of narrow-band telephone networks and speech codecs*, https://www.itu.int/rec/T-REC-P.862, Accessed: 2025-04-19, 2001.

[9] S.-H. Yoon and H.-J. Yu, "A simple distortion-free method to handle variable length sequences for recurrent neural networks in text dependent speaker verification," *Applied Sciences*, vol. 10, no. 12, p. 4092, 2020, ISSN: 2076-3417. DOI: 10.3390/app10124092. [Online]. Available: https://www.mdpi.com/2076-3417/10/12/4092.

[10] S. R. Park and J. W. Lee, "A fully convolutional neural network for speech enhancement," in *Proceedings of the Annual Conference of the International Speech Communication Association (INTERSPEECH)*, ISCA, 2017, pp. 1993–1997. DOI: 10.21437/Interspeech.2017-1465.

[11] C. Valentini-Botinhao, *Noisy speech database for training speech enhancement algorithms and tts models*, https://datashare.ed.ac.uk/handle/10283/2791, Accessed: 2025-03-23, 2017.

[12] Creative Commons, *Attribution 4.0 international (cc by 4.0)*, https://creativecommons.org/licenses/by/4.0/legalcode, Accessed: 2025-03-23, 2013.

[13] O. Ronneberger, P. Fischer, and T. Brox, *U-net: Convolutional networks for biomedical image segmentation*, 2015. arXiv: 1505.04597 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1505.04597.

[14] MindSpore Contributors, *Mixed precision training — mindspore documentation*, Accessed: 2025-03-28, 2024. [Online]. Available: https://www.mindspore.cn/tutorials/en/r2.3.0rc2/advanced/mixed_precision.html.

[15] T. Wu, S. He, H. Zhang, and X. Zhang, "Scaleformer: Transformer-based speech enhancement in the multi-scale time domain," in *2023 Asia Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, 2023, pp. 2448–2453. DOI: 10.1109/APSIPAASC58517.2023.10317310.

[16] Y. Hu and P. C. Loizou, "Subjective evaluation and comparison of speech enhancement algorithms," *Speech Communication*, vol. 49, no. 7-8, pp. 588–601, 2007. DOI: 10.1016/j.specom.2006.12.006.

Appendix A Project Structure

The overall structure of the project is modular, enabling clean separation of concerns and supporting future extensibility. The project is organised around two main pipelines **training** and **denoising**. Each pipeline is implemented as a collection of dedicated modules, coordinated through the central `main.py` script, which serves as the entry point for running the system.

To maintain clarity and modularity, all helper modules are organised within a *Utils* directory. This directory contains the core functionality for dataset handling, ML model definitions, training routines, inference logic, and classical baseline methods. Additionally, a centralised configuration file, `config.py`, located alongside `main.py`, manages all project parameters using a dictionary-based structure. This design allows users to easily modify hyperparameters or experiment settings in a single location, improving usability and reproducibility.

An overview of the key files in the *Utils* directory is provided below:

- `dataset.py`: Contains all dataset classes used in the project, implemented as PyTorch `Dataset` objects. The file supports multiple strategies for handling variable-length audio inputs, as discussed in Section 6.1. Along with the neccesary helper functions.

- `model.py`: Defines the ML model architectures used for speech enhancement. Each model is implemented as a subclass of PyTorch's `nn.Module`, and the module is designed for easy experimentation with new architectures or changes in hyperparameters. Many implementation had to be adapted from the original paper and the desing of such changes is discussed in Section 5.2.

- `train.py`: Implements the training loop, validation logic, and performance tracking. It supports training using any of the dataset classes defined in `dataset.py`. It also includes the OOM handeling disucssed in Section 6.2 and justified in Section 7.2

- `denoise.py`: Handles inference and evaluation after training. This module includes two core functions: `batch_denoise` for evaluating entire datasets and `single_denoise` for individual file processing. It supports both ML-based models and classical denoising algorithms, allowing seamless switching between the two approaches. The file also defines the `compute_metrics` function, which calculates objective evaluation metrics such as SNR, MSE, LSD, PESQ, and STOI. All but LSD are calculated using libary function. Furthermore, metric calcualtions are consistent throughout the whole project. Classical methods such as SS, WF, and

MMSE-LSA are self implemented bsince there methods need to be adapted for spectrogram data.

Appendix B Suppression Effects from Improper `Tanh` Activation

Before final evaluations could be conducted, a critical implementation flaw affecting early model performance was identified. Initially all ML models, except Conv-TasNet, included a `Tanh` activation function at the output layer. This was based on the assumption that constraining the real and imaginary spectrogram outputs to the range $[-1, 1]$ would improve numerical stability during ISTFT reconstruction.

However, during early testing and visualization, this constraint was found to significantly suppress amplitude dynamics. The outputs became unnaturally flattened, leading to degraded waveform clarity and poor denoising metrics. SNR, PESQ, and STOI scores in particular, even when models appeared to converge during training.

This effect is illustrated clearly in Figures **??** and **??**, models when trained with `Tanh` applied. The waveforms exhibit a flattened energy profile, and the spectrograms lack contrast and high-frequency content.

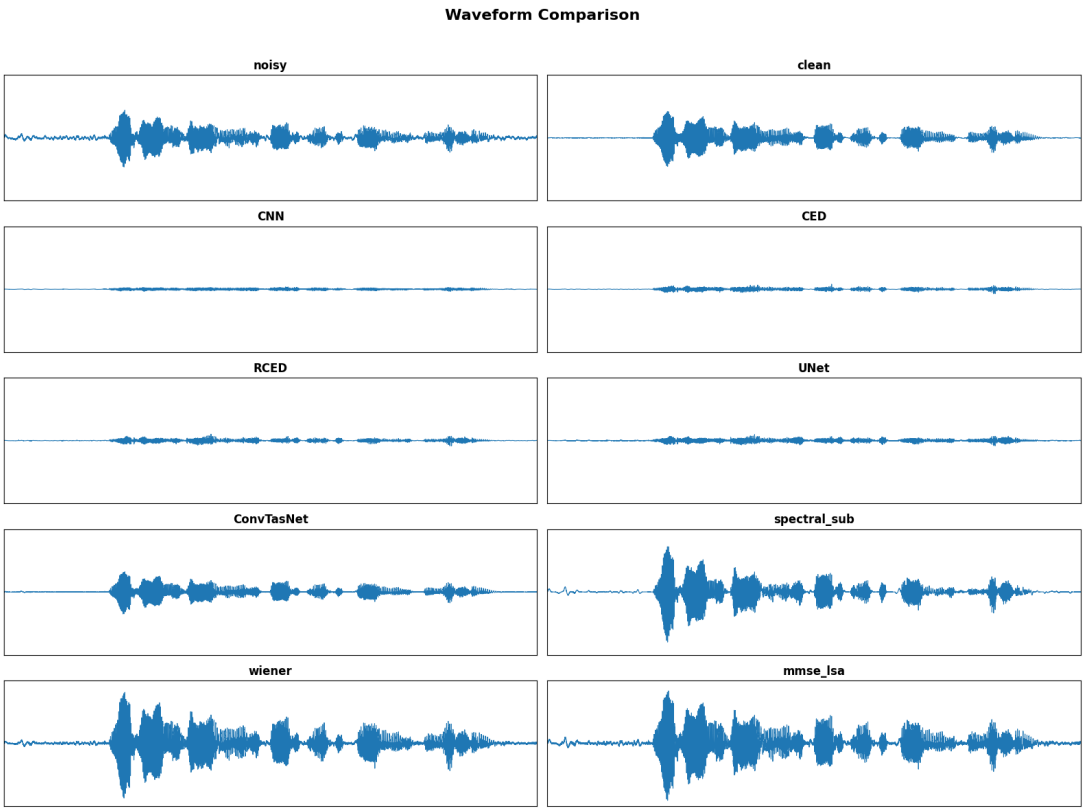


Figure B.1 Flattened waveform energy caused by `Tanh` activation. Output amplitude is artificially suppressed.

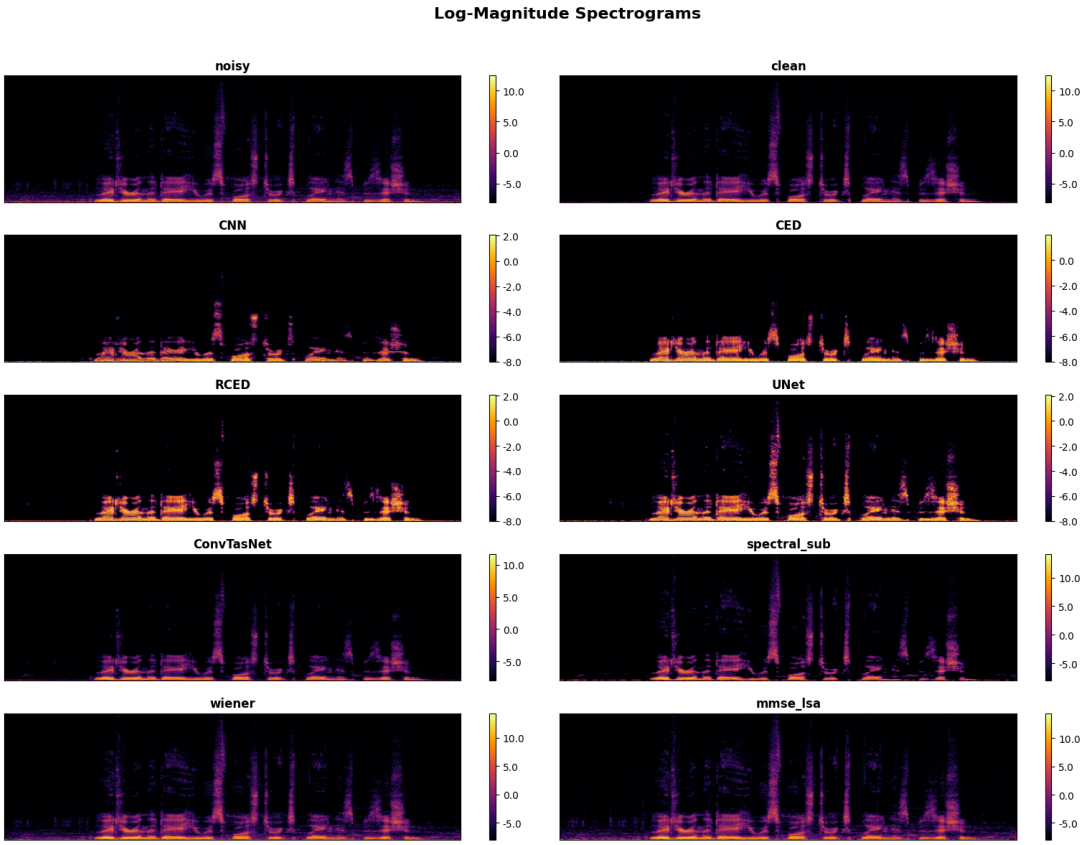


Figure B.2 Spectrograms with Tanh activation exhibit reduced contrast and dynamic range.

After removing Tanh, the models were retrained. The resulting outputs restored amplitude range and natural structure, both in waveform shape and spectrogram clarity. These improvements are shown in Figures B.3 and B.4.

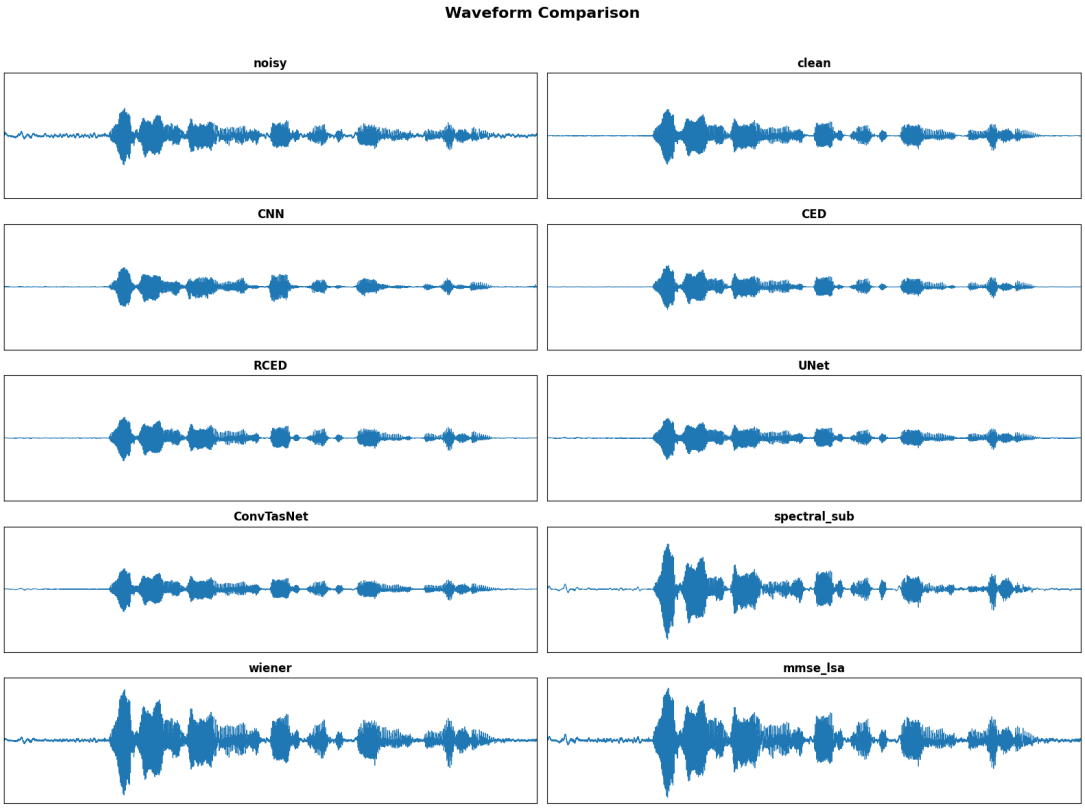


Figure B.3 Corrected waveform comparison after removing Tanh. Energy and dynamics are restored.

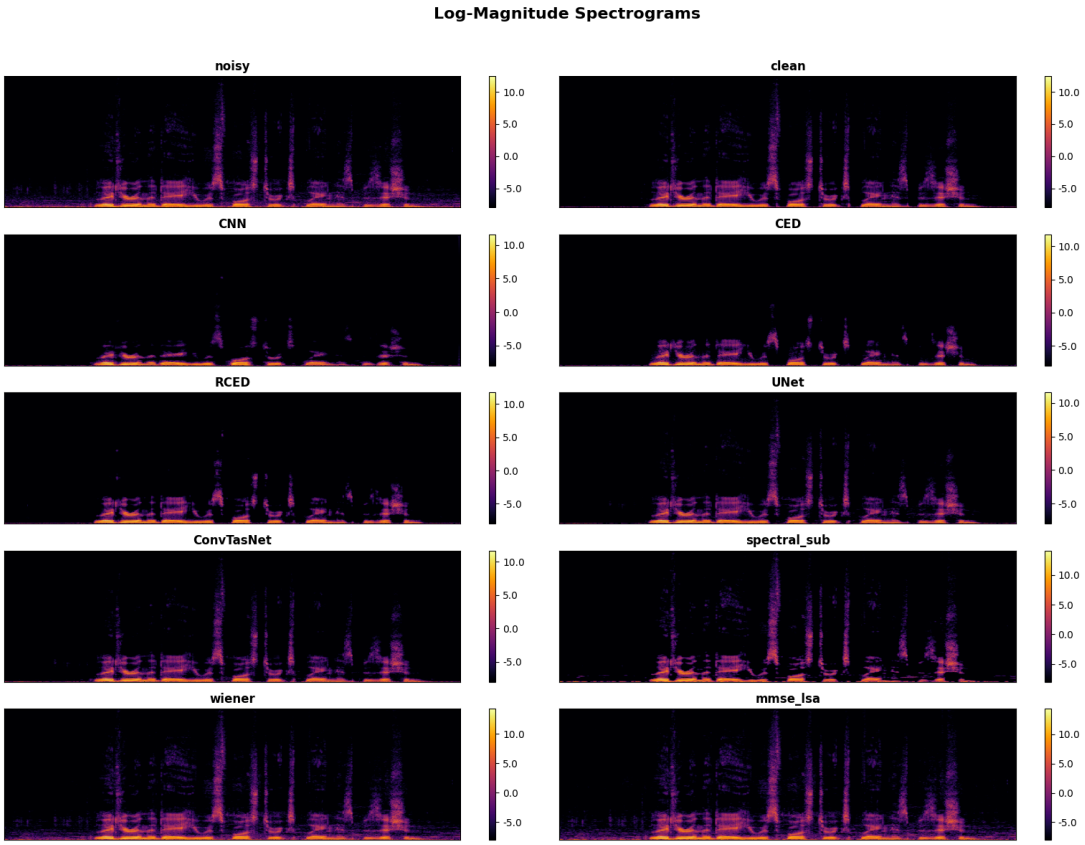


Figure B.4 Log-magnitude spectrograms showing a restored dynamic range and greater clarity.

All final models, presented in Chapter 7 were retrained without the `Tanh` activation. This correction proved critical to achieving meaningful denoising results and validates the importance of properly scaled output layers in spectrogram-based models.

Appendix C Extra Evaluation

C.1 Subjective Evaluation

C.2 Comparison with Pre-trained Models

The results in Table 7.7 clearly show ConvTasNet as our highest-performing model. Importantly, all models in this project were trained entirely from scratch on our chosen dataset and adapted from literature to work with spectrogram-domain data. While this approach yielded strong results, it also means that these models are inherently specialized to the data they were trained on, and may not generalisewell to unseen or real-world audio conditions.

This underscores the value of pre-trained models in the field of speech enhancement. Pre-trained systems like *Demucs*, *VoiceFixer*, and *DeepFilterNet* are designed to work across a wider range of audio conditions, having been trained on large and diverse datasets. They leverage transfer learning to adapt to new tasks or domains, which can be particularly beneficial when the target dataset is limited in size or diversity. These models are often built on architectures that have been shown to be effective and incorporate advanced techniques.

To contextualise the performance of our best model ConvTasNet. We compared it against these pre-trained systems, which are also designed for speech enhancement tasks. We used the same evaluation metrics as in our previous experiments and to ensure a fair comparison two samples are used for this single denoising task. The first sample is drawn from our native dataset, while the second is taken from the NOIZEUS [16], a foreign benchmark dataset for speech enhancement research. This allows us to evaluate the generalization capabilities of these models across different datasets.

Table C.1 Evaluation of Pre-trained Models vs ConvTasNet (per sample)

Sample	Model	↑SNR	↓MSE	↓LSD	↑PESQ	↑STOI
p232_334	ConvTasNet	12.40	0.0002	0.471	2.85	0.993
p232_334	Demucs	-2.95	0.0093	0.928	1.21	0.099
p232_334	VoiceFix	-2.26	0.0079	0.966	1.19	0.009
p232_334	DeepFilterNet	21.62	0.0000	0.460	3.56	0.992
sp07	ConvTasNet	-5.85	0.0059	1.614	1.04	0.230
sp07	Demucs	-4.95	0.0048	1.378	1.04	0.243
sp07	VoiceFix	-7.37	0.0084	1.565	1.04	0.193
sp07	DeepFilterNet	4.25	0.0006	1.012	1.34	0.848

The results in Table C.1 offer important insights into the generalization capabilities of our trained from scratch ConvTasNet model compared to large scale Pre-trained models.

For the `p232_334` sample from our native dataset, ConvTasNet performs exceptionally well, achieving strong scores across all metrics. An SNR of 12.40 dB, PESQ of 2.85, and a near perfect STOI of 0.993. This confirms the model's robustness when applied to data similar to its training distribution. Interestingly, DeepFilterNet significantly outperforms ConvTasNet in SNR (21.62 dB) and PESQ (3.56), suggesting that either a superior architecture or a more advanced training strategy may have been used to achieve such state of the art results. It is unclear, however, whether DeepFilterNet had prior exposure to this dataset or whether the sample is also foreign to its training distribution.

In contrast, Demucs and VoiceFixer underperform on this native sample, negative SNR values and extremely low intelligibility scores (STOI < 0.1). This suggests a mismatch between their design assumptions and the characteristics of our spectrogram domain dataset, or possibly issues in the integration process used to integrate these models into our evaluation pipeline.

The results shift significantly for the `sp07` sample from the NOIZEUS dataset [16]. Here, ConvTasNet's performance deteriorates, yielding a negative SNR of -5.85 dB and a PESQ of 1.04. This decline is expected given the differences in microphone quality, noise types, and recording environments. All of which presented unseen conditions for our model. Similar trends are observed for Demucs and VoiceFixer, both of which also produce negative SNRs and generally lower perceptual scores. However, ConvTasNet still slightly outperforms them in perceptual metrics, suggesting a marginally better generalization capacity.

DeepFilterNet again shows the strongest generalization, maintaining a positive SNR of 4.25 dB and a PESQ of 1.34. Although this performance is lower than its results

on our native sample, it remains superior to all other models. This indicates more robust generalisation to foreign audio. This also suggests that the `sp07` file may be a particularly challenging test case due to its distinct noise profile and recording setup.

Overall, these results validate both the strengths and limitations of our trained from scratch models. ConvTasNet performs strongly within its domain but struggles with generalization. Highlighting the challenge of deploying domain specific models in uncontrolled environments. Conversely, Pre-trained models particularly DeepFilterNet, demonstrate better cross-domain resilience. This underscores the value of large scale, diverse training data and advanced model design when targeting real-world deployment.

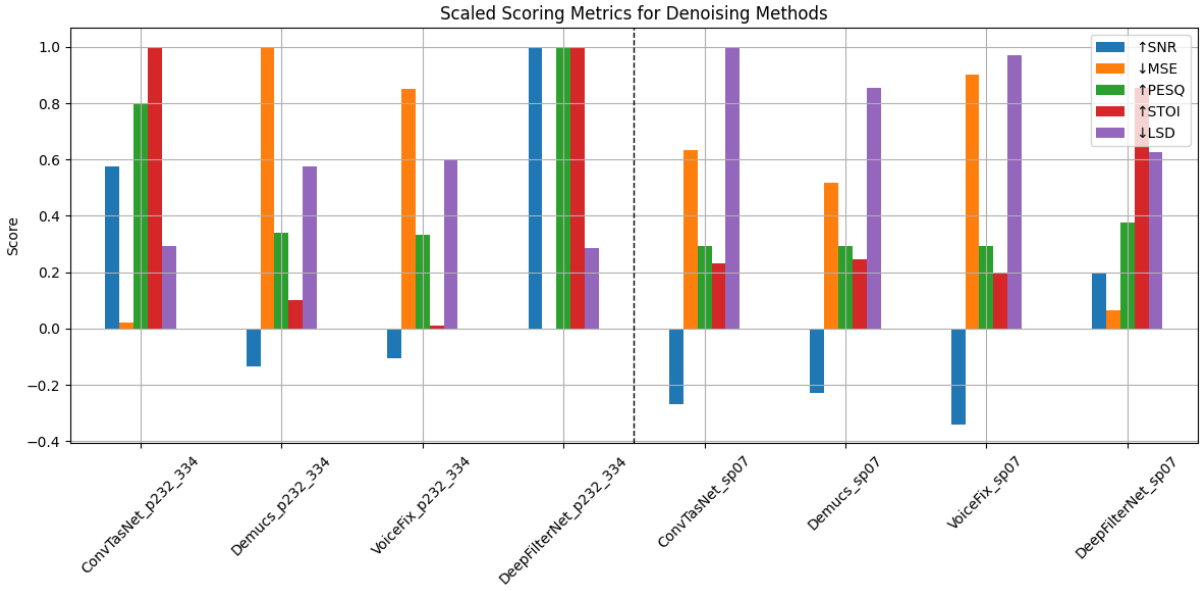


Figure C.1 Scaled Scoring of ConvTasNet vs Pre-trained Models

Figure C.1 provides a visual representation of the evaluation metrics for both ConvTasNet and the Pre-trained models. The scores are normalized to facilitate better visual comparison.