Vector quantization (VQ)–based generative DNN models for low delay speech and audio coding

A model-based approach to encoding packets for packet-loss robustness.

Master's Thesis - Computer Engineering (AI, Vision and Sound)

Lukas Bisgaard Kristensen - Group 1045i

Aalborg University
Electronics and IT

Electronics and IT
Aalborg University
http://www.aau.dk

AALBORG UNIVERSITY
STUDENT REPORT

Title:
Vector quantization (VQ)–based generative DNN models for low delay speech and audio coding.

Theme:
Audio Compression and Packet Loss

Project Period:
Spring Semester 2024

Project Group:
1045i

Participant(s):
Lukas Bisgaard Kristensen

Supervisor(s):
Jan Østergaard
Mohammad Bokaei

Copies: 1

Page Numbers: 57

Date of Completion:
June 26, 2024

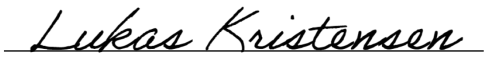
Abstract:

Recent developments in the state-of-the-art of audio compression have led to models achieving low bit rates while maintaining a good reconstruction of the compressed embeddings. The findings make it interesting to explore model-based techniques for making audio packages robust to packet loss, which led to the development of three models with varying bit rates in this project. The three models had bit rates of 768kbps, 192kbps and 6kbps and were trained on the LibriTTS Corpus dataset, where data samples had a bit rate of 384kbps. The largest model showed the best potential for package loss, where it had a good reconstruction ranging from 20% to 80% packet loss probability. The main limitation of the results seemed to be the underlying autoencoders, which opens up for future work applying the same technique for more improved frameworks at lower bitrates.

Contents

Preface

Aalborg University, June 26, 2024



Lukas Bisgaard Kristensen
lbkr19@student.aau.dk

1. Project Summary

This project investigates to which extent a model-based approach can be used for encoding packages to be robust for package loss. It covers the theory behind digital audio structure and what makes it good in terms of sound quality. Where the main focus was the sample-rate, bit-depth and frame-length. A short introduction to internet protocols is being introduced, as the choice between UDP and TCP covers an important concept in the interaction between sender and receiver, mainly having a potentially slow and stable connection or an unstable and fast connection. From here, some of the fundamental techniques for handling packet loss are presented and are mainly divided into sender-based and receiver-based. Finally, for the problem analysis, the concept of vector quantization is covered, and the state-of-the-art model SoundStream[24] is reflected upon. Based on this project's high performance, their model structure was the initial goal to implement for this project. Due to limitations of the scope, it was out of reach to implement all the blocks, and three simplified model versions were made, mainly two normal autoencoders and an autoencoder with vector quantization. The dataset used for this project was the LibriTTS Corpus, where each sample had a bit-rate of 384kbps. The three model implementations had the following bit-rates when passing the audio sample through the encoders: 768kbps, 192kbps and 6kbps, where the last mentioned model uses vector quantization. After having trained the models on the data set, there was a correlation that the higher the kbps, the better the reconstruction loss when training without packet loss. When training for packet loss, there was evidence suggesting that a larger frame length for each packet encoding indicates a better reconstruction loss, but it is not conclusive. The models were trained for the following packet losses: 20%, 40%, 60% and 80%, where the packet loss was a probability on whether their packages would be set to minus one. The high kbps model performed significantly well at all the packet loss rates. In contrast, the results were not as significant for the lower kbps models, but most likely due to the baseline structure not being good enough at the autoencoding task.

2. Introduction

Recent developments from Google in their application "SoundStream"[24] have shown promising results in compressing audio down to a low-bit representation and reconstructing it back to a high-quality version with respect to the original audio. The underlying concept behind this is having a model learn a sufficient amount of relationships and redundancies between its original audio form to outputting compressed embeddings at the encoder. Sound streaming with packet loss remains a crucial element in interactive environments, as it is important to have a stable flow of communication. Packets going missing means losing information in the transmission and waiting for them to be requested again could mean that the information is outdated for the communication. There exist solutions for compensating for missing packages by interpolating between the received packages or by sending additional information, which is meant for reconstruction by, for example, performing XOR operations. This project seeks to investigate the opportunities of using a model-based approach to encode packages with a sufficient amount of information to be robust for packet loss.

2.1 Initial Problem Statement

> "To what extent can Vector Quantization-based generative DNN models be used for low-delay speech and audio coding that is robust to packet loss errors?"

3. Problem Analysis

3.1 Digital Sound Coding

This section covers the fundamental theory of going from an analogue signal to a digitalized signal while presenting values that describe the amount of information used in audio signals.

3.1.1 Analog to Digital Conversion

The way audio is stored in digital format is important in how well it can be reconstructed back to its analogue format when playing it again. Starting with a simple sine wave as seen on Figure 3.1, it can be chosen how many times a second the signal should be sampled. We define this as the sampling rate, which is defined as:

$$sampling\_rate[Hz] = 1/T \tag{3.1}$$

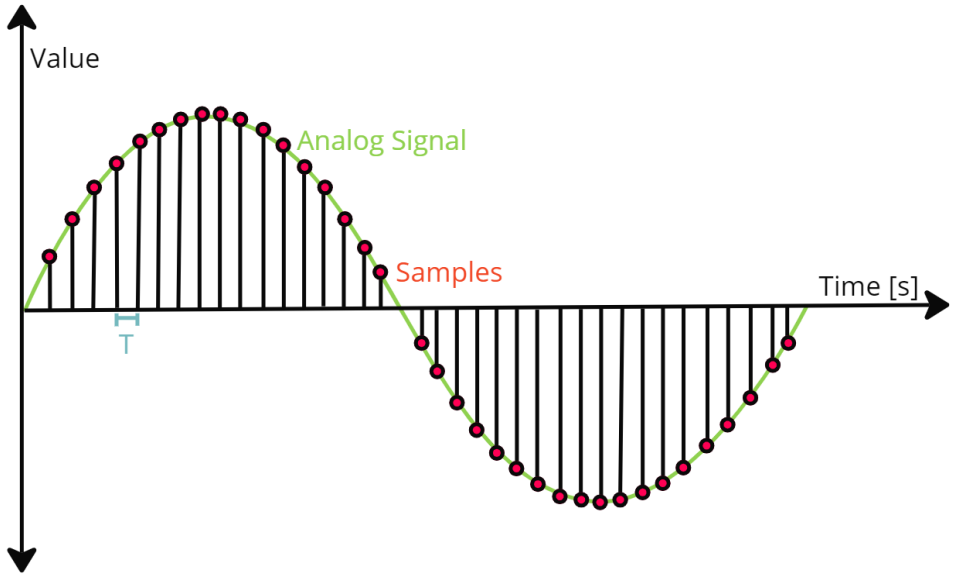


Figure 3.1: Illustration of the sampling frequency of an analogue signal.

Where T is the time between samples, choosing a low sampling rate could mean losing a significant amount of information from the original signal. In contrast, a high sampling rate is expensive for the resources. In communications, it is common to have a sampling rate of 8kHz, 16kHz or 48kHz.[23]

Quantization

After defining how frequently the signal should be sampled, the value of each sample should be saved. Since the analogue signal is a continuous value, it should be converted to a discrete signal to be stored in the memory. The more bits allocated to representing the digitized signal, the more it will look like the original analogue signal. The amount of bits allocated for the digital representation is called bit-depth, which can be seen on Figure 3.2. A common bit-depth for applications would be 16 bits.[18]

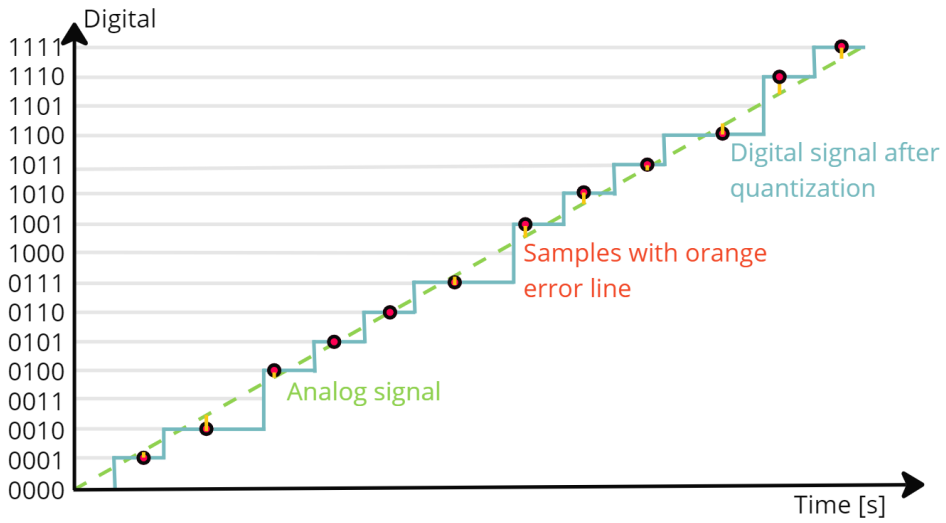


Figure 3.2: Quantization of an analogue signal with a 4-bit binary representation for the bit-depth. This example shows an orange line error between the analogue and digitized signal.

Finally, the bit rate can be calculated after determining the sampling rate and bit depth. The bit rate represents the information contained in the file per second. It is calculated by:

$$bit\_rate[bps] = bit\_depth[bits] * sampling\_rate[Hz] * channel\_amount \tag{3.2}$$

3.2 Packet Coding

When sending a stream of audio for communications, the stream is usually split up into time segments, which we define as frames (see Figure 3.3). The frame is coded into a packet with additional meta-data concerning the packet's origin. Here, a trade-off exists with choosing how large this window should be. If the system is defined to have a long time segment for each frame, the packet receiver will have to wait a significant amount of time before receiving it. In contrast, having a small time segment will be inefficient for packetization as it may prove overhead in transmitting the information through the number of packets sent or the amount of information sent. In the state of the art, a low frame length is typically chosen for audio compression, where the authors of Lyra[17] suggest a window of 20ms.

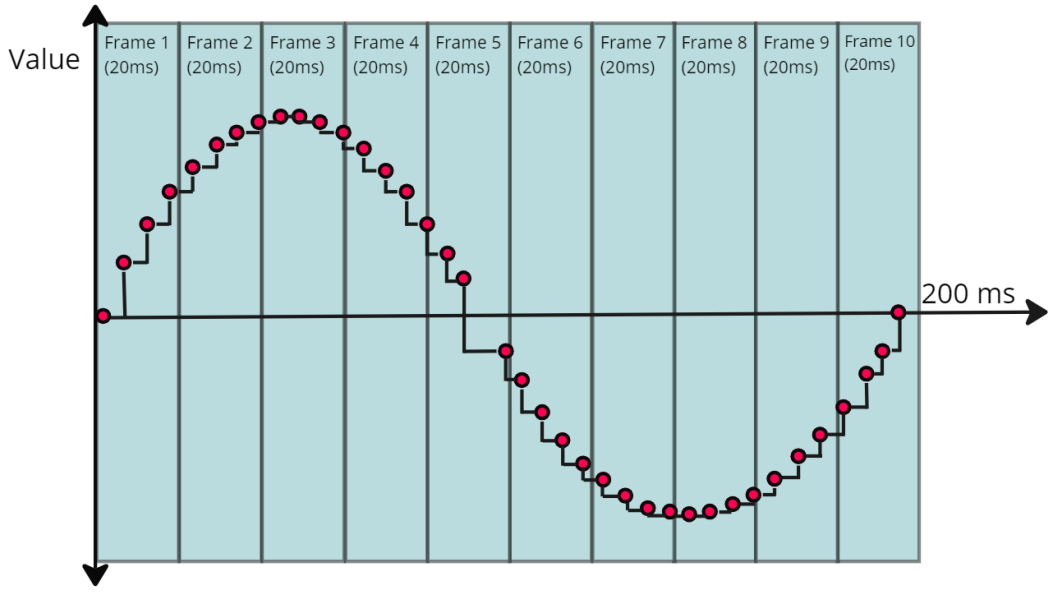


Figure 3.3: Simplified version of 20ms frame segmentation of a 200ms digital signal, which converts 10 frames (190hz sample frequency for simplicity in the example).

3.3 Transport Protocols

After coding the framed audio stream into packages, they are ready to be sent to the receiver. The common transport protocols are Transmission Control Protocol (TCP) or User Datagram Protocol (UDP).[12] The main difference between the two is that the TCP uses a three-way handshake to ensure the reliability of the connection. The handshake consists of the client first sending a synchronization number, after which the server sends an acknowledgement of the connection and finishing it; the client acknowledges the server's response (see Figure 3.4).[14]

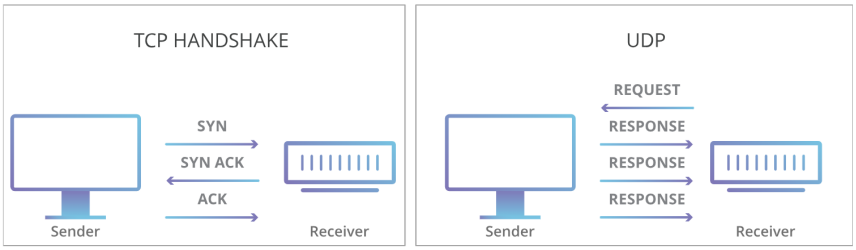


Figure 3.4: TCP vs UDP communication (Figure from: [8]).

This process of validating the client-server connection and the integrity of the packets makes the protocol unreliable for time-sensitive information. In interactive environments, the amount of time it takes for the data to arrive from the sender to the receiver is crucial for the flow of the communication. If a packet is lost, it may not be worth requesting a new one since, by the time it arrives, it may not be relevant to the current context of the communication.

The more preferred protocol for the transport layer regarding time-sensitive information is UDP, which is faster but does not guarantee packet transfers. When using this protocol, the receiver only requests packages without a handshake, after which the sender sends packages without validating their arrival and integrity.

3.3.1 Packet Loss

The amount of packet loss introduced at a connection over time is defined as a packet loss rate (PLR), which we define as:

$$PLR = \frac{N^{TX} - N^{RX}}{N^{TX}} * 100\% \tag{3.3}$$

Where N^{TX} is defined as the number of transmitted packets and N^{RX} is defined as the number of packets received. It should be noted that multiple errors exist within packet loss, where Jérémie Lecomte and Tom Bäckström[4] define them as bit-errors, lost packets and delayed packets. The area of focus for this project is lost packets.

3.4 Packet Loss and Concealment

The need for speed and packet integrity has introduced a list of methods for handling the problem of packets going missing during transmission for "best-effort delivery" protocols. This section covers three categories described by Jérémie Lecomte and Tom Bäckström, the authors of [4] for handling packet loss mainly sender-based, receiver-based and interactive methods. The authors[4] define the term *concealment* as a way of hiding the error and the term *recovery* as retrieving the original input after an error occurs in the signal.[4]

3.4.1 Sender-based

Interleaving

The timeframes are distributed across multiple packets so that the packet loss error is distributed over a longer period of time at the cost of a longer delay.[4] For example, having four time-frames across four packets, the first index of each packet would be from the first packet, and the second index would be for the second packet (see Figure 3.5).[4] While the method still leaves errors for the receiver, it is effective in smoothing them out over a larger time frame.[4]

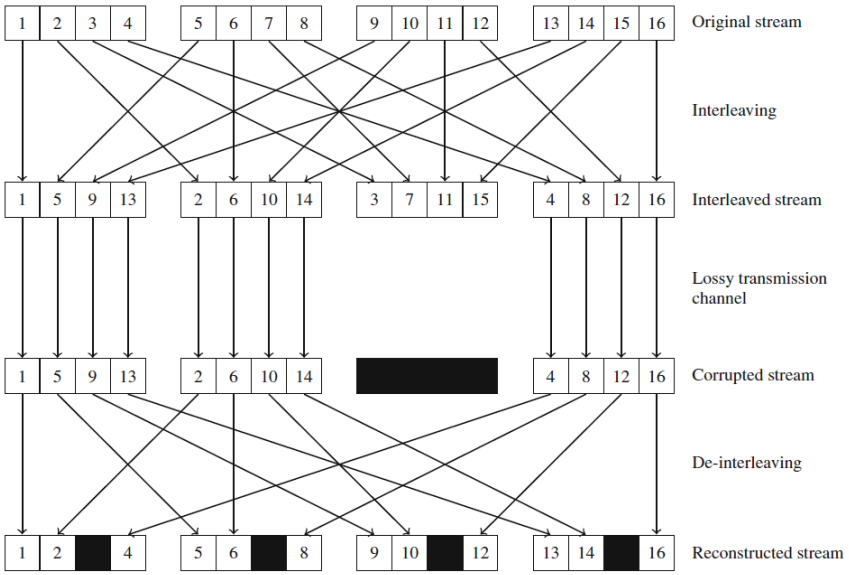


Figure 3.5: Interleaving of 4 packets (Figure from: [4]).

Forward Error Correction

Using Forward Error Correction (FEC) adds redundant information, which can later be used for error correction when retrieving the packets.[4] An example of

FEC is the XOR operation, which effectively retains information about packets that can be used for reconstruction. By defining a variable z (see Table 3.1) as an XOR operation of x and y:

Package x	Package y	Package z (x XOR y)
0	0	0
0	1	1
1	0	1
1	1	0

Table 3.1: XOR truth table

The x or y packets can now be obtained by doing XOR operations on the obtained packages:

$$x\ XOR\ z = \ x\ XOR\ (x\ XOR\ y) = y \tag{3.4}$$

$$y\ XOR\ z = \ y\ XOR\ (x\ XOR\ y) = x \tag{3.5}$$

Assuming that at least two of the three packets arrive without bit errors, the packet receiver would always be able to recreate the original output of x and y.

Multiple Description Coding

A Multiple Description Coder (MDC) splits a signal into multiple channels. Ideally, if one of the channels is received, a good quality can be obtained, and if both channels are retrieved, an ideal quality can be obtained as seen on Figure 3.6.[4]

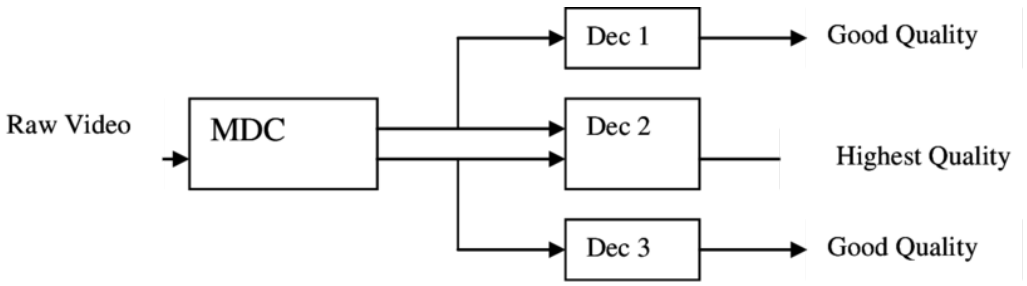


Figure 3.6: Example of MDC for a video stream (Figure from: [2]).

3.4.2 Receiver-based

Insertion-based

For insertion-based methods, the silence substitution involves muting the part where the missing packet is.[4] Noise substitution is described as having white noise represented at the lost packages with lower intensity than the surrounding frames.[4] For packet repetition, the previous frame is repeated at the missing packet.[4] In contrast to the two previous methods, the packet repetition resembles some of the sound characteristics of the context.[4]

Waveform Matching

The waveform matching technique includes matching the frame's start and end to the borders' waveform in the surrounding frames by taking a template of the previous frames.[4]

3.4.3 Interactive Methods

For interactive methods, the encoder and decoder can interact to vary the strategies in the context of the current packet loss.[4] For example, the receiver of the missing packets responds to the sender of the missing packets, either by asking for a retransmission of the missing/broken packages or by switching up the sender-based methods to adapt to the packet loss.[4]

3.5 Vector Quantization

When using vector quantization a vector is taken in as an input and is then quantized by looking into a codebook of template vectors. By doing this, the vectors have a significantly lower data size representation, as the original data can be represented as an index in a codebook at the cost of losing some information. The reference vector is found by minimizing the distance from the input to the nearest code vector:

$$k^* = \arg\min_k ||x - c_k||^2 \tag{3.6}$$

Where c_k represents the code vector for the codebook, x as the input and k^* as the optimal solution.[6]

This method is suitable for lossy compression tasks, where it is given that some information is lost.[6] Recent development within audio compression [24][11] makes use of this technique, where the authors obtain a significant compression rate by making use of vector quantization.

As seen on Figure 3.7, the method uses clustering algorithms, where a set of code vectors represents a set of data points. The vectors are typically optimized using Expectation Maximization (EM) or a gradient descent approach where the loss is minimized by backpropagating through the neural network with the chain rule.[6] As illustrated, the vectors represent the high-frequent data points well, whereas the outliers would have a high reconstruction loss.

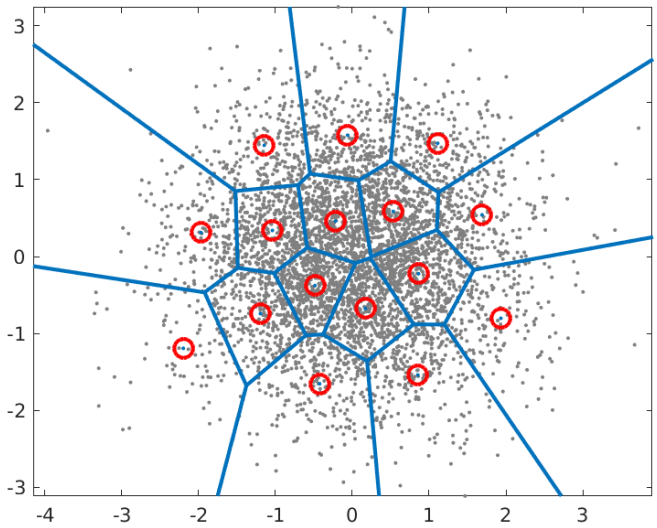


Figure 3.7: Codebook for Vector Quantization. Red circles are the code vectors, grey dots are the data points, and the blue walls are Voronoi Cells. (Figure from: [6])

After optimizing the codebooks, they can be used for inference. As seen on Figure 3.8, the process starts by finding the closest code vector as described in Equation 3.6. From here, the index can be sent through the channel to the receiver so they can look it up in the codebook.

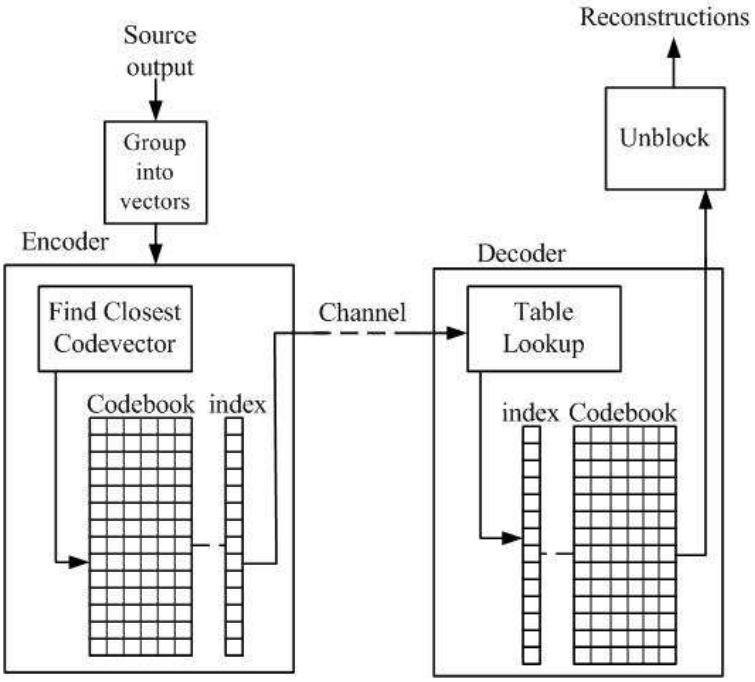


Figure 3.8: Process of converting a source vector into an index from the codebook and reconstructing it by lookup. (Figure from: [10])

By doing this method, a compression rate of the set of vectors to indexing can be defined:

$$input\ size(bits) = vector\_size * bit\_precision \tag{3.7}$$

$$codebook\ index\ size\ (bits) = Ceil(\sqrt{codebook\ vector\ count}) \tag{3.8}$$

In the example from Figure 3.8, the codebook has 14 different codebook vectors representing 6 feature vectors, meaning the index representation could be represented as 4 bits:

$$4^2 = 16\ codebook\ vectors \tag{3.9}$$

Compared to the original latent space output, which is typically 32 bits precision per node, meaning that in this example, it would have a vector with the size

of 6, resulting in 6*32bits = 192 bits. It makes the quantized latent space 2.08% of the original latent space size for transmission through the channel.

3.6 Related Work

3.6.1 SoundStream

Google recently developed a neural network-based codec using the encoder-decoder system with a discriminator and vector quantization (see Figure 3.9) called SoundStream[24]. The codec showed promising results compared to their state-of-the-art, encoding sound down to 3kbps and decoding them back to their original form while maintaining a good perceptual score.[24] In addition to their compression results, they also show promising results for their denoising feature. They showcase that they can dynamically turn this feature on and off during interference, which makes it effective for cases where noise may happen dynamically. Finally, they also show scalability for their bitrates in their model, making it flexible to handle multiple compressions and could serve as an interactive method between sender and client depending on the current bandwidth.[24]

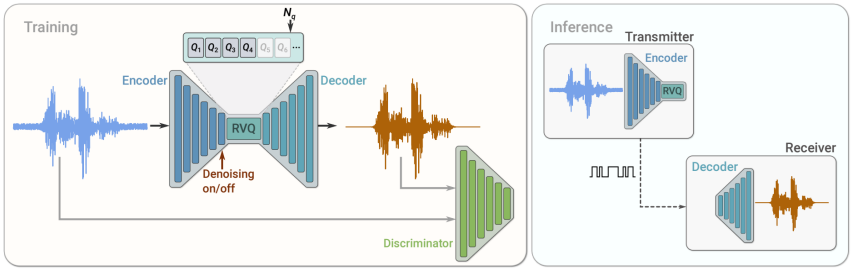


Figure 3.9: Model Architecture of SoundStream at training and inference. (Figure from: [24])

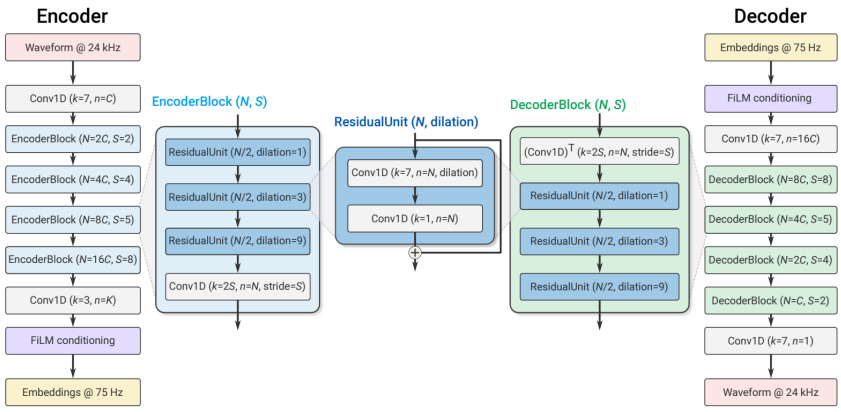


Figure 3.10: Encoder and Decoder of SoundStream. (Figure from: [24])

Encoder Blocks

The encoder consists of four encoder blocks, where the features passed through the network are down-sampled for each block (see Figure 3.10). The down-sampling process consists of three Residual Units containing two one-dimensional convolutional layers, and at the end of the unit, an additional one-dimensional convolutional layer is present.[24] The channels are doubled when the features are passed through the encoder block.[24]

Decoder Blocks

The decoder follows the same procedure as the encoder; it uses the same residual units, and the channel amount is halved for each decoder block.[24]

Residual Vector Quantization Layer

The model utilizes a residual version of vector quantization, which involves passing through multiple stages of vector quantization.[24] The use of residual vector quantization allows for having varying bitrates, which scale with a lower computational cost.

3.7 Final Problem Statement

A final problem statement was chosen based on the problem analysis:

> "To what extent can Vector Quantization-based generative DNN models be used for low-delay speech and audio coding that is robust to packet loss errors?"

4. Dataset

4.1 LibriTTS corpus

Data set accessed through `https://openslr.org/60/`.

4.1.1 Motivation for usage

In the field of speech synthesis, the LibriTTS is one of the standardized datasets used for benchmarking performance.[9] It was created by Zen et. al [25] as a modification of the original dataset LibriSpeech to suit the purpose of text-to-speech(TTS) research.[25] The samples from the original "LibriSpeech ASR corpus" originate from a set of audiobooks, where the authors[21] trained models to align the original text to the read text to generate sentence samples for the data set.[21] Additionally, a significant change is that it has a higher sampling rate of 24kHz in contrast to the previous 16kHz.[25] The new modifications of the dataset for the LibriTTS corpus preserved only the data samples which fit a set of requirements[25]:

- Alignment of sample to text
- Length of sample
- The average word duration
- Signal-to-Noise-Ratio (SNR)

Where 30% of the original data were removed due to misalignment, 25% due to SNR and less than 1% due to the length of the sample and the word duration.[25]

4.1.2 Specifications

For the project's current scope, it was chosen to work with the "clean" dataset in contrast to the "other" dataset, which is more challenging. The samples from the dataset have a sample rate of 24kHz and a bit rate of 384kbps - resulting in a bit-depth of 16 bits for representing each sample. For each data sample in the data

set, the .wav file has two .txt files with the original text from the speech input and a normalized text from the speech.

Split type	Size	Samples	Duration	Speakers
Development (clean)	1.2GB	5736	8.97 hours	40
Training (clean-360)	27.0GB	116500	191.29 hours	904
Testing (clean)	1.2GB	4837	8.56 hours	39

Table 4.1: Distribution of the clean splits of the LibriTTS corpus data set. [25]

4.1.3 Dataset Distribution

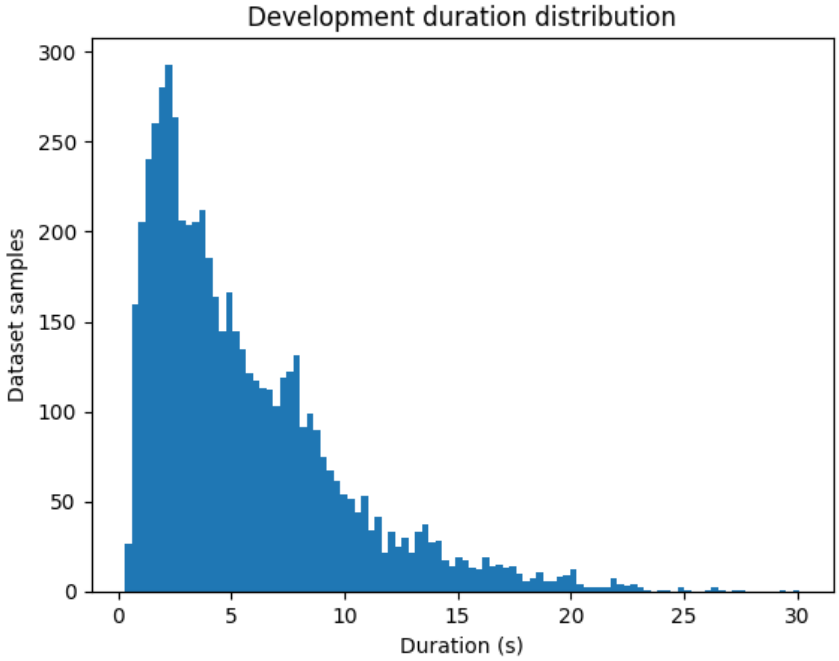


Figure 4.1: Distribution of the sample length of the development data set (100 bins)

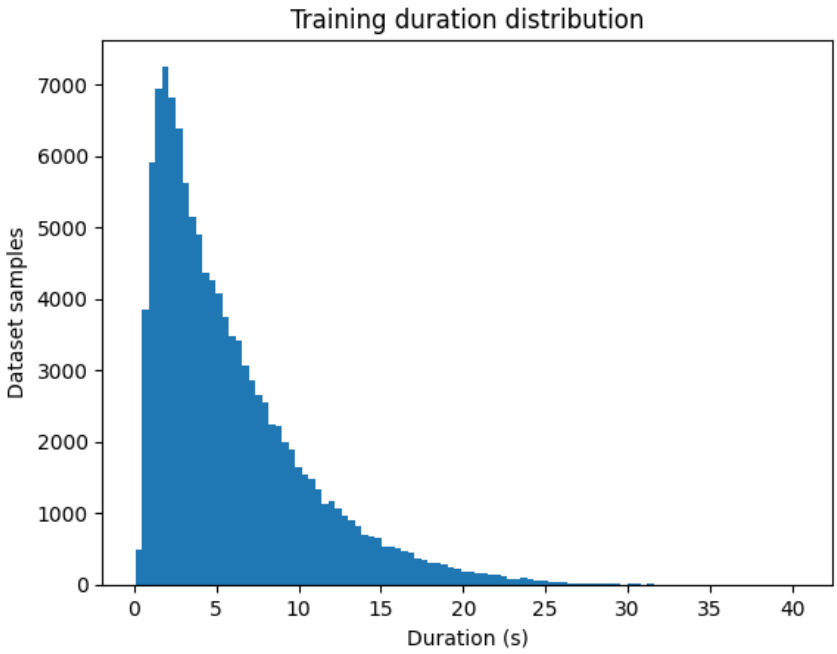


Figure 4.2: Distribution of the sample length of the training data-set (100 bins)

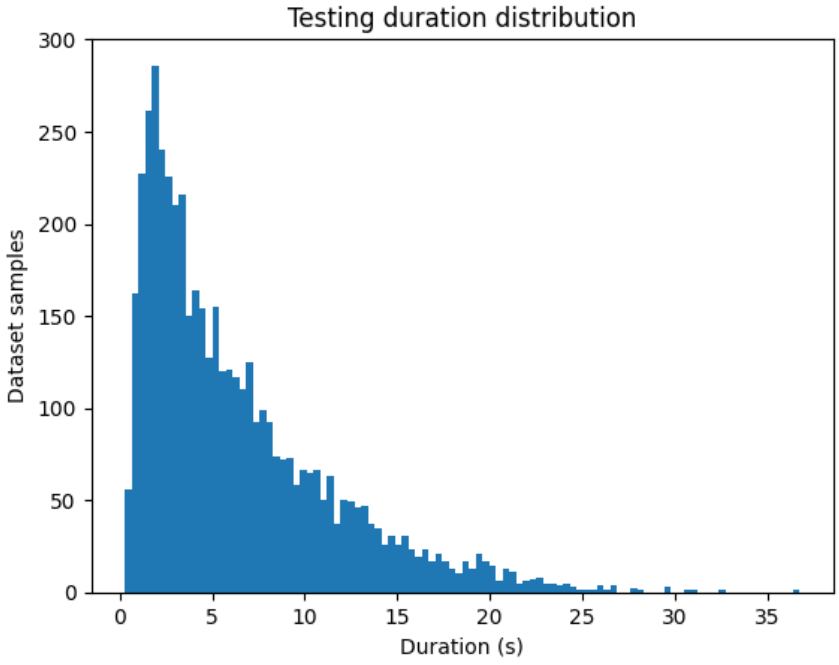


Figure 4.3: Distribution of the sample length of the testing data set (100 bins)

4.1.4 Examples

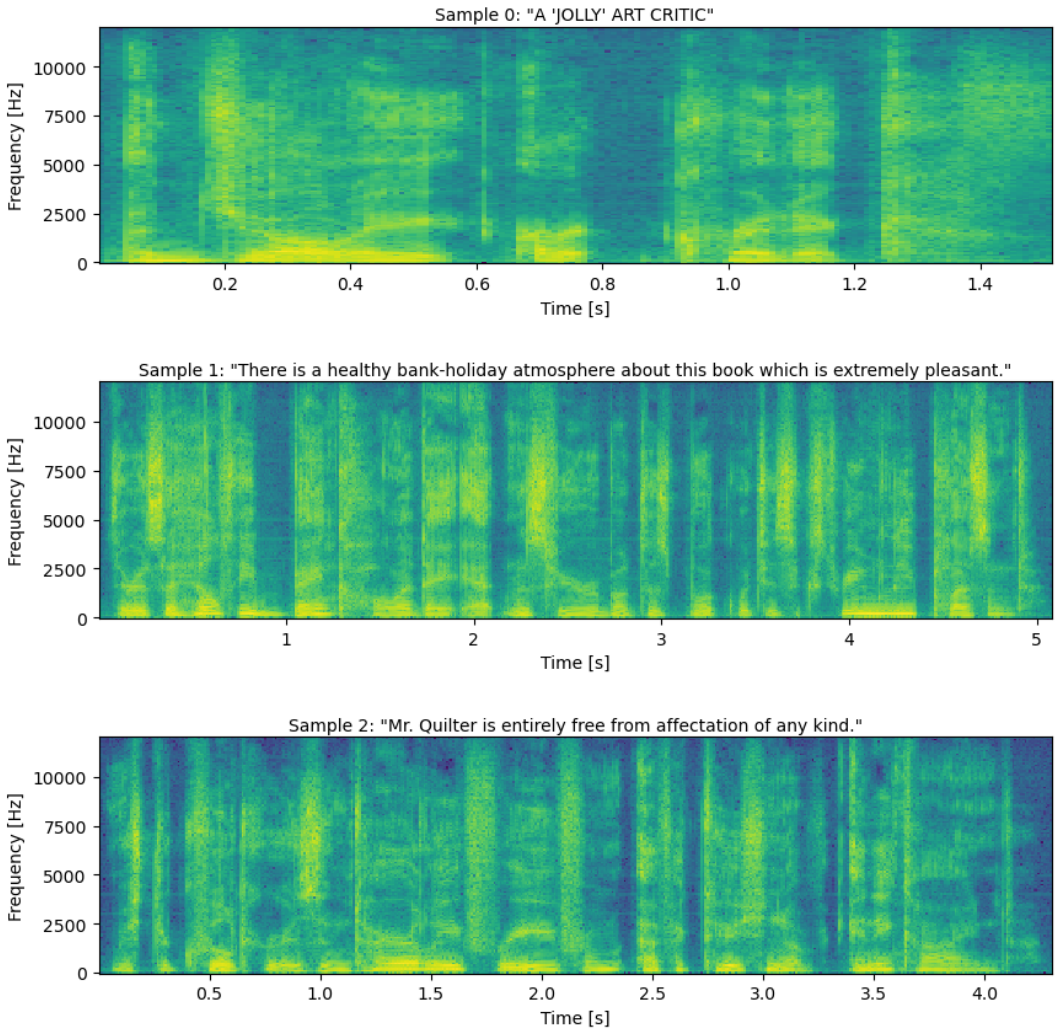


Figure 4.4: Frequency Spectrograms of the first three samples from the development dataset[25].

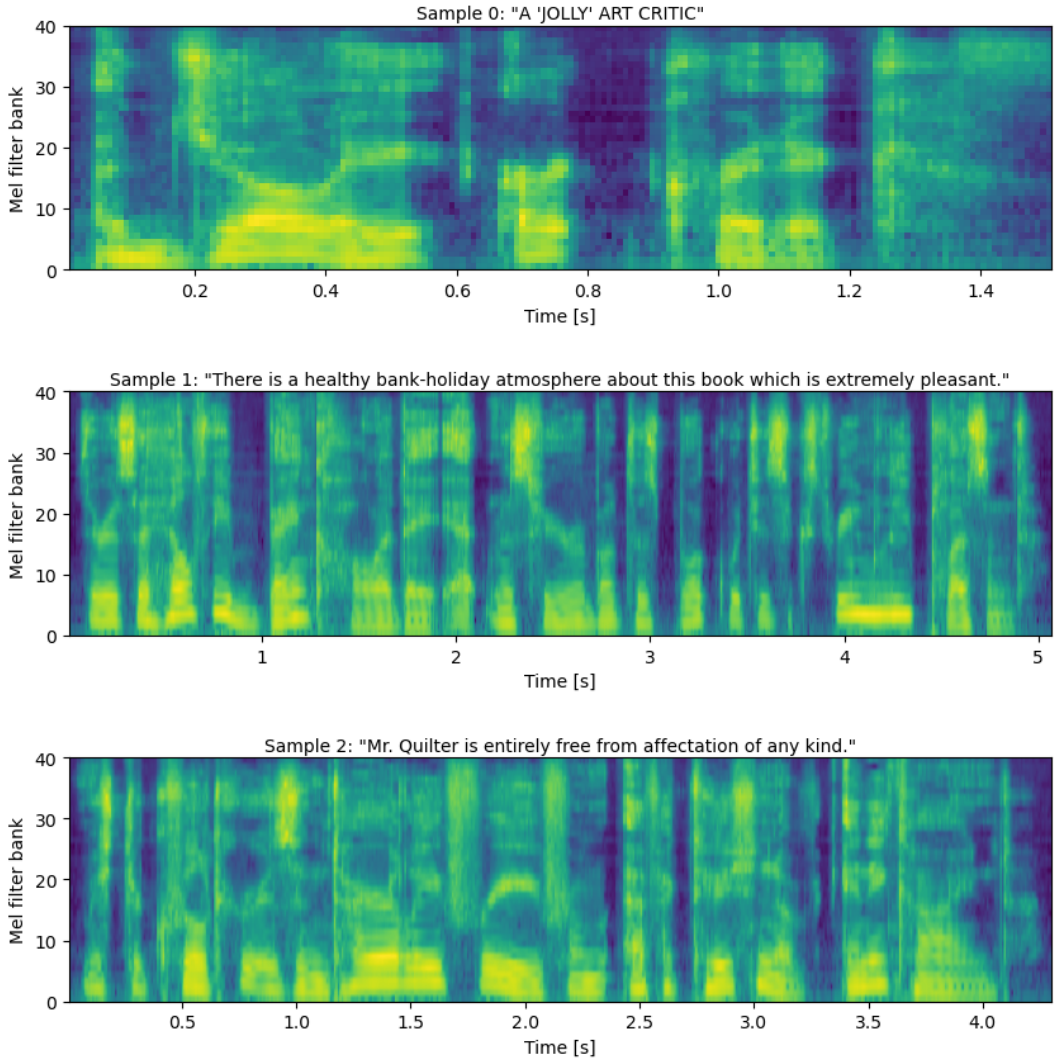


Figure 4.5: MEL Spectrograms of the first three samples from the development dataset[25].

4.2 Pre-processing

For the data preprocessing, each sample was split into frames of 20ms with a non-overlapping window, and any last samples below 20ms would be removed from the dataset. With a sample rate of 24kHz, each frame would have:

$$24000Hz * 0.02s = 480\ samples \tag{4.1}$$

$$480samples * 16bit\ depth = 7680\ bits \tag{4.2}$$

5. Methodology

5.1 Development

The models were developed in Python (3.12.2) using TensorFlow[3] for modelling. The training of the models was monitored using the development platform Weights and Biases[5], where logs were kept for comparison across the different runs. During the development, Github Copilot[16] and ChatGPT[1] were used to improve the model implementation and debugging process.

5.1.1 Hardware

Most of the model training was done on the cloud platform Strato provided by CLAAUDIA[7]. The machine which was used had 16 VCPUs and 64 GB RAM.

5.2 Concept

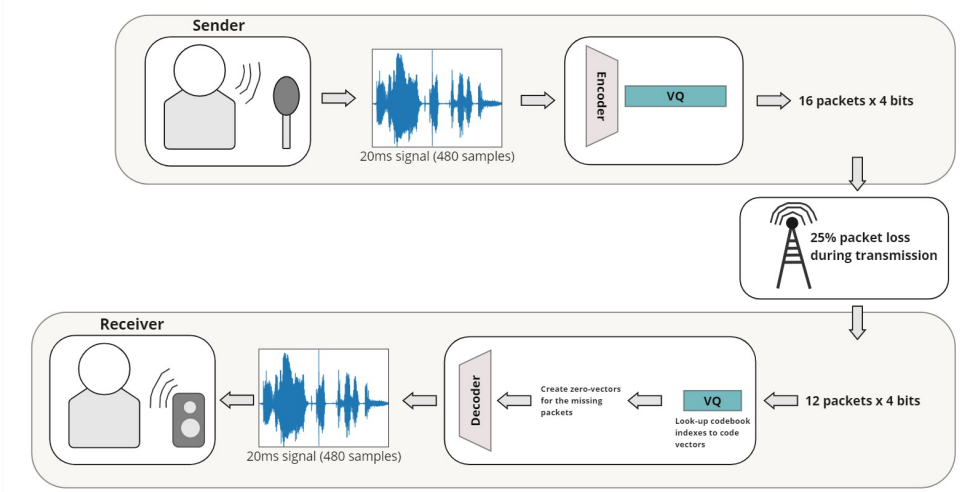


Figure 5.1: Model concept for having a sender send a voice packet, which has a 25% packet-loss probability of being lost during transmission and the receiver decoding the packages.

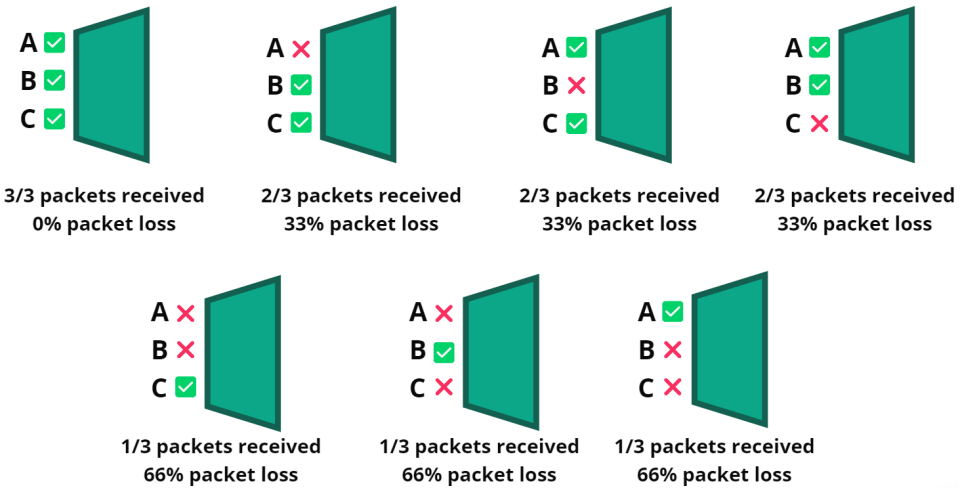


Figure 5.2: Concept of decoder variant to multiple packet losses at inference.

5.3 Model

5.3.1 Vector Quantization

The library used for vector quantization was used from: `https://github.com/lucidrains/vector-quantize-pytorch`. A library which has been created based on the implementations of Deepmind. It consists of variations from the state-of-the-art research, which utilizes vector quantization like SoundStream[24]. As proposed by [24], a set of centroids are being trained on the first batch by running the k-means algorithm instead of randomly assigning them.[20]

5.3.2 Loss Functions

Defining the loss functions for training the models, the mean-squared error was chosen, which is defined as:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(Y_i - \hat{Y}_i)^2 \tag{5.1}$$

In addition to the MSE of the reconstructed audio compared to the original, another loss is taken into consideration based on how well the Vector Quantization performs at indexing its codebook vectors:

$$commitment\_loss = \frac{1}{N}\sum_{i=1}^{N}||x - c_k||^2 \tag{5.2}$$

This is a loss function for measuring the distance between the assigned codebook vectors compared to the original; the commitment loss is used when back-

propagating to optimize the codebook vectors. Since the reconstructed audio error might be more important than the commitment loss of the codebook vectors, a weighted value is applied to the commitment loss defined as a cost to prioritize between the two:

$$total\_loss = MSE + commitment\_loss * commitment\_cost \tag{5.3}$$

Additionally, experimentations were performed with the STFT loss on a discriminator as suggested by the SoundStream[24] architecture, but due to limitations of the development scope, it was not chosen to include a discriminator approach to the model.

5.3.3 Probability-based packet loss

Defining 10 example packages with a length of 5:

$$\begin{bmatrix}0\\1\\2\\3\\4\end{bmatrix}\begin{bmatrix}5\\6\\7\\8\\9\end{bmatrix}\begin{bmatrix}0\\1\\2\\3\\4\end{bmatrix}\begin{bmatrix}5\\6\\7\\8\\9\end{bmatrix}\begin{bmatrix}0\\1\\2\\3\\4\end{bmatrix}\begin{bmatrix}5\\6\\7\\8\\9\end{bmatrix}\begin{bmatrix}0\\1\\2\\3\\4\end{bmatrix}\begin{bmatrix}5\\6\\7\\8\\9\end{bmatrix}\begin{bmatrix}0\\1\\2\\3\\4\end{bmatrix}\begin{bmatrix}5\\6\\7\\8\\9\end{bmatrix} \tag{5.4}$$

For each of the packages, a random float would be generated:

$$\begin{bmatrix}0.12 & 0.34 & 0.90 & 0.71 & 0.55 & 0.64 & 0.03 & 0.89 & 0.40 & 0.21\end{bmatrix} \tag{5.5}$$

A packet loss is defined to be, for example, 30%. All the packages with their randomly generated associated float to be below the packet-loss threshold would be marked for having their array set to minus one. We defined this as a boolean for whether the packet would be received during the transmission:

Random Float	0.12	0.34	0.90	0.71	0.55	0.64	0.03	0.89	0.40	0.21
Input frame	x	✓	✓	✓	✓	✓	x	✓	✓	x

Table 5.1: Randomly assigned float to each packet for packet-loss threshold selection.

Applying minus ones to the lost packages indexes retrieved from Table 5.1:

$$\begin{bmatrix}-1\\-1\\-1\\-1\\-1\end{bmatrix}\begin{bmatrix}5\\6\\7\\8\\9\end{bmatrix}\begin{bmatrix}0\\1\\2\\3\\4\end{bmatrix}\begin{bmatrix}5\\6\\7\\8\\9\end{bmatrix}\begin{bmatrix}0\\1\\2\\3\\4\end{bmatrix}\begin{bmatrix}5\\6\\7\\8\\9\end{bmatrix}\begin{bmatrix}-1\\-1\\-1\\-1\\-1\end{bmatrix}\begin{bmatrix}5\\6\\7\\8\\9\end{bmatrix}\begin{bmatrix}0\\1\\2\\3\\4\end{bmatrix}\begin{bmatrix}-1\\-1\\-1\\-1\\-1\end{bmatrix} \tag{5.6}$$

5.3.4 Architecure

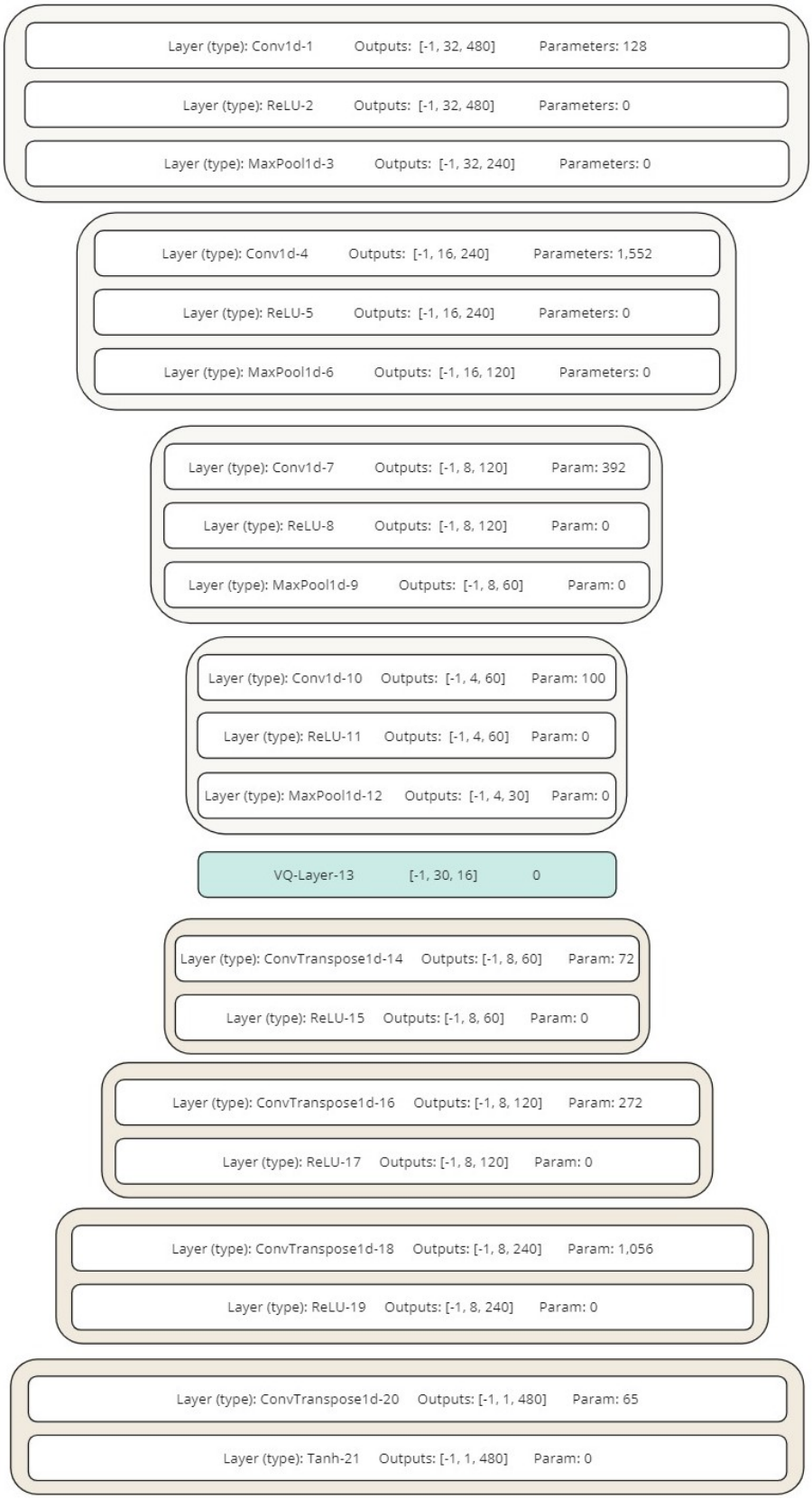


Figure 5.3: Model architecture for input size 20ms (480samples), where the first index (-1) is a placeholder for the batch-size.

5.4 Model Layers

The layers for the encoder and decoder follow a traditional Convolutional Neural Network (CNN) model structure, with a set of convolutional layers with pooling operations and activation functions being fully connected. As proposed by the Soundstream paper[24], a vector quantization layer is between the encoder and decoder, where they are being trained with k-means clustering on the first batch. It was out of scope to implement the entire SoundStream[24] structure during development, so it was gradually implemented considering multiple layers of complexity and considering the author's suggestions.

5.4.1 Convolutional

Unlike the SoundStream[24] paper, no experimentations were done with the dilation of the convolutional layers. The SoundStream[24] structure has an increasing dilation size for each ResidualUnit in the encoder block, which should help sampling from a wider context in the feature space. During the development, the model complexity had been varied. In the final tests, two models were made, where one of them had the same dimensionality as the input size but with double bit-depth, meaning that it was double its input size measured in bits. The other encoder variation had 25% of the dimensionality as its input, which had a bit-depth of 32bit and resulted in the encoder output having half the size of its input measured in bit size. An illustration of a convolution can be seen on Figure 5.4, where a dilation rate of one is set, with a stride of one and a kernel size of 3x3.

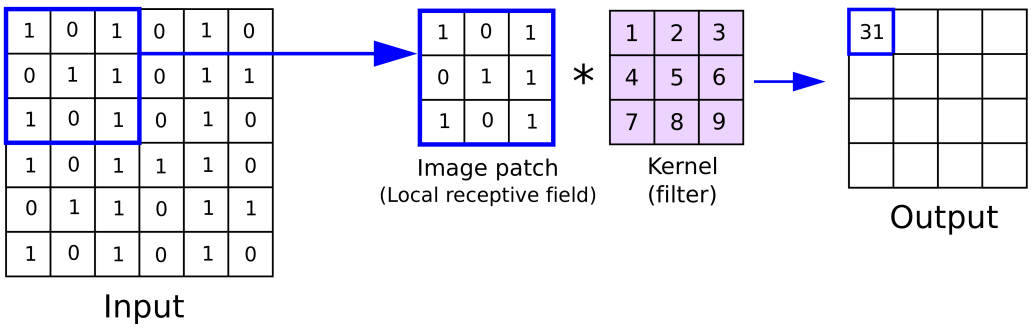


Figure 5.4: Convolutional operation example (Figure from: [22]).

5.4.2 Convolutional Transpose

The model's decoder blocks are similar to the encoders, where the convolution operation is a transposed operation and does not have a max pooling operation at the end. In contrast to the encoder's convolution operation, where the sliding window of the kernel is applied to the same size the transposed applies the kernel window to a smaller dimension in the upsampling process. (see Figure 5.5).

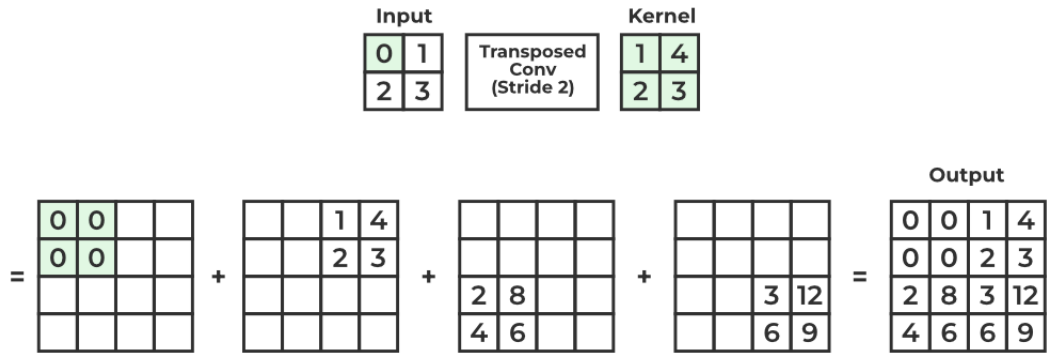


Figure 5.5: Example of Transposed Convolution operation with stride 2 (Figure from: [15]).

5.4.3 Max Pooling

Max pooling is used for downsampling the feature space. It takes the maximum value over a filter defined by size and stride. The values which are used by the model are the same as seen on Figure 5.6, but where the operation is one-dimensional. The filter used is non-overlapping, with a filter-size of two and a stride of two.

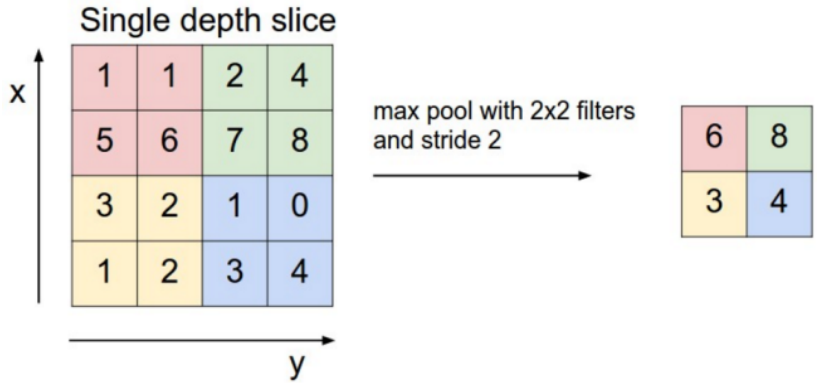


Figure 5.6: Max pooling operation (Figure from: [13]).

5.4.4 Activation Functions

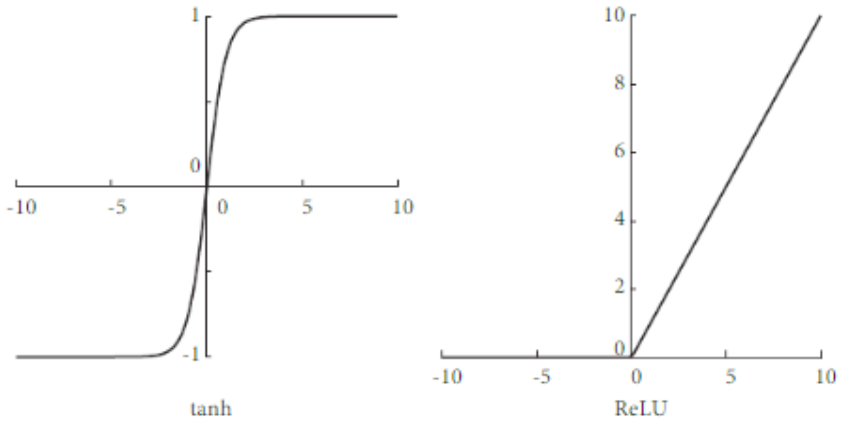


Figure 5.7: Hyperbolic Tangent and Rectified Linear Unit activation functions (Figure from: [19]).

For all of the layers except the last the activation function used between the layers is Rectified Linear Unit (ReLU), which is defined as:

$$ReLU(x) = max(0, x) \tag{5.7}$$

In the final layer, the signal is reconstructed back to the input signal's original range [-1, 1] by using the Tanh activation function:

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{5.8}$$

5.5 Model complexities

At the end of the development phase, three different models with varying complexities were made with the following specifications:

Name	Architecture	Embedding Size	Bit Depth	Bit Size	Bit-rate
Model A	Autoencoder	480	32	15360	768kbps
Model B	Autoencoder	120	32	3840	192kbps
Model C	Autoencoder VQ	30	4	120	6kbps

Table 5.2: Model list comparison, where the input is a 20ms frame with 24kHz sampling rate and 16bit-depth.

Based on the following input frame specifications for the models:

	Sample rate	Frame size	Samples	Bit-depth	Bit size	Bit-rate
Input frame	24kHz	20ms	480	16	7680	384kbps

Table 5.3: Defining the frame specifications for reference to model statistics.

5.6 Hyper-parameter Sweeping

A parameter sweep was conducted for the hyper-parameters of the model across the following:

Hyper-parameter	Variations
Epochs	10-50
Batch Size	16-32-64-128-256
Commitment Cost	0.2-0.4-0.6-0.8-1.0
Learning Rate	0.001-0.0001-0.00001
Vector Quantization Dimension	4-8-16-32-64

Table 5.4: Set of hyperparameters with their ranges

For the parameter sweep, a limited set of data set sizes was chosen to limit the computation time at the benefit of trying more variations. A set of 1000 data points was chosen, and the average run-time for each sweep was 51.62 minutes. After splitting the 1000 data points into 0.02sec frame-lengths, the training data contained 88163 samples, and the validation contained 21998 samples. The initial sweep contained 216 variations of parameters, where it took a total of 185.83 hours to compute. Following this sweep, more in-depth sweeps containing wider ranges were conducted for the parameters relevant to vector quantization and configuring the learning rates and batch sizes (see Figure 5.8 and Figure 5.10).

The sweeps were computed using a nested for loop with each variation and were for each epoch logged to a Weights and Biases project. After the sweep was completed a .csv file was downloaded from the Weights and Biases project with the hyper-parameter variations together with the training and validation losses. A pair-wise comparison was done using the file: *grid_search_data_analysis.py* for visualizing trends in the parameter adjustments.

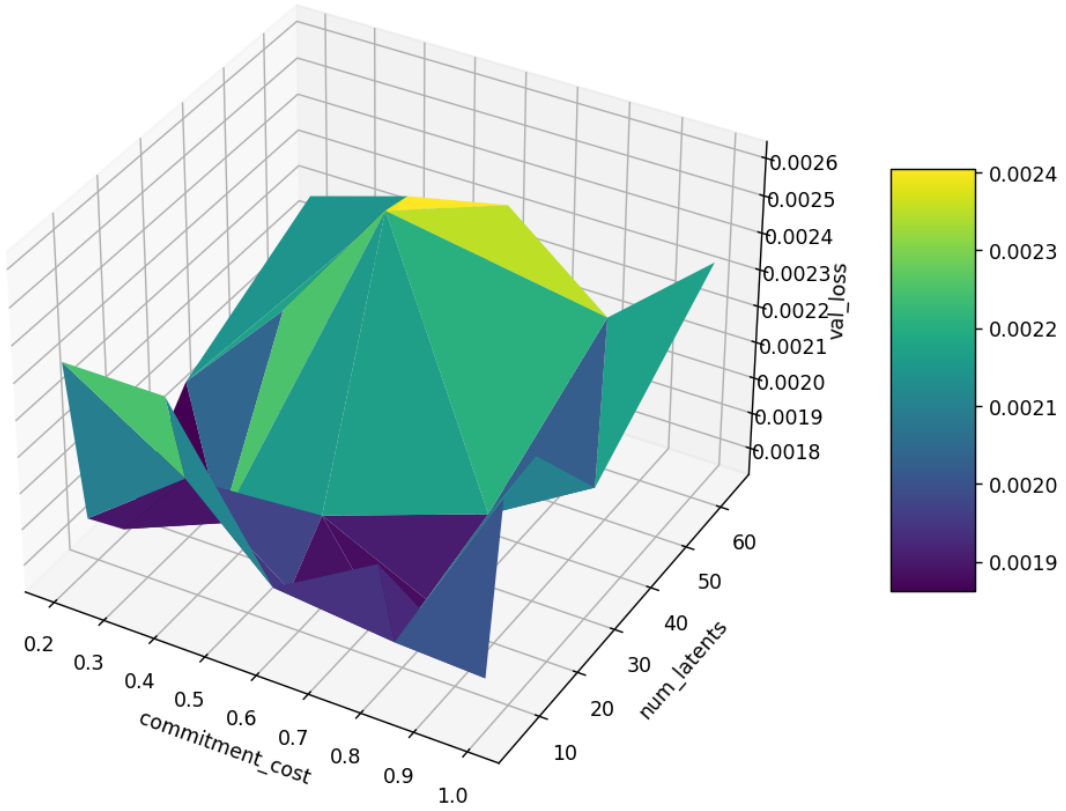
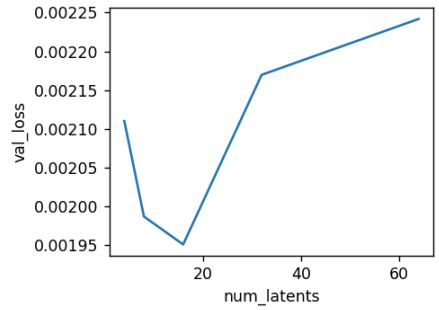
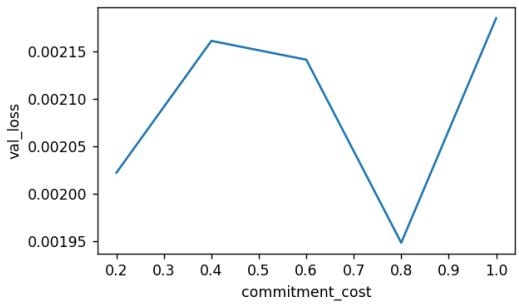


Figure 5.8: Gradient of the vector quantizer hyperparameter losses.



(a) Averaged latent size across commitment cost



(b) Averaged commitment cost across latent size

Figure 5.9: Validation loss for vector-quantizer parameter sweep

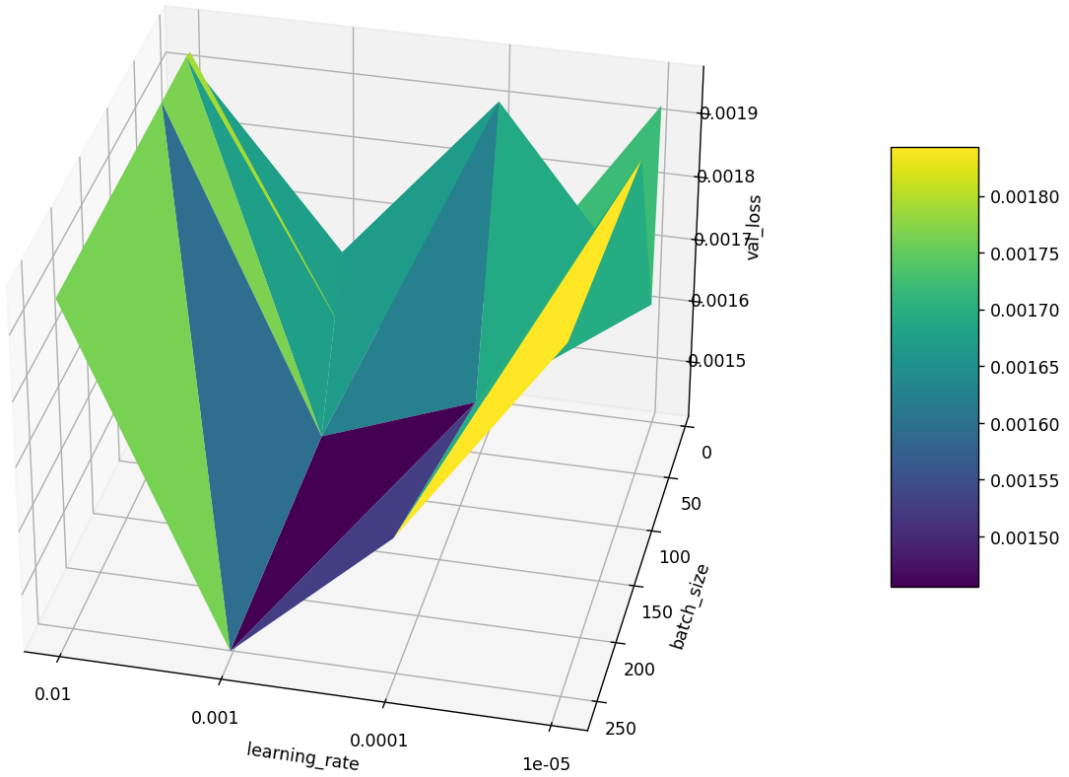
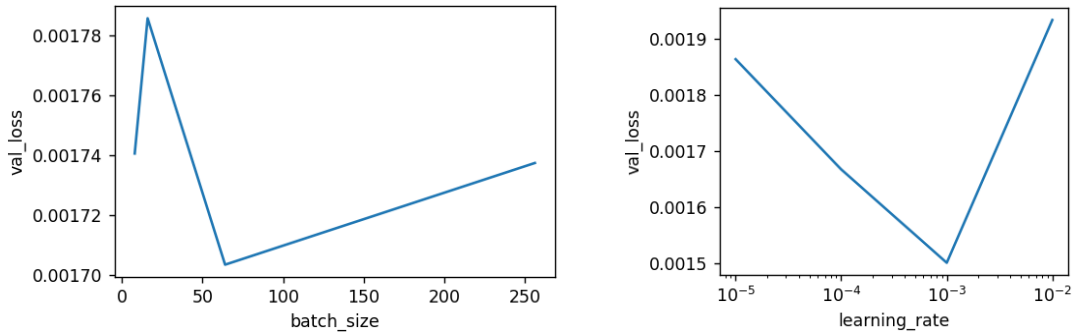


Figure 5.10: Gradient of the learning rate and batch size



(a) Averaged batch size across learning rate

(b) Averaged learning rate across batch size

Figure 5.11: Validation loss for learning rate and batch size parameter sweep

5.7 GitHub Repository

The code for the project can be found in the GitHub Repository:
`https://github.com/LukasKristensen/Master-Thesis`

6. Results

All results presented are from the test split of the loaded data set.

6.1 Baseline Performance

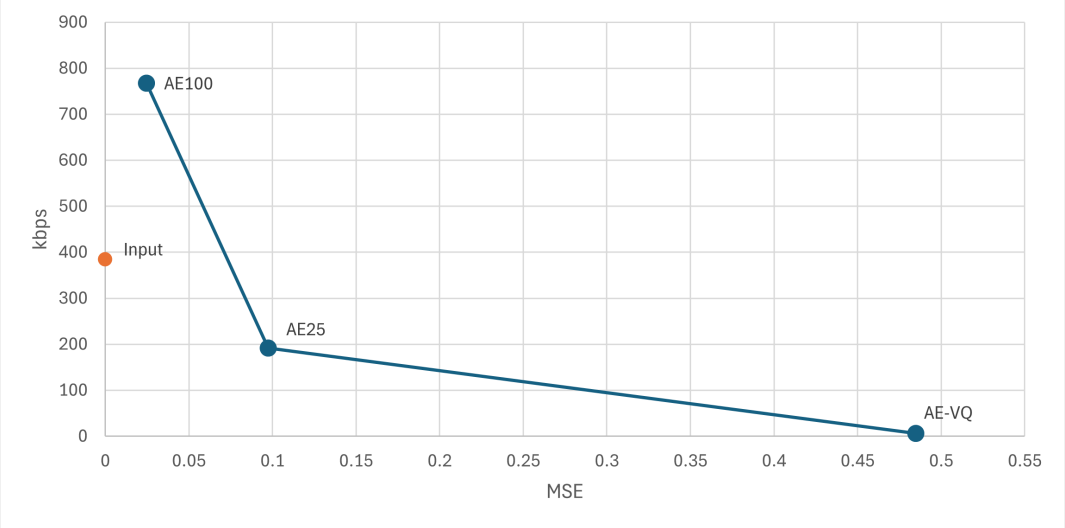


Figure 6.1: Bit-rate to MSE curve of the models without packet loss (24kbps, 0.02s frame per sample).

6.2 Inference Latency

The models were run on a personal computer rather than a performance-optimized cloud computer, which had 16 VCPUs and 16 GB RAM for a more realistic inference time.

Model	kbps	Inference time (encoding + decoding)
Model A	768	28ms
Model B	192	19ms
Model C	6	82ms

6.3 Packet Loss Curve

6.3.1 Model A - AutoEncoder 200% bit-size output

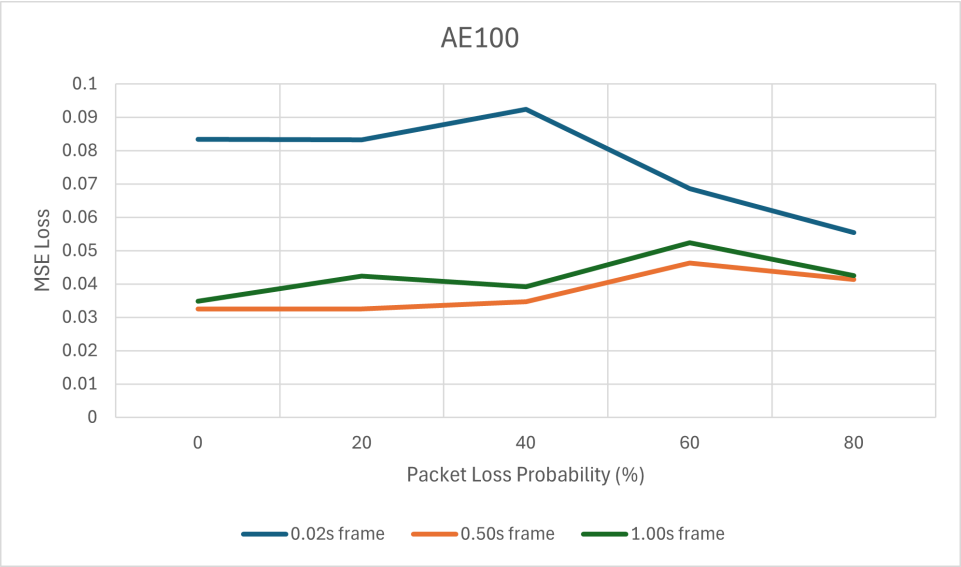


Figure 6.2: Packet Loss Probability to loss over the various frame lengths for Model A.

6.3.2 Model B - AutoEncoder 50% bit-size output

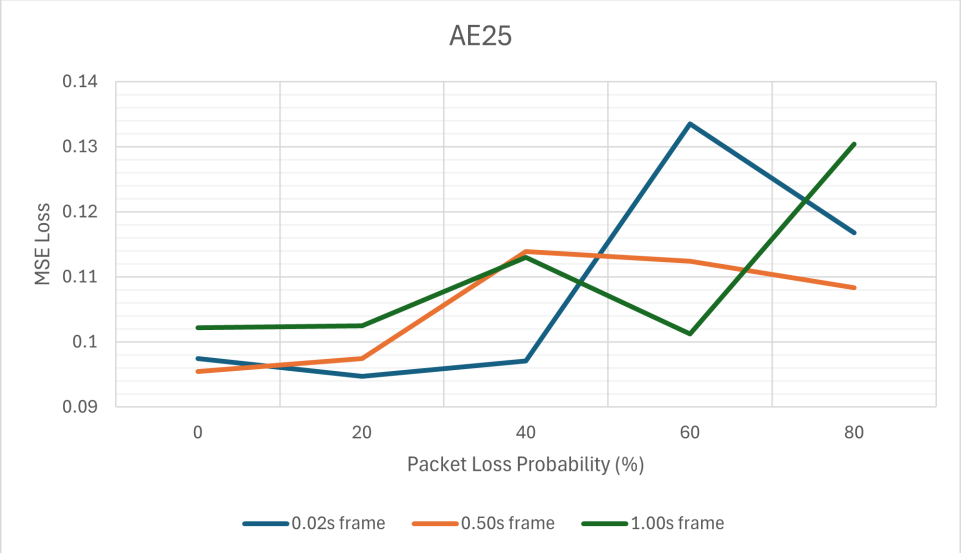


Figure 6.3: Packet Loss Probability to loss over the various frame lengths for Model B.

6.3.3 Model C - AutoEncoder VQ 1.56% bit-size output

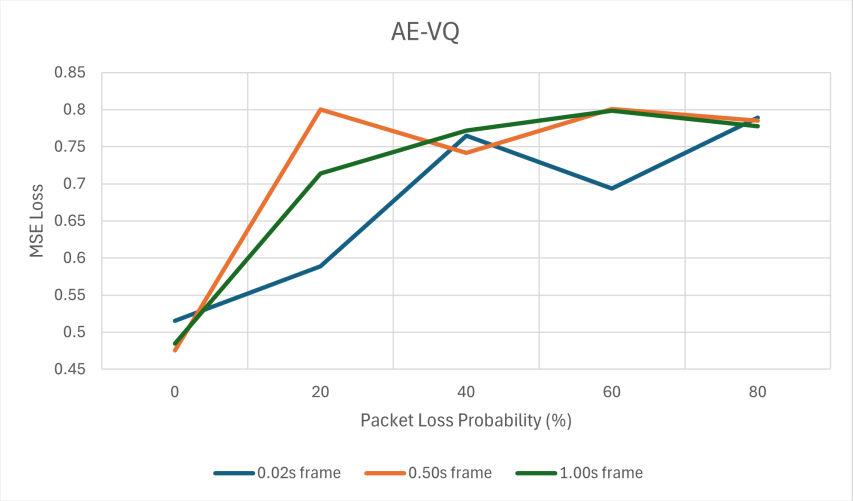
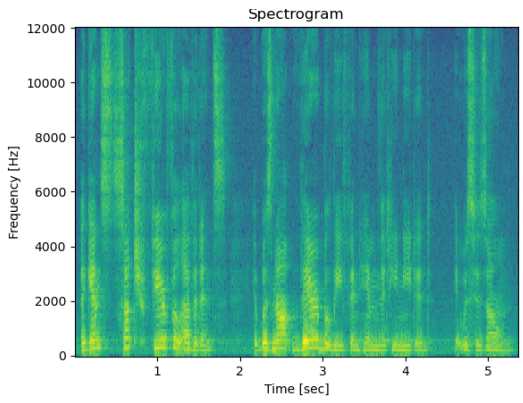


Figure 6.4: Packet Loss Probability to loss over the various frame lengths for Model C.

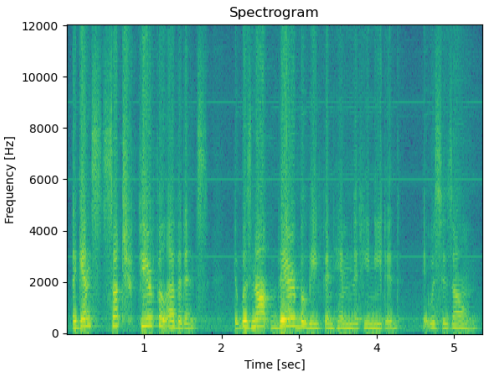
6.4 Examples from test split

In Appendix A, a sample for each model and packet loss variation can be seen in their waveform and in Appendix B, they can be seen as spectrograms. The sound clips can be accessed in the digital appendix with their ground truth. For reference, a sample of each model at 40% packet loss with 20ms frame with its ground truth can be seen:

6.4.1 Model A - AutoEncoder 200% bit-size output

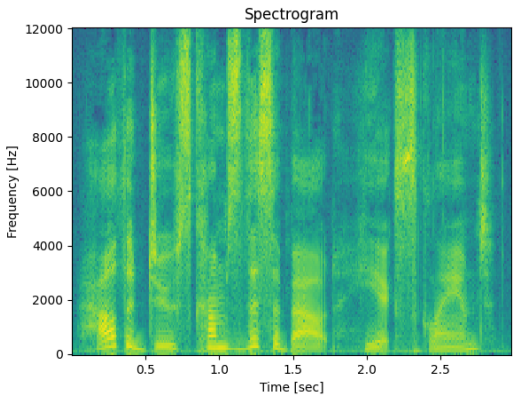


(a) Autoencoder 768kbps example ground truth

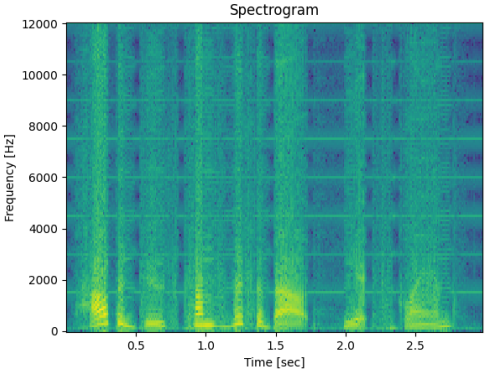


(b) Autoencoder 768kbps example with 40% Packet Loss Probability

6.4.2 Model B - AutoEncoder 50% bit-size output

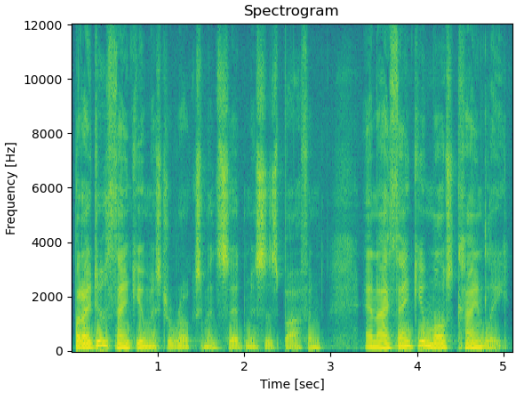


(a) Autoencoder 192kbps example ground truth

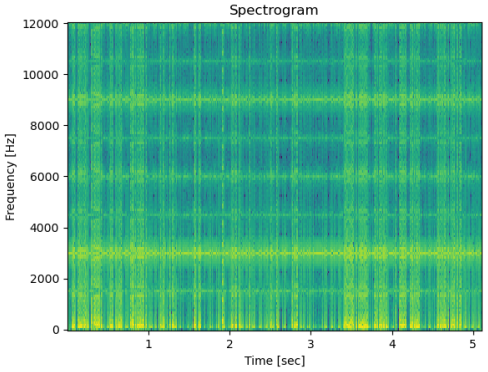


(b) Autoencoder 192kbps example with 40% Packet Loss Probability

6.4.3 Model C - AutoEncoder VQ 1.56% bit-size output



(a) Autoencoder with VQ 6kbps example ground truth



(b) Autoencoder with VQ 6kbps example with 40% Packet Loss Probability

7. Discussion

7.1 Performance

Based on the results presented in chapter 6, a model-based approach for encoding packets for packet loss robustness showed promising results, which will be discussed in this section.

7.1.1 Baseline Performance

Starting from the baseline performance as seen on Figure 6.1, the models with higher kbps outperform the lower kbps models. There was no significant benefit in the implemented auto-encoder using vector quantization. Even though it was able to capture the patterns of the sound, it did an insignificant job of capturing the depth of the information in each sample. It should be mentioned that this rate is close to the state-of-the-art of audio compression and can be difficult to outperform. The Model A with double the bit-rate as the input was measured at an MSE of 0.025 compared to Model B with half the input bit-rate at MSE 0.097. When comparing Model A to the input sample, they sound alike with some noise. When comparing Model B to the input, it sounds robotic, indicating that the model could need further optimization to improve the results.

7.1.2 Packet Loss Robustness

As seen in the results presented in section 6.3, Model A and Model B have a similar loss going from 0 to 40% packet loss. The performance fluctuates from 60% to 80% packet loss, possibly due to the random train-test splits. The random splits can affect the result as it might randomly encounter samples it has not been familiar with due to dissimilarity between the training data and test data. During the evaluation, the amount of data trained was reduced due to training time and the number of models that should be evaluated. An easy solution to preventing this issue would be to include more of the data set to expose the model to more varied data.
Besides the variation in test results across packet loss rates, there was a decent

reconstruction when applying packet loss to the packages for Model A and Model B. As seen in the spectrograms from section 6.4, Model A has a lot of similarity at 40% packet loss and sounds alike. When comparing Model B's performance at 40% packet loss, it does not perform that well but still sounds a lot like the decoded output with no package loss. Model C has an insignificant result at 40% package loss as it is not audible at all what is being said in the decoded version. When looking at the spectrogram of the model output, it only seems to recognize some of the timesteps of the recording while failing to reconstruct the information.

7.2 Interactive Methods

Since each model variation has a model for the different packet losses, it is effective for interactive communication between the sender and receiver. If the receiver experiences no packet loss, it might be better not to encode the packages to get a better sound quality and instead send the original sound stream. In contrast, if the receiver experiences a specific packet loss rate in a period, the sender can encode the packages for that rate.

7.3 Training for packet loss

During development, various approaches were considered for training for packet loss robustness. For the project's model implementations, only a single decoder was implemented for each model variation, where each model was trained for its own packet loss rate. An alternative approach was proposed to have a decoder for each variation of the possible packet amount incoming. This would mean that the encoder would have to encode enough information in the embedding for each individual decoder to reconstruct it at sound quality matching the packet loss rate as seen on Figure 7.1.

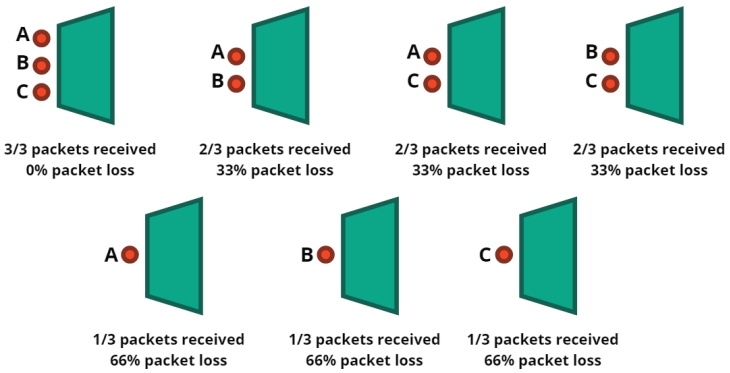


Figure 7.1: Concept of having multiple decoders for each packet loss probability. In this example, the encoder would have encoded a total of three packages: A, B and C.

For the models, it was implemented to generate new packages with the same length as the missing packages, with minus one in each index. This approach seemed to work well as the normal packages would have information in the range of zero and one.

7.3.1 Bursts of packet loss

It should be addressed that packet loss sometimes tends to occur in bursts. This means that the model encodings are only as effective for packet loss robustness as the length of the packet loss burst, meaning that if the encoding length is smaller than the burst, the packages cannot be recovered. Since the model is convolutional, it is flexible for handling multiple frame lengths, which opens up the opportunity to use the interaction between the encoder and decoder if the packet loss burst length is known. The drawback of this approach is a delayed communication flow since the encoder would have to wait for additional information.

7.4 Future Work

As the model results suggest, the package loss results are only as good as the base auto-encoder model, which indicates that if future work for the project were to be done, the main focus would be to improve the models. As proposed by SoundStream[24], one of the model architectures to investigate could be the GAN.

Additionally, investigating the benefits of using multiple decoders could be an opportunity, as proposed in section 7.3.

Finally, it could be interesting to investigate the opportunity of using the Multiple Description Coding (MDC) concept for packet encoding, where different packet losses are weighted depending on the desired sound quality for each packet loss variety.

8. Conclusion

In conclusion of the project, using models for encoding package robustness for packet loss showed promising results. Experiments were conducted on varying the frame length of the packages together with model complexity. Some evidence suggested that a larger frame length was more effective for making the packages robust. However, there was nothing conclusive as some results proved uncertain, possibly due to a lack of data inclusion during training. There was a strong correlation between the model complexity and its ability to reconstruct the input at a packet loss. The largest model got good results at packet loss rates up to the highest one tested at 80%, suggesting that this method could be a viable solution for packet loss. Since the state-of-the-art has encoded packages to even lower rates while keeping a good reconstruction, it allows future work to be conducted utilizing some of the same technology together with the project's findings.

Bibliography

[1] OpenAI (2024). *ChatGPT (3.5) [Large language model]*. (Accessed 31/03/2024). URL: https://chat.openai.com.

[2] Jafar Ababneh and Omar Almomani. "Survey of Error Correction Mechanisms for Video Streaming over the Internet". In: *International Journal of Advanced Computer Science and Applications* 5 (Mar. 2014), pp. 148–154. DOI: 10.14569/IJACSA.2014.050322.

[3] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[4] Tom Backstrom. *Speech Coding: with Code-Excited Linear Prediction*. eng. 1st ed. 2017 edition. Signals and Communication Technology. Netherlands: Springer Nature, 2017. Chap. 12. ISBN: 3319502042.

[5] Lukas Biewald. *Experiment Tracking with Weights and Biases*. Software available from wandb.com. (Accessed 31/03/2024). 2020. URL: https://www.wandb.com/.

[6] Tom Bäckström et al. *Introduction to Speech Processing*. 2nd ed. 2022. DOI: 10.5281/zenodo.6821775. URL: https://speechprocessingbook.aalto.fi.

[7] *CLAAUDIA Research Data Services*. (Accessed 28/04/2024). URL: https://www.strato-docs.claaudia.aau.dk/.

[8] Cloudflare. *TCP vs UDP Communication*. (Accessed: 11/05/2024). URL: https://www.cloudflare.com/learning/ddos/glossary/user-datagram-protocol-udp/.

[9] Papers With Code. *LibriTTS Dataset*. (Accessed 05/03/2024). URL: https://paperswithcode.com/dataset/libritts.

[10] Jayanta Debnath, Newaz Rahim, and Wai-keung Fung. "A Modified Vector Quantization Based Image Compression Technique Using Wavelet Transform". In: July 2008, pp. 171 –176. ISBN: 978-1-4244-1820-6. DOI: 10.1109/IJCNN.2008.4633785.

[11] Alexandre Défossez et al. *High Fidelity Neural Audio Compression*. 2022. arXiv: 2210.13438 [eess.AS].

[12] Kris Koishigawa (freeCodeCamp). *TCP vs. UDP — What's the Difference and Which Protocol is Faster?* (Accessed: 10/05/2024). URL: https://www.freecodecamp.org/news/tcp-vs-udp/.

[13] GeeksForGeeks. *Introduction to Convolution Neural Network*. (Accessed: 13/05/2024). URL: https://www.geeksforgeeks.org/introduction-convolution-neural-network/.

[14] geeksforgeeks. *TCP 3-Way Handshake Process*. (Accessed: 11/05/2024). URL: https://www.geeksforgeeks.org/tcp-3-way-handshake-process/.

[15] GeeksForGeeks. *What is Transposed Convolutional Layer?* (Accessed: 13/05/2024). URL: https://www.geeksforgeeks.org/what-is-transposed-convolutional-layer/.

[16] GitHub. *Github Copilot*. (Accessed 31/03/2024). URL: https://github.com/features/copilot.

[17] Google. *Lyra*. (Accessed 12/03/2024). URL: https://github.com/google/lyra.

[18] IzoTope Griffin Brown. *Digital Audio Basics: Audio Sample Rate and Bit Depth*. (Accessed: 16/05/2024). URL: https://www.izotope.com/en/learn/digital-audio-basics-sample-rate-and-bit-depth.html.

[19] Rui Jin and Qiang Niu. "Automatic Fabric Defect Detection Based on an Improved YOLOv5". In: *Mathematical Problems in Engineering* 2021 (Sept. 2021), pp. 1–13. DOI: 10.1155/2021/7321394.

[20] lucidrains. *vector-quantize-pytorch*. (Accessed: 16/05/2024). URL: https://github.com/lucidrains/vector-quantize-pytorch.

[21] Vassil Panayotov et al. "Librispeech: An ASR corpus based on public domain audio books". In: *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2015, pp. 5206–5210. DOI: 10.1109/ICASSP.2015.7178964.

[22] Anh H. Reynolds. *Convolutional Neural Networks (CNNs)*. (Accessed: 14/05/2024). URL: https://anhreynolds.com/blogs/cnn.html.

[23] SpeechZone. *Bitrate of a waveform*. (Accessed: 16/05/2024). URL: https://speech.zone/bitrate.

[24] Neil Zeghidour et al. *SoundStream: An End-to-End Neural Audio Codec*. 2021. arXiv: 2107.03312 [cs.SD].

[25] Heiga Zen et al. *LibriTTS: A Corpus Derived from LibriSpeech for Text-to-Speech*. (Accessed on 05/03/2024). 2019. arXiv: 1904.02882 [cs.SD].

A. Appendix - Model Outputs Waveform

In the figures, the blue waveform is the original input, and the orange waveform is the model output.

A.1 Autoencoder 768kbps trained on 20ms frames (Model A)

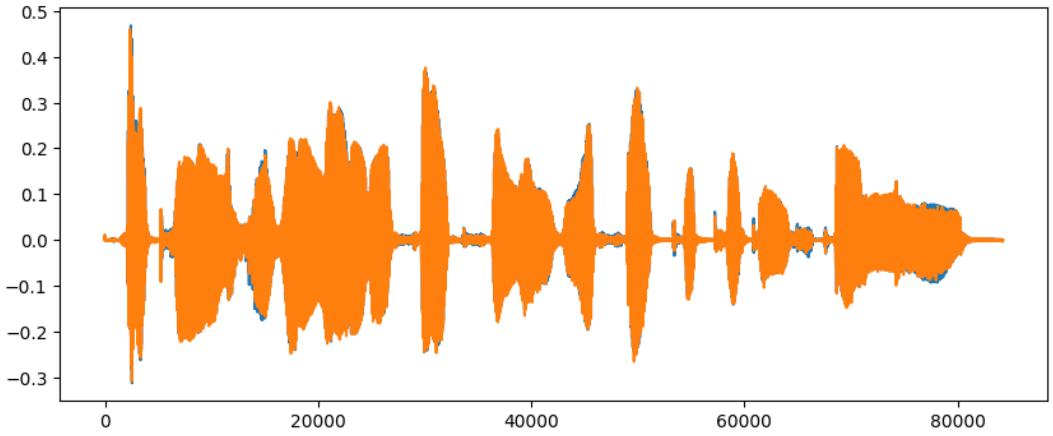


Figure A.1: Autoencoder 768kbps with 0% Packet Loss Probability

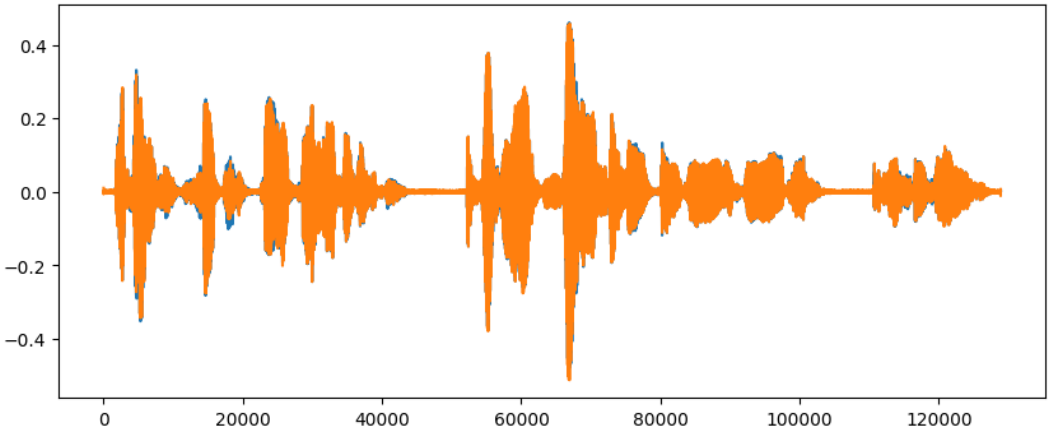


Figure A.2: Autoencoder 768kbps with 20% Packet Loss Probability

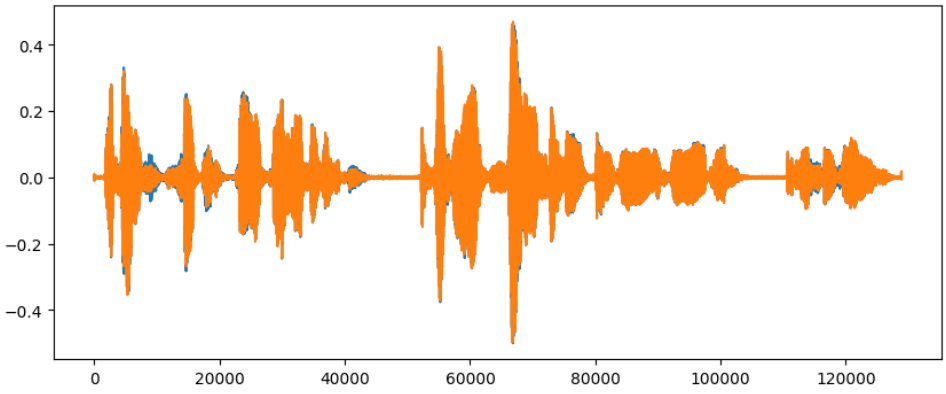


Figure A.3: Autoencoder 768kbps with 40% Packet Loss Probability

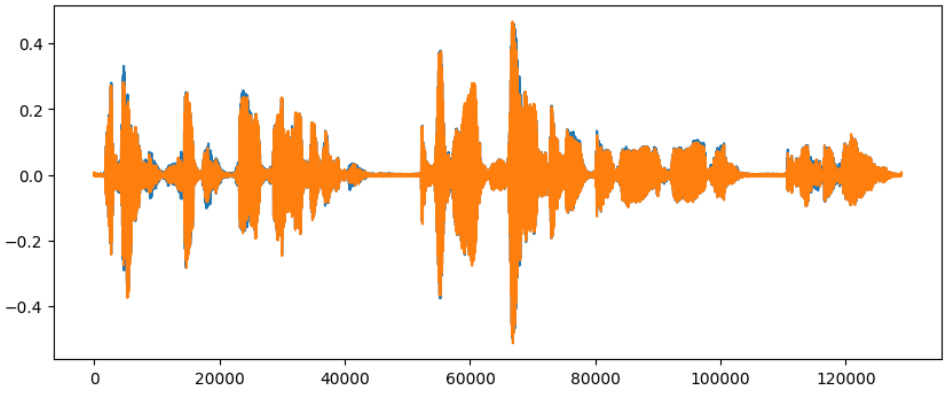


Figure A.4: Autoencoder 768kbps with 60% Packet Loss Probability

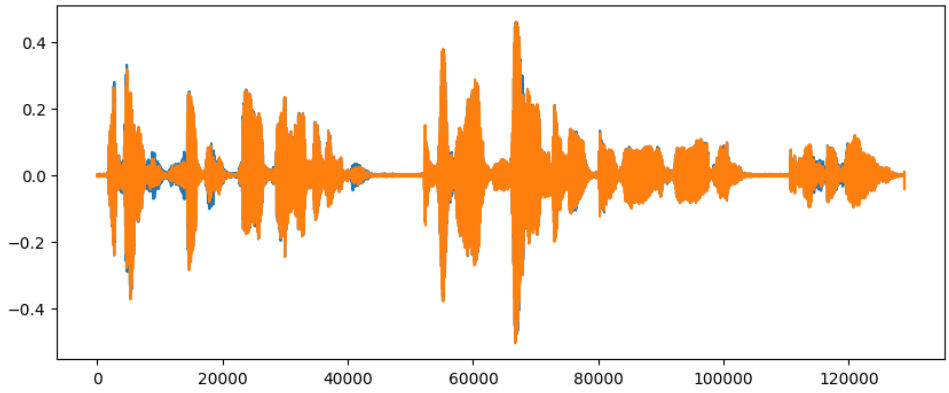


Figure A.5: Autoencoder 768kbps with 80% Packet Loss Probability

A.2 Autoencoder 192kbps trained on 20ms frames (Model B)

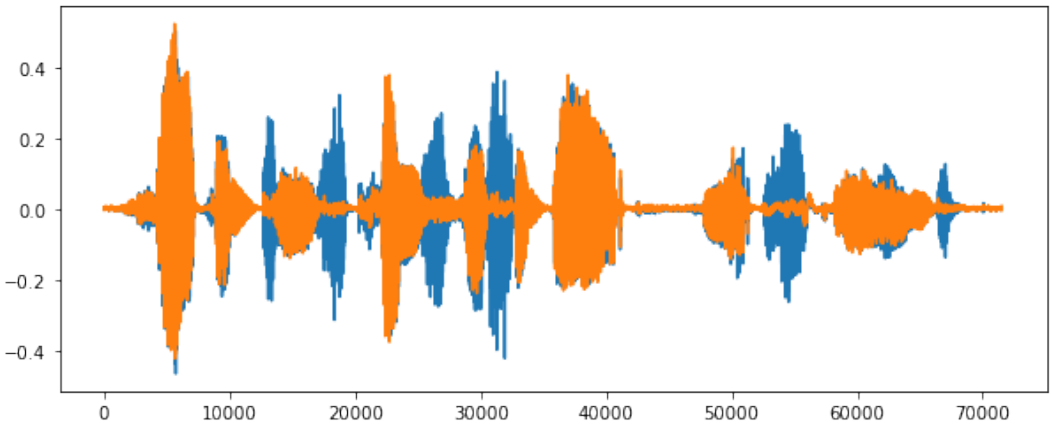


Figure A.6: Autoencoder 192kbps with 0% Packet Loss Probability

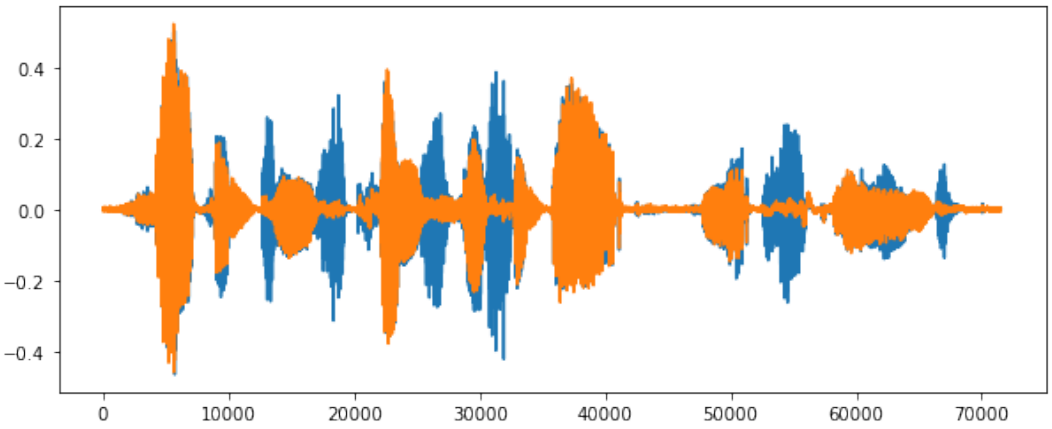


Figure A.7: Autoencoder 192kbps with 20% Packet Loss Probability

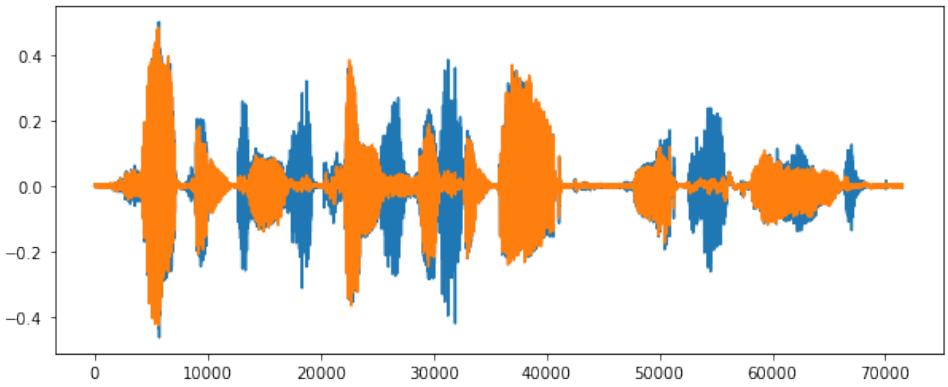


Figure A.8: Autoencoder 192kbps with 40% Packet Loss Probability

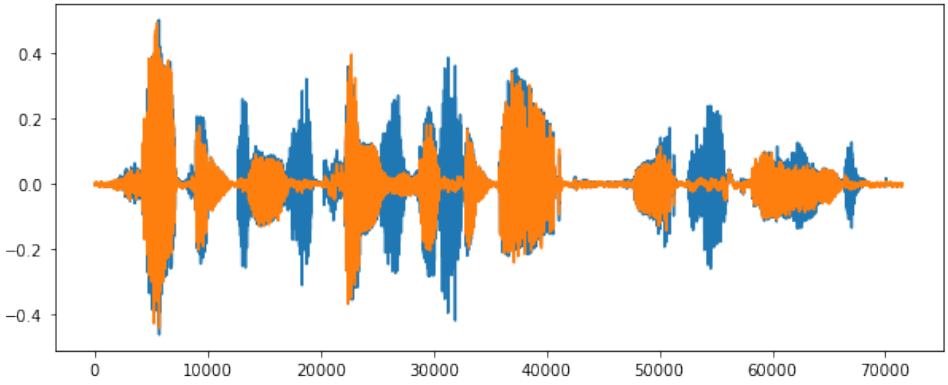


Figure A.9: Autoencoder 192kbps with 60% Packet Loss Probability

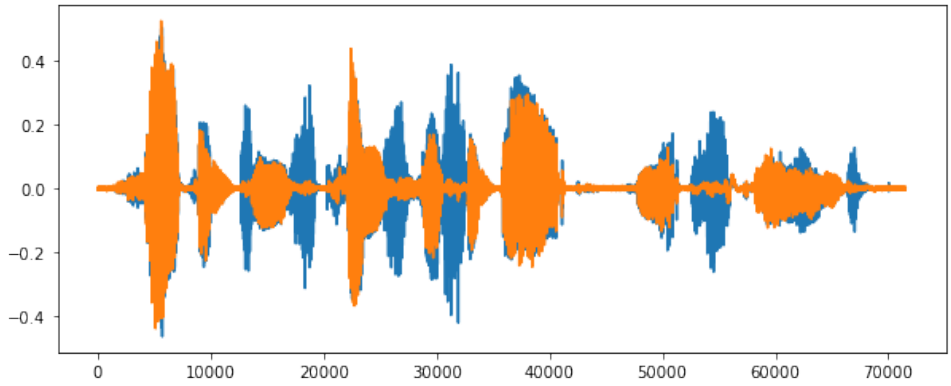


Figure A.10: Autoencoder 192kbps with 80% Packet Loss Probability

A.3 Autoencoder with VQ 6kbps trained on 20ms frames (Model C)

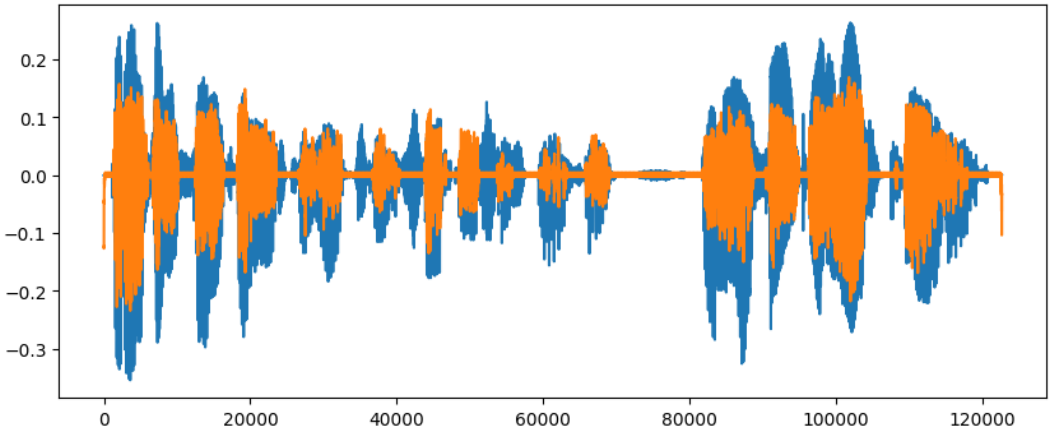


Figure A.11: Autoencoder with VQ 6kbps with 0% Packet Loss Probability

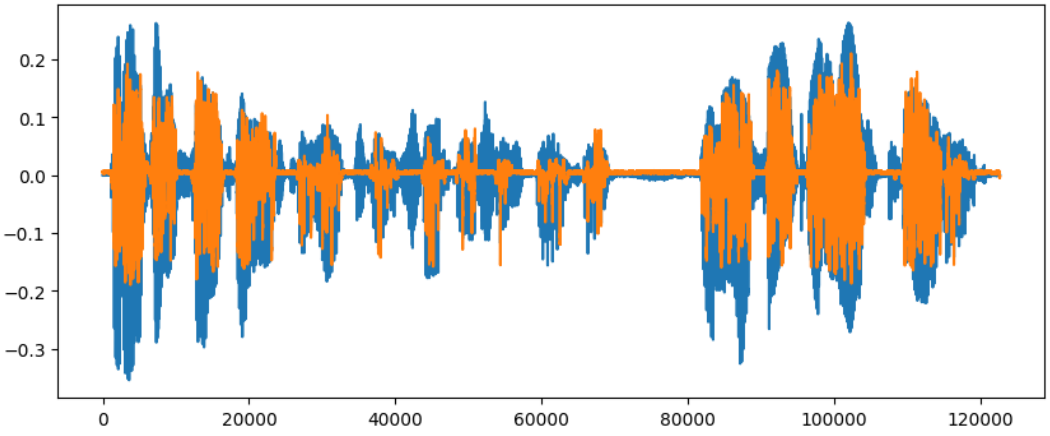


Figure A.12: Autoencoder with VQ 6kbps with 20% Packet Loss Probability

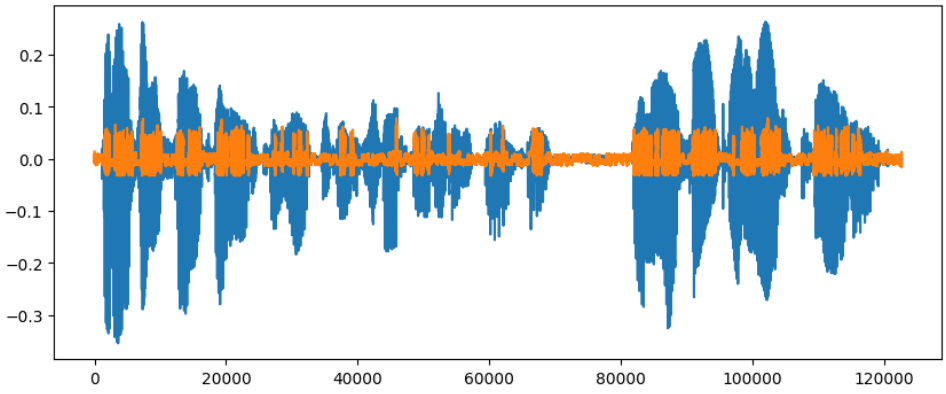


Figure A.13: Autoencoder with VQ 6kbps with 40% Packet Loss Probability

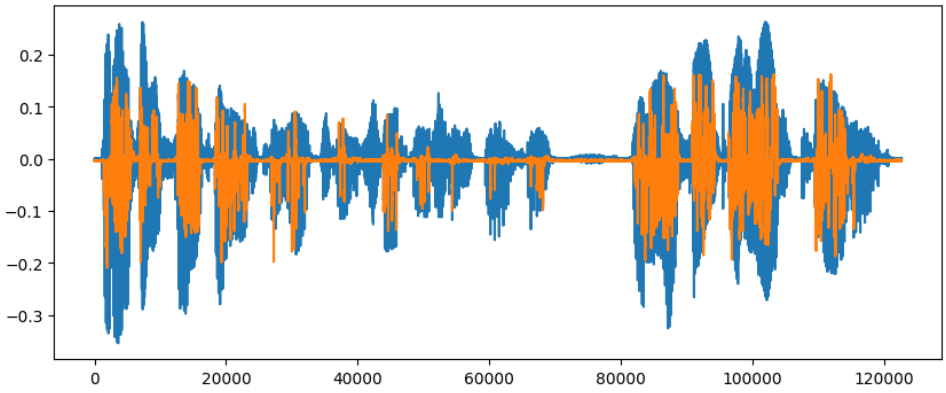


Figure A.14: Autoencoder with VQ 6kbps with 60% Packet Loss Probability

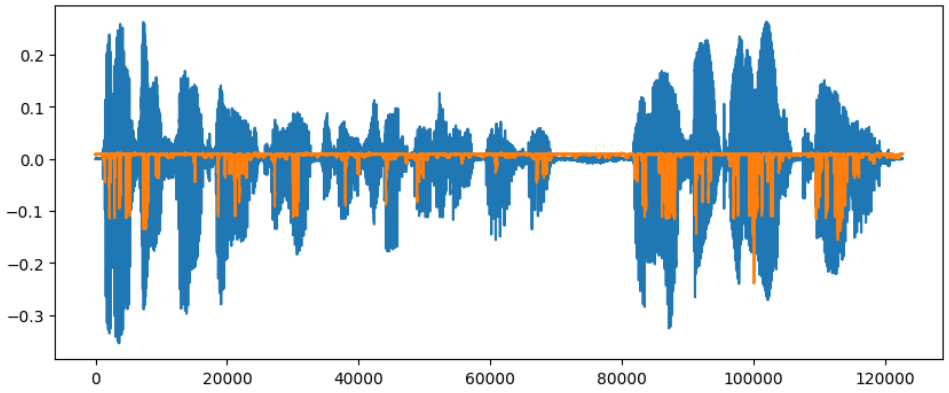
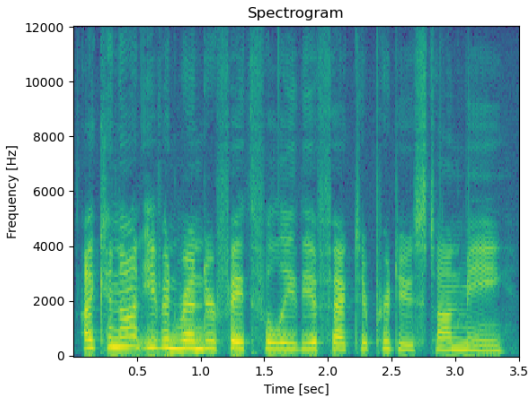


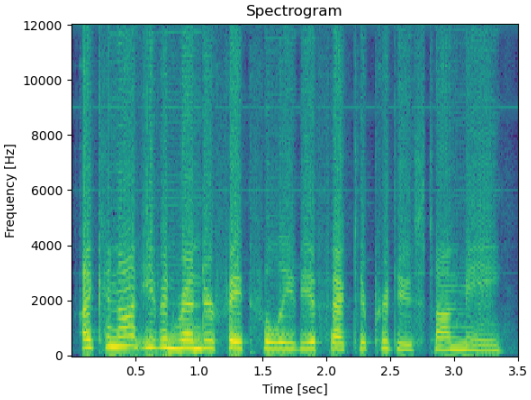
Figure A.15: Autoencoder with VQ 6kbps with 80% Packet Loss Probability

B. Appendix - Model Outputs Spectrogram

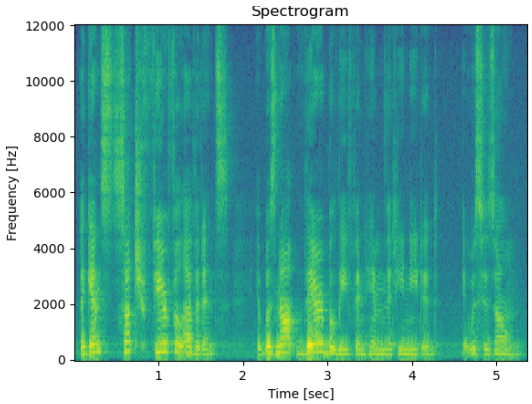
B.1 Autoencoder 768kbps trained on 20ms frames (Model A)



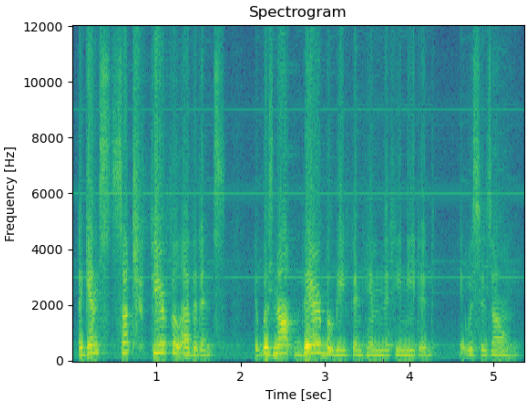
(a) Autoencoder 768kbps ground truth for 0% packet loss



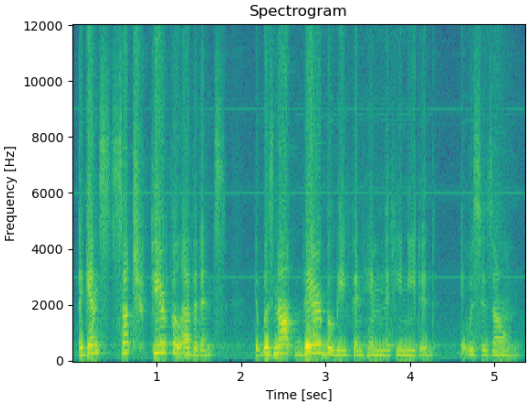
(b) Autoencoder 768kbps with 0% Packet Loss Probability



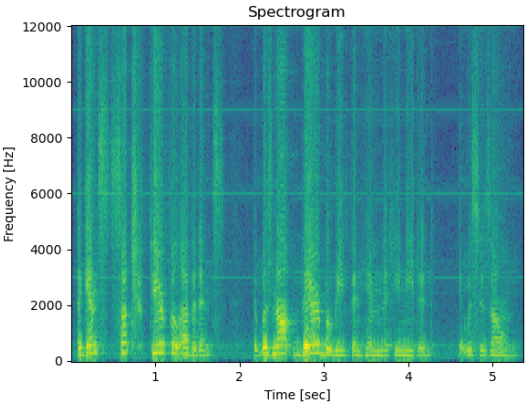
(a) Autoencoder 768kbps ground truth for 20% - 80% packet loss



(b) Autoencoder 768kbps with 20% Packet Loss Probability



(a) Autoencoder 768kbps with 40% Packet Loss Probability



(b) Autoencoder 768kbps with 60% Packet Loss Probability

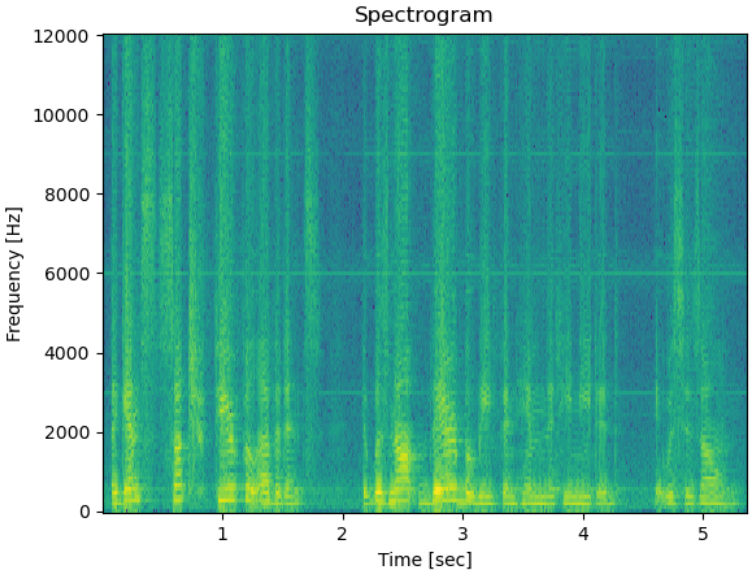
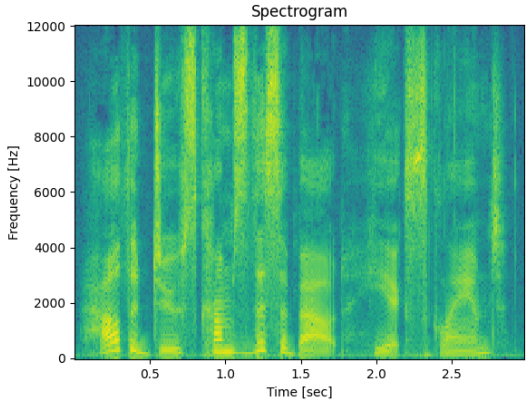
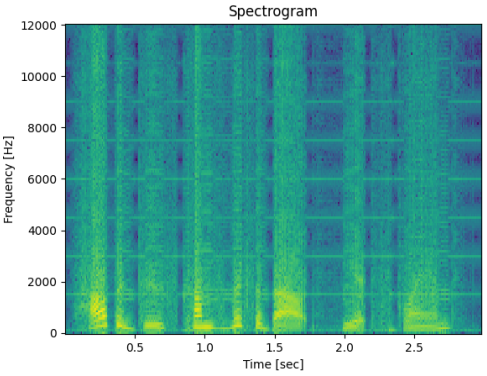


Figure B.4: Autoencoder 768kbps with 80% Packet Loss Probability

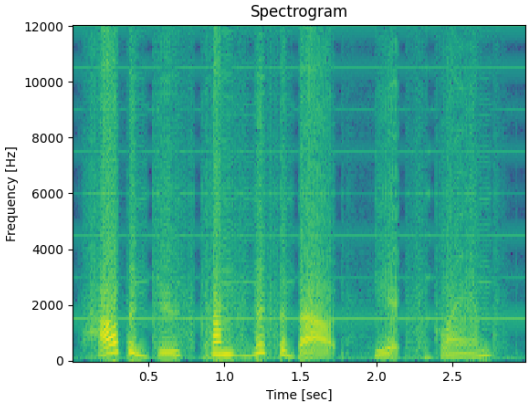
B.2 Autoencoder 192kbps trained on 20ms frames (Model B)



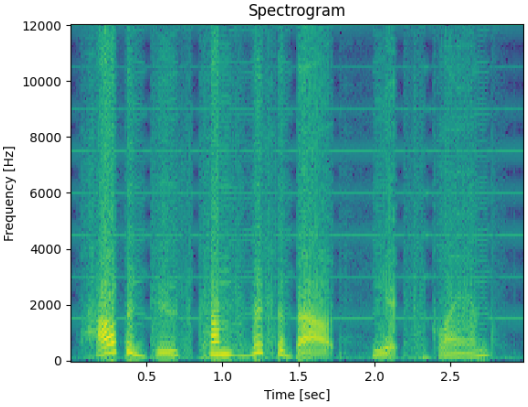
(a) Autoencoder 192kbps ground truth



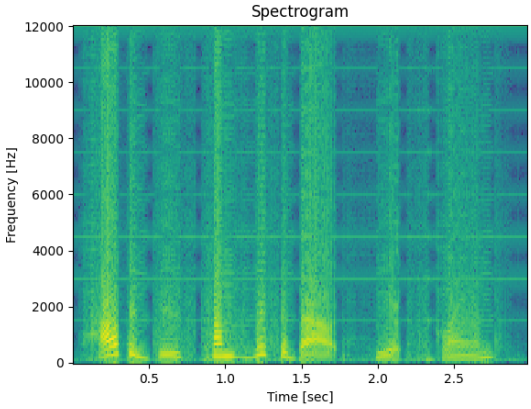
(b) Autoencoder 192kbps with 0% Packet Loss Probability



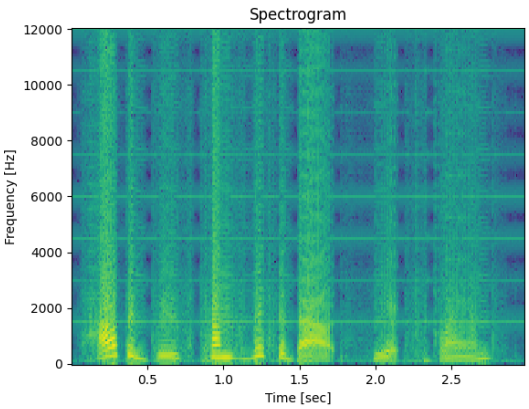
(a) Autoencoder 192kbps with 20% Packet Loss Probability



(b) Autoencoder 192kbps with 40% Packet Loss Probability

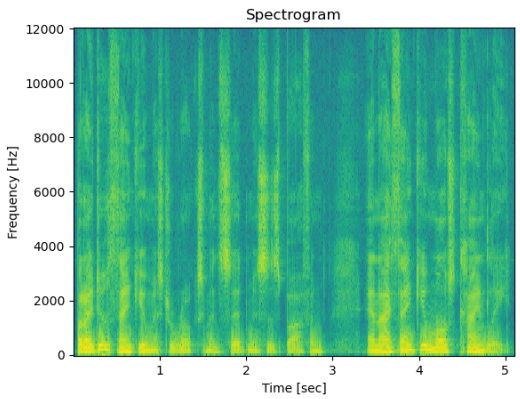


(a) Autoencoder 192kbps with 60% Packet Loss Probability

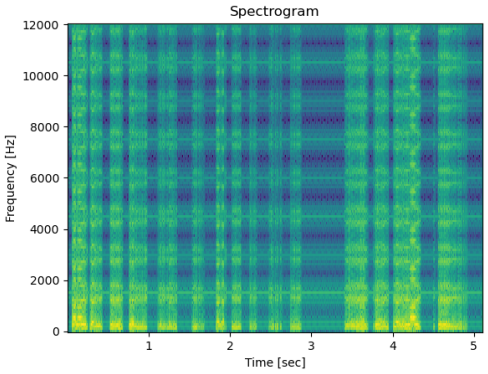


(b) Autoencoder 192kbps with 80% Packet Loss Probability

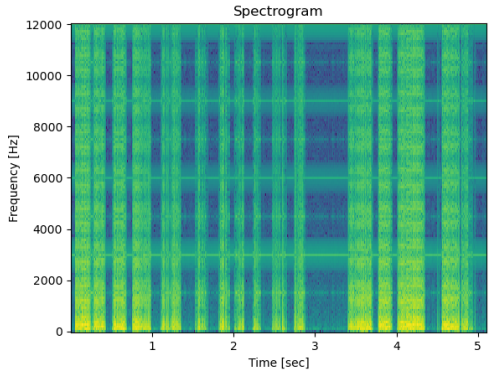
B.3 Autoencoder with VQ 6kbps trained on 20ms frames (Model C)



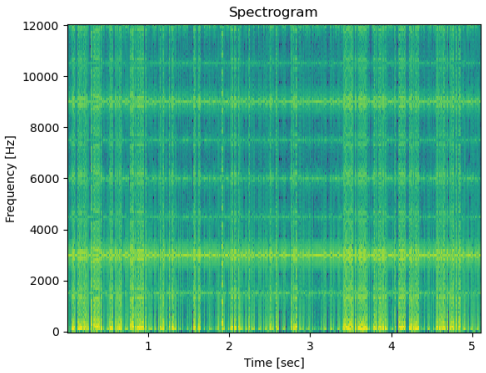
(a) Autoencoder with VQ 6kbps ground truth



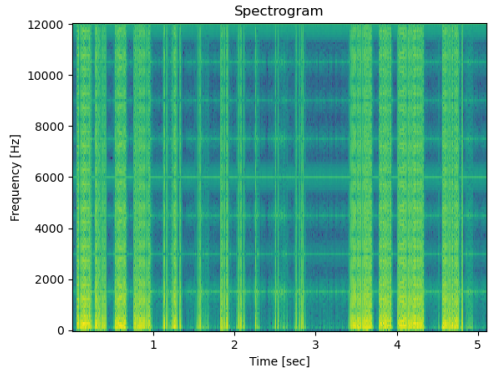
(b) Autoencoder with VQ 6kbps with 0% Packet Loss Probability



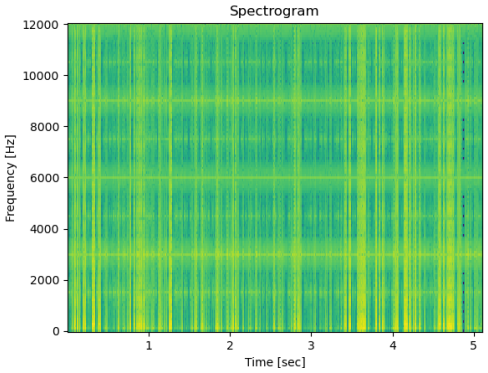
(a) Autoencoder with VQ 6kbps with 20% Packet Loss Probability



(b) Autoencoder with VQ 6kbps with 40% Packet Loss Probability



(a) Autoencoder with VQ 6kbps with 60% Packet Loss Probability



(b) Autoencoder with VQ 6kbps with 80% Packet Loss Probability