Background: Duke Kunshan University (DKU) is an interdisciplinary institution that grants dual undergraduate degrees, an MOE Chinese degree and a degree from Duke University in Durham, United States. The principal structure of DKU majors is robustly interdisciplinary. No student confines their study to a single discipline (for example, biology or economics). Instead, all students engage in broad inquiry related to a subject or question (for example, political economy or global health) and take a wide variety of courses related to that area (for example, in public policy, history, ethics, or economics). As a result, our graduates are prepared to engage in a wide variety of inquiries using multiple methodologies to address complex issues that require interdisciplinary approaches. This has implications for our vision and expectations of undergraduate theses and design projects, which reflect this broad interdisciplinary training. At DKU, every student completes a two-year project known as signature work which consists of multiple interconnected parts including thematic courses, experiential learning, capstones, and a final product. It seeks to integrate students' interdisciplinary educational experience and culminates in the creation of a product in a scholarly, creative, or applied nature in leu of an undergraduate thesis or design required by JED. Because DKU encourages students to cultivate their independence and creativity as one of its institutional student learning outcomes, the student-led signature work projects often reflect students' own particular interdisciplinary interests and training. In addition, signature work has an intensive emphasis on problem-solving and skill-development which is much needed for any interdisciplinary inquiry; thus, students' final products are evidence of transferrable skills that students have acquired and demonstrated through the 2-year program, rather than content knowledge narrowly defined by disciplinary training. In sum, while the Chinese major declared with any given student might be construed narrowly, the experience of our students is much broader—and intentionally so. This is a distinctive feature of our curriculum, and this distinctiveness results in broadly interdisciplinary submissions from our graduates' submitting theses or design projects. We have designed this to prepare our students for a wide variety of graduate programs in China and the West, where interdisciplinary training is a competitive advantage.

UTILIZING DISCRETE TOKENS AND LANAUGE MODELS FOR TARGET SPEAKER EXTRACTION

by

Beilong Tang

Signature Work Product, in partial fulfillment of the
Duke Kunshan University Undergraduate Degree Program

March 4, 2025

Signature Work Program
Duke Kunshan University

APPROVALS

Mentor: Ming Li, Division of Natural and Applied Sciences

CONTENTS

ABSTRACT

Target Speaker Extraction (TSE) aims to isolate a specific speaker's voice from a multi-speaker mixture. With the rapid advancements of generative models—particularly language models in Natural Language Processing and Computer Vision—there is growing interest in applying them to Speech Processing. However, little research has explored the use of discrete tokens and language models for TSE. In this work, we investigate the effectiveness of discrete tokens derived from both self-supervised learning (SSL) models and neural audio codecs for TSE. Specifically, we use Kmeans clustering to discretize multiple layers of WavLM for SSL-based approaches and employ FunCodec with a LauraGPT decoder-only transformer backbone for neural audio codec-based methods. Experimental results demonstrate that these generative approaches achieve high speech quality while maintaining competitive speech intelligibility. Notably, neural audio codecs significantly outperform Kmeans-based models in speaker similarity, likely due to reduced information loss during discretization. However, a performance gap remains between discrete and continuous methods, particularly in intelligibility and speaker similarity. To bridge this gap, future research will focus on refining discretization techniques and exploring hybrid models that better preserve both acoustic and linguistic features.

ACKNOWLEDGEMENTS

Firstly, I would like to express my gratitude to SRS for providing me with this research opportunity. I am also deeply thankful to Dr. Ming Li for his invaluable guidance throughout my work. Additionally, I would like to extend my thanks to the Kunshan SuperComputing Network (SCNet) for generously providing the necessary computing resources.

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

1.1 Background

Speech is a cornerstone of human communication, and with the advancements in machine learning, it has become integral to numerous applications. For instance, Speech Recognition [**speech_recognition**] converts spoken content into text, enabling tasks like Machine Translation [**machine_translation**]. Speaker Verification [**speaker_verification**] identifies individuals based on their voice, while Speech Anonymization [**speech_anno**] removes speaker identity while retaining contextual and emotional information. Additionally, Speaker Diarization [**cheng2024sequence**] segments an audio stream into homogeneous parts based on speaker identity. A common requirement across these tasks is the need for relatively clean speech input.

However, real-world speech recordings are often far from ideal. Environmental factors, such as background noise at a party or high reverberation in a room, can significantly degrade audio quality. Humans possess the remarkable ability to focus on a specific speaker amidst noise, a phenomenon known as the Cocktail Party Effect [**cocktail**]. Unfortunately, machines struggle to replicate this capability.

Noisy inputs, which may include background noise or overlapping speakers, can severely impair the performance of downstream tasks like Speech Recognition [**speech_recognition_noisy**]. Addressing these challenges is crucial for improving the robustness and accuracy of speech-based systems in real-world scenarios.

Based on the recording environment, the tasks are divided into three subtasks:

- Speech Enhancement: filtering out targeted speech from background noise.
- Speech Separation: separating every speaker's speech in a mixture speech with multiple speakers.
- Target Speaker Extraction: isolating the speech of a specific speaker of interest while filtering out all other speech

Due to the scope of this work, I will focus solely on Target Speaker Extraction in Deep Learning, with an emphasis on single-channel speech.

1.2 Target Speaker Extraction (TSE)

Target Speaker Extraction (TSE) aims to extract the clear speech of a target speaker from a noisy and mixed speech signal with the aid of prior information about the target speaker. It has a

wide applications including:

- Human-Computer Interaction: In scenarios such as smart assistants and smart homes, target speaker extraction technology can help devices more accurately recognize user commands, especially in situations where multiple people are speaking simultaneously or there is significant background noise. By isolating the target user's speech, devices can respond more precisely to user needs, enhancing the interaction experience.
- Multi-Person Meetings: In multi-person meeting scenarios, target speaker extraction technology can assist in automatically recording and transcribing the speech of specific speakers, improving the accuracy and efficiency of meeting minutes. Additionally, this technology can be used for multi-target speaker enhancement in remote meetings, ensuring that each participant's speech is clearly audible.
- Speech Recognition and Translation: In speech recognition and translation systems, target speaker extraction technology can help systems more accurately recognize and translate the speech of target speakers, particularly in multi-speaker conversations or noisy environments. This is of great significance for applications such as cross-language communication and real-time translation.

In TSE, the three main categories of prior auxiliary information are audio, spatial, and visual clues. Audio clues typically refer to additional speech from the same speaker. Visual clues can include the lip or facial movements of the target speaker. Due to the scope of this work, we will focus solely on audio-based auxiliary information.

1.2.1 Problem Definition

Assume we have a single-channel mixture speech $x(t)$ composed of C speakers $s_1(t), \dots, s_C(t) \in \mathbb{R}^{1\times T}$, where

$$x(t) = \sum_{i=1}^{C} s_i(t) \tag{1.1}$$

We also have reference speech information s_{ref} as auxiliary information regarding the target speaker $s_{\text{target}}(t)$ in the mixture. Our goal is to build a deep learning model G_θ to extract the target speaker's speech, utilizing the auxiliary information, defined by the following model:

$$\hat{s}_{\text{target}}(t) = G_\theta(x(t), s_{\text{ref}}) \tag{1.2}$$

where $\hat{s}_{\text{target}}(t)$ is the generated target speaker's speech, and θ are our model's parameters.

1.3 Discriminative Models

Current models for Target Speaker Extraction (TSE) are predominantly discriminative models, where they aim to minimize the distance between the estimated speech and the clean speech signals [**luo2019conv**, **spex_plus**, **sef_net**, **sepformer**].

One of the popular discriminative model frameworks utilize masking strategies[**luo2019conv**, **spex_plus**, **sef_net**, **sepformer**]. The high-level structure is demonstrated in **Figure 1.1**. The Encoder usually uses Conformer [**gulati2020conformer**] or Convolutional Neural Network (CNN) to extract a high dimensional speech representations on noisy speech and the reference speech, resulting in E_{ref} and E_{mix}, respectively. The Separation will take E_{ref} and E_{mix} and produce a masked embedding E_{mask}. It is later element-wise multiplied with E_{mix}, and the

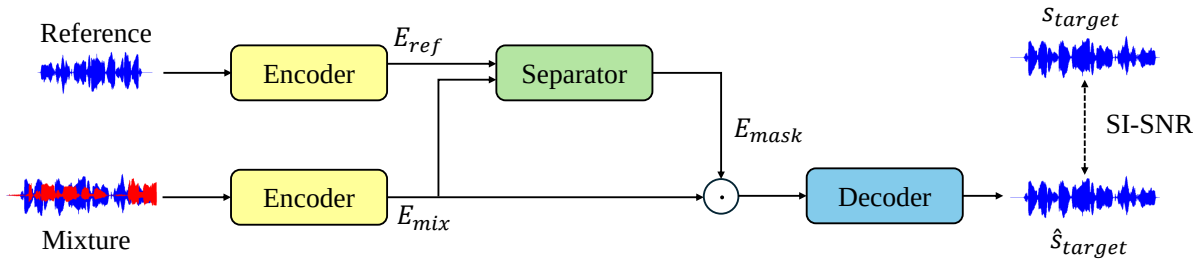

Figure 1.1: Masking Discriminative TSE Model Framework. ⊙ stands for element-wise multiplication.

output will be passed through a decoder to construct the estimated target speaker speech $\hat{s}_{target}$. The loss function is the Scale-Invariant Signal-to-Noise Ratio (SI-SNR) with the ground truth target clean speech s_{target}. This end-to-end neural network will train all the parameters of the Encoder, Separator, and Decoder.

Scale-Invariant Signal-to-Noise Ratio (SI-SNR) is defined as

$$\text{SI-SNR}(\hat{(s)}, s) = 10 \log_{10} \frac{||\alpha s||^2}{||\hat{s} - \alpha s||^2} \tag{1.3}$$

where s is the clean signal and $\hat{s}$ is the estimated signal. α is a scaling factor defined as

$$\alpha = \frac{\langle \hat{s}, s \rangle}{||s||^2} \tag{1.4}$$

The SI-SNR formula aligns the estimated speech with the ground-truth speech frame by frame.

This masked discriminative approach essentially learns a direct mapping from the mixture speech to clean speech. However, several challenges arise with this method. First, it often struggles to generalize to unseen data and can introduce unwanted noise and distortion [**distortion**], particularly when the input data is highly corrupted or of low quality. Mapping directly from such "garbage" data to clean speech may not be the optimal approach.

Moreover, the SI-SNR loss is minimized frame by frame, which can be suboptimal for overall speech quality. In some cases, enforcing frame-by-frame alignment does not necessarily lead to perceptual improvements, as evidenced by subjective listening tests [**ma2011snr**].

Additionally, because this approach applies a masking strategy to the mixture, the output may still contain residual interference from the competing speaker, often manifesting as subtle background noise. This observation is based on my own evaluations, aligning with findings in prior studies on masking-based separation methods [**Williamson2016ComplexRM**].

1.4 Generative Models

Unlike discriminative models, which rely on masking strategies to classify data points, generative models learn the underlying distribution of the data to generate new, similar data. This approach is more complex and challenging than discriminative models. Discriminative models simply focus on classifying whether the data points belong to the target distribution, while generative models learn the target distribution of data points first and sample from it [**gen2021brief**].

The four main types of the generative models are:

- Variational Autoencoders (VAE) [**kingma2013auto**], which learn a distribution over the data and generate samples by drawing from this distribution.
- Generative Adversarial Networks (GAN) [**goodfellow2020generative**], which train a generator and a discriminator in tandem, with the two models competing against each other to achieve optimal generation results.
- Large Language Models (LLM), sequence-to sequence models which are typically characterized by a vast number of parameters and can perform a wide range of tasks. LLMs are commonly divided into three categories: Encoder-only, Decoder-only, and Encoder-Decoder. Examples of decoder-only LLMs include DeepSeek and ChatGPT [**chatgpt**].
- Diffusion Models [**diffusion**], which generate data through a gradual process of adding noise to the data and then reversing this process, iteratively denoising to generate realistic samples. These models have gained prominence in image and speech generation tasks.

Assume the dataset is x and the correct label is y. For discriminative models, we are focus on learning the conditional probability:

$$P(y \mid x)$$

which represents the probability of the label given the input x x. In contrast, generative models aim to learn the joint distribution of y and x, i.e., they estimate:

$$P(y, x)$$

This requires the model to be adept at estimating both the dataset and the target distributions, making generative models more complex and versatile.

With the increasing computational power, generative models have become increasingly prevalent in Speech Enhancement tasks. For example, [**se_gan**] uses GANs, [**se_vae_1**] uses VAEs, [**se_diff**] employs Diffusion Models, and [**selm**] utilizes language models. It has been demonstrated that generative models can achieve results comparable to, and even surpass, those of discriminative models [**selm**, **mask_sr**].

However, generative models, particularly language models, have not been fully explored in the Target Speaker Extraction (TSE) domain. This work aims to explore the use of language models for TSE.

1.5 Language Models

1.5.1 Attention

Language models are widely applied into the Natural Language Processing (NLP) fields. The components of the language models are Transformers [**vaswani2017attention**]. The core of transformers are the Attention mechanism [**vaswani2017attention**], which instructs the model to pay attention to the key points in a sentence. The scaled dot-product attention is the most commonly used attention mechanism, and it is formulated as follows:

- **Step 1: Compute the attention scores:**

  The attention scores are computed by taking the dot product between the query (Q) and key (K) vectors, followed by scaling by $\sqrt{d_k}$ to avoid large values:

$$\text{Attention Scores} = \frac{QK^T}{\sqrt{d_k}}$$

Where:

- $Q \in \mathbb{R}^{n_q \times d_k}$ is the matrix of queries.
- $K \in \mathbb{R}^{n_k \times d_k}$ is the matrix of keys.
- d_k is the dimension of the key vectors.

- **Step 2: Apply softmax:**

  The attention scores are passed through a softmax function to normalize them:

$$\text{Attention Weights} = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)$$

  The softmax function ensures that the attention weights sum to 1.

- **Step 3: Compute the output:**

  The attention weights are used to compute a weighted sum of the value vectors (V) to produce the final attention output:

$$\text{Attention Output} = \text{Attention Weights} \cdot V$$

  Where $V \in \mathbb{R}^{n_v \times d_v}$ is the matrix of values.

Thus, the complete attention formula is:

$$\text{Attention Output} = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) \cdot V$$

In multi-head attention, this process is repeated multiple times in parallel for different sets of queries, keys, and values, and the results are concatenated and projected through a linear transformation.

1.5.2 Decoder-only Language Models

Decoder-only language models are **autoregressive**, meaning they predict the probability of the next token based on previously generated tokens.

Assume we want to predict the probability of a sentence s of T tokens $w_1, w_2, w_3, \ldots, w_T$. The decoder-only model predicts the probability of the sentence using the chain rule:

$$P(w_1, w_2, \ldots, w_T) = P(w_1) \cdot P(w_2 \mid w_1) \cdot P(w_3 \mid w_1, w_2) \cdot \cdots \cdot P(w_T \mid w_1, w_2, \ldots w_{T-1}) \tag{1.5}$$

During inference, the decoder-only models will output one word at a time, and the output word will be taken as the input to the predict the next conditional probability. This is why it is called autoregressive. One example is shown in **Figure 1.2**, suppose we want to predict the word w_3 given the preceding words w_1 and w_2. The attention mechanism considers only past tokens (w_1, w_2) rather than future ones (w_3, w_4). Once w_3 is predicted, it is fed back into the model, which then predicts w_4 based on w_1, w_2, w_3.

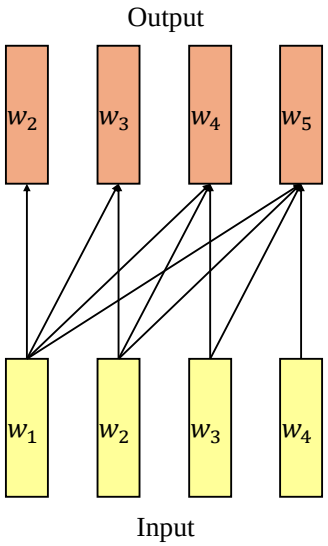

Figure 1.2: Decoder-only language models overview. Arrow stands for how attention is calculated, which is referred to as causal attention.

During training, instead of training the models with one word at a time, we can use the causal attention mechanism so that the attention will only focus on the words before without knowing the future, as shown in **Figure 1.2**. This causal attention greatly improves the training speed compared with other state methods like Recurrent Neural Networks (RNNs) and Long-short Term memories (LSTMs). However, the inference computation is still demanding as we need to do this calculation for each output word until the model generates a probability distribution regarding as the ending signal.

1.5.3 Discretization

NLP tasks are essentially classification tasks. Due to the nature of text inputs, all input data consists of words, or discrete tokens. They are called "discrete" because the number of possible outputs for each token is finite, based on the vocabulary size. The goal of most NLP language modeling tasks is to predict the probability distribution of the next token given the previous tokens.

Assume we have a vocabulary V, and discrete tokens $w_1, w_2, w_3, \dots, w_i \in V$, the objective is to predict the next token based on the previous tokens. In other words:

$$\hat{w}_{i+1} = \arg\max_{w \in V} P(w \mid w_1, w_2, \dots, w_i) \tag{1.6}$$

We can note that the output $\hat{w}_{i+1}$ is just a word belonging to V.

However, audio and speech signals are continuous features, and it is not possible to directly apply classification methods as there is no finite unit for audio signals. To apply NLP methods (classification methods) to audio, we must first **discretize** the audio signals by extracting small units and representing each unit with a token from a finite vocabulary, similar to text tokens. This allows us to use language models to process the audio. There are two primary approaches for discretization: **Neural Audio Codecs** and **Kmeans from Self-Supervised Learning Models (SSLs)**.

Neural Audio Codec

Audio codec serves as an efficient way to compress audio. With the development of neural networks, neural audio codec has outperformed traditional audio codec [**defossez2022highfi**].

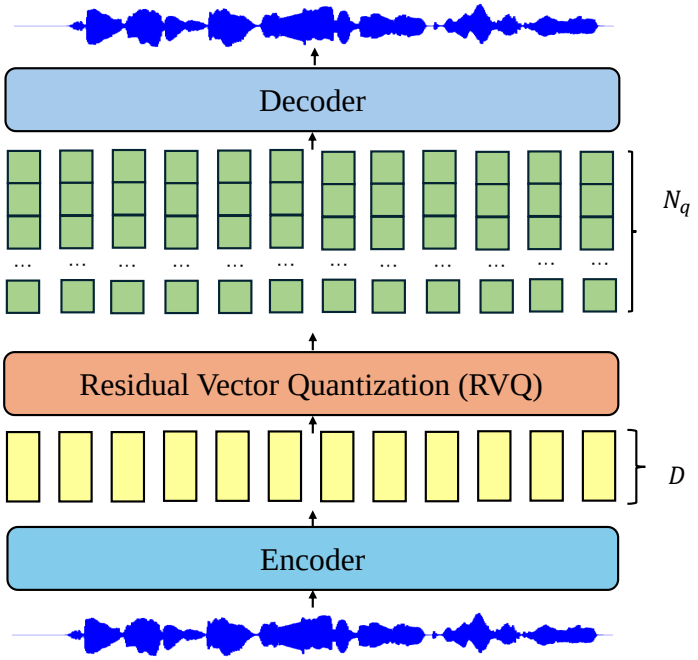

Figure 1.3: The overall framework Neural Audio Codecs. N_q stands for the number of quantization. Each green square after the RVQ layer represents a discrete token.

Most audio codecs like Descript Audio Codec [**dac**], FunCodec [**du2024funcodec**] follows Encoder - Residual Vector Quantization (RVQ) - Decoder paradigm as shown in **Figure 1.3**.

The encoder is composed of CNN layers, and it will output a high dimensional **dense embedding** sequences E. Assume we have a mono-channel speech $s(t) \in \mathbf{R}^{1\times T}$, then:

$$E = Encoder((s(t))) \in \mathbb{R}^{T\times D} \tag{1.7}$$

where D is the feature dimension.

This embedding will then pass through a **Residual Vector Quantization (RVQ)** layer to produce layers of discrete representations. The purpose of RVQ is to represent these embeddings using a finite set of representations, thus achieving compression. RVQ operates as follows:

First, we define a number of RVQ layers, denoted as N_q. Generally, the higher the number of layers, the higher the quality of the audio, though this also increases the computational cost. For each layer, we randomly define a codebook (C) where each codebook contains a finite number of randomly initialized embeddings (or codewords), corresponding to the vocabulary size $|V|$.

- **First Quantization Step:**
  - Each frame $x \in \mathbf{R}^D$ from the embedding E is approximated by a centroid (or codeword) q_1 from the first codebook C_1.
  - The residual is computed as:
    $$r_1 = x - q_1$$
    capturing what is left after the first quantization.
- **Iterative Refinement:**

- The residual r_1 is then quantized using a codebook from the second layer C_2, yielding another codeword q_2.
- A new residual is computed as:
$$r_2 = r_1 - q_2$$
- This process continues iteratively for multiple stages, producing progressively smaller residuals:
$$r_k = r_{k-1} - q_k, \quad \text{for } k = 2, \dots, N_q.$$

- **Final Representation:**
  - The final quantized representation of x is given by:
$$\hat{x} = q_1 + q_2 + \cdots + q_{N_q}$$
  - Instead of storing x directly, the indices of the selected codewords from each codebook are stored, allowing efficient reconstruction while saving storage.

We conduct RVQ for each frame in E to get

$$\hat{E} = [\hat{x_1}, \hat{x_2}, \dots, \hat{x_T}] \in \mathbf{R}^{T \times D} \tag{1.8}$$

which is a discretized version of E.

Finally, the Decoder will reconstruct the audio using $\hat{E}$. It is worth noting that the RVQ layer does nothing but provide a way to approximate the dense embedding E using finite codewords. Therefore, the Decoder can reconstruct the audio not only from $\hat{E}$, but also from E.

As we can see, the Neural Audio Codec serves as a direct way for discretization by using the discrete codebook words from the RVQ layers. We can then use the index of the codewords to simulate a token in an NLP task.

Kmeans from Self-Supervised Learning Model (SSL)

Self-Supervised Learning Models (SSLs) leverage self-supervised learning to train directly on data without requiring labels, generating high-dimensional, fine-grained representations.

As illustrated in **Figure 1.4**, the single-channel input audio $s(t) \in \mathbb{R}^{1 \times t}$ is first processed by a CNN encoder, producing a sequence of high-dimensional representation embeddings $E \in \mathbb{R}^{T \times D}$:

$$E = \text{CNN}(s(t)), \quad E = [x_1, x_2, \dots, x_T] \in \mathbb{R}^{T \times D} \tag{1.9}$$

where T is smaller than t due to the use of windowing and striding, which segments the audio into frame units.

We then randomly mask a subset of embeddings by replacing them with a special learnable masking embedding, denoted as MSK:

$$E_{\text{masked}} = [x_1, x_2, MSK, MSK, \dots, x_T] \tag{1.10}$$

The masked embeddings are then passed through Transformer encoders, where the attention mechanism learns to infer the masked positions using contextual information from the remaining embeddings.

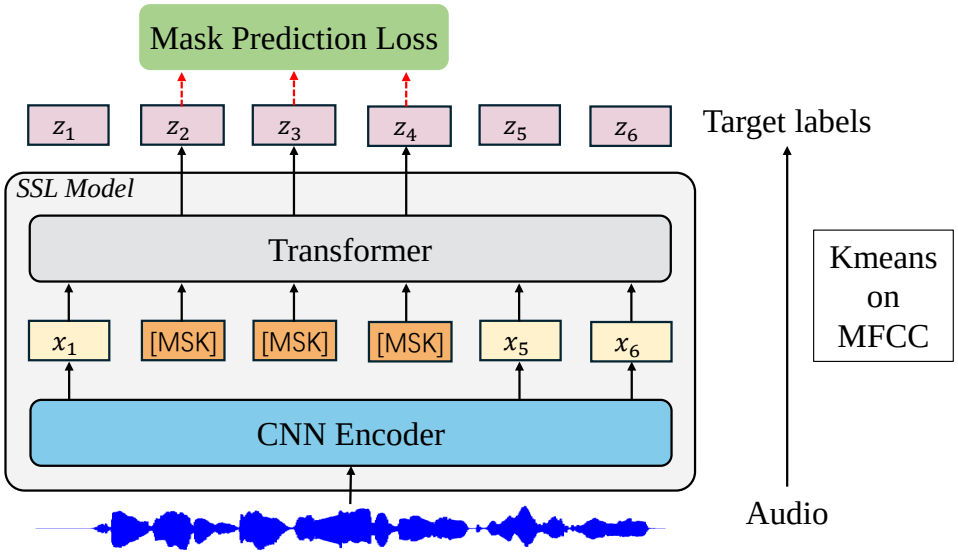

Figure 1.4: The overall framework of Self-Supervised Learning Model (SSL).

As the learning target for SSL models, we apply Kmeans clustering to the MFCC features extracted from the original audio. MFCC is a traditional speech feature extraction technique, and Kmeans provides discrete cluster assignments that serve as pseudo-labels for each MFCC class. Finally, cross-entropy loss is applied exclusively to the masked positions.

WavLM [**chen2022wavlm**] and HuBERT [**hsu2021hubert**] are two widely used SSL models for speech processing. Both models follow a similar framework, as illustrated in **Figure 1.4**. It is proved that the SSL models are good at catching semantic information [**chen2022wavlm**].

The key distinction between them lies in their training data and objectives. WavLM is partially trained on mixtures of speech, where the goal is to identify and model the primary speaker in the mixture. In contrast, HuBERT is trained exclusively on clean speech recordings, without the additional complexity of interfering speakers.

However, contrary to the Neural Audio Codec, the SSL representations are continuous instead of discrete. In fact, some discretization methods [**selm**, **dasb**, **tokensplit**] apply Kmeans on the intermediate representations from SSL models, and use the cluster center as discrete tokens. For example, training a Kmeans model with 1024 clusters will give you a finite 1024 number of discrete tokens.

1.6 Contribution

There are limited studies on discretization in Target Speech Extraction (TSE). SkiM-UniCATS [**gen_tse**] is one of the first to incorporate discrete tokens in TSE, utilizing SSL models such as HuBERT and vq-wav2vec [**vq_wav2vec**]. However, it overlooks the WavLM model, which has demonstrated superior performance in speech separation [**chen2022wavlm**]. Additionally, the focus of SkiM-UniCATS is primarily on single-layer outputs for discretization, and its evaluation is restricted to speech quality, ignoring key aspects like intelligibility and speaker similarity, which are essential for TSE. Moreover, it does not analyze how to conduct TSE using neural audio codecs. In this work, we explore two possible ways of using discrete tokens and language models for TSE:

- TSE using Kmeans from SSL model and encoder-only language models
- TSE using codec and decoder-only language models

where both works provide a thorough evaluation in terms of speech quality, speech intelligibility, and speaker similarity.

Chapter 2

MATERIAL AND METHODS

2.1 TSE using SSL model and encoder-only Language models

We propose TSELM: Target Speaker Extraction using Discrete Tokens and Language Models. TSELM is based on using discretization from Kmeans of multiple WavLM layers and encoder-only language models.

2.1.1 Overview

TSELM has three stages: encoding, modeling, and decoding, as shown in **Figure 2.1**. In the encoding stage, we use WavLM and Kmeans on both the reference speech from the target speaker and the mixture speech. For the reference speech, it is directly passed to the encoder. However, for the mixture speech, we first concatenate it with the reference speech on both sides before passing to the encoder. After the WavLM and Kmeans, we only keep the tokens corresponding to the mixture speech. In the modeling stage, we apply a so-called attention embedding mechanism to combine the embeddings from different layers. A cross-attention, as used in [**usef_tes**] is employed to combine the reference speech features and the mixture speech features. This approach is proved more efficient than the traditional TSE models where the model also combines auxiliary loss on Speaker Verification from the reference speech. An encoder-only language model then models the combined feature and then multiple linear classifier map the combined feature to the probability of the discrete tokens for each WavLM layer. In the decoding stage, we use a pretrained scalable HiFi-GAN in [**how_discrete**] to reconstruct the discrete tokens to speech. Different from SELM [**selm**], where a conformer detokenizer is trained to firstly reconstruct WavLM continuous embeddings from discrete ones before passing through HiFi-GAN for reconstruction, the scalable HiFi-GAN applies a dropout technique to directly reconstruct audio from multiple layers of discrete tokens. This eliminates the need to train a conformer detokenizer and a separate HiFi-GAN for each layer. During training, both the encoder and decoder are frozen. We have conducted extensive experiments, and they demonstrate that our method can achieve excellent speech quality and comparable speech intelligibility results.

2.1.2 Encoding

In our work, we use the pretrained SSL model WavLM Large[**chen2022wavlm**] to encode both the reference speech and the mixture speech into continuous representations due to the superior performance of WavLM in speech separation compared to other SSL models [**chen2022wavlm**]. Following [**dasb**] on Speech Separation, we use 6 hidden layers from WavLM to encode our speech.

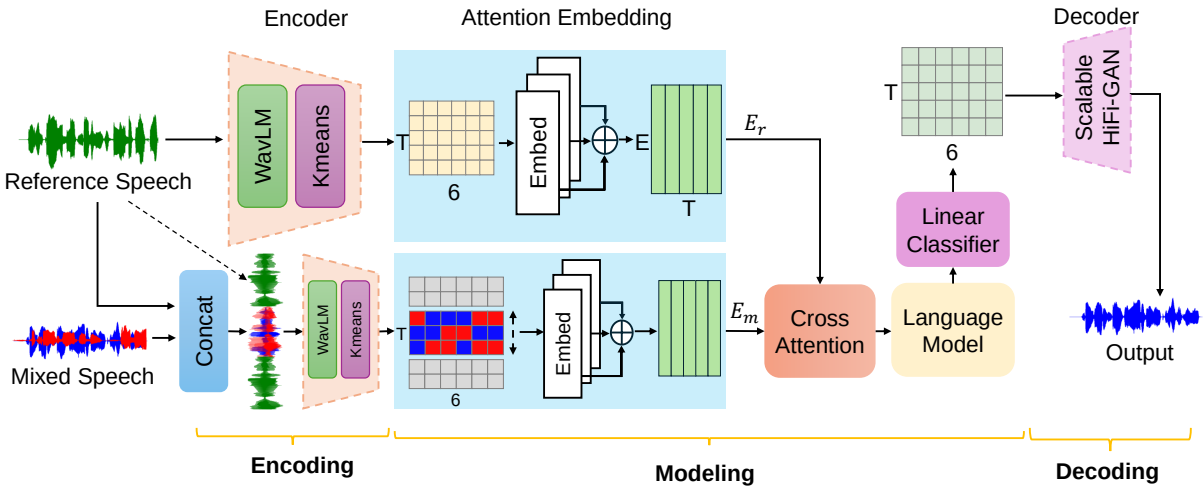

Figure 2.1: Overview of our target speaker extraction framework with discrete tokens from SSL and language models

Assume we have a speech signal $s_t \in \mathbb{R}^{1\times T'}$, then the output of multiple WavLM layers is a tensor $E = [E_1, E_2, \dots, E_n] \in \mathbb{R}^{n\times T\times D}$, and $E_i \in \mathbb{R}^{T\times D}$, where n is the number of the output layers, which is 6 in our case, and E_i stands for the output embedding for the i-th layer from WavLM. T is the time dimension, and it is the number of frames produced by each layer of WavLM, and D is our embedding dimension. After that, we apply n K-means models $[KM_1, KM_2, \dots KM_n]$ with each model using the same number of clusters K to each of the output layers from WavLM, respectively, and we get a tensor d of shape $n \times T$:

$$d = [d_1, d_2, ..., d_n] \in \mathbb{R}^{n\times T} \tag{2.1}$$

$$d_i = KM_i(E_i) \text{ for } i \in \{1, 2, \dots, n\} \tag{2.2}$$

where each value $d_{i,j}$ is a discrete integer value $\in (0, K-1)$.

In our experiments, we set $K = 1000$. For both the reference and the mixed speech, the same Kmeans model and the same layer combination from WavLM Large are employed. The WavLM is already pretrained on a large corpus of data, therefore we keep it frozen during training. Regarding the Kmeans, it is first trained on the training data to learn the clusters. It is then used to predict the center of the clusters during our actual training in the modeling stage of TSELM.

One thing to note is that the encoding strategy for mixture speech is crucial to the overall performance for the model. Given a reference speech $s_r \in \mathbb{R}^{T^r}$ and the mixture speech $s_m \in \mathbb{R}^{T'}$, we follow the described procedure above to encode the reference speech into a tensor d_r of shape $n \times T_r$. However, regarding the mixture speech, rather than directly feeding this to WavLM model, we first concatenate it with the reference speech on both sides of the mixture, creating a signal:

$$s' = [s_r, s_m, s_r] \in R(T^r + T' + T^r). \tag{2.3}$$

This concatenated signal s' is then input to the WavLM and Kmeans encoder, resulting in an output discrete tensor d' of shape $n \times (T^r + T + T^r)$, where T stands for the length corresponding to the mixture speech embedding. This tensor d' contains the discrete tokens for two segments of the reference speech and the mixture speech. Finally, we extract the portion of tensor from

d' which corresponds to the discrete tokens of mixture speech, producing the output tensor d of shape $n \times T$.

This concatenation strategy is inspired by WavLM [**chen2022wavlm**]. WavLM is trained by overlapping the target clean speech with an interfering speech covering less than 50% of the clean speech, and the target label is the K-means on MFCC of the target clean speech only. This step will allow WavLM to identify the target speaker by the relative length compared with the interfering speech, through transformer encoders, making WavLM produce target speaker dependent embeddings. Our experiments demonstrate that this concatenation strategy greatly enhances the model's performance by actually extracting more information on our target speaker's information.

2.1.3 Attention Embedding

After obtaining the discrete tensor d with shape $n \times T$, we ues N learnable embedding tables each with K entries to embed the N layers respectively, each resulting in a tensor E_i of shape $T \times D$:

$$\hat{E}_i = Embedding(d_i) \in \mathbf{R}^{T \times D} \quad \text{for} \quad i \in 1, 2, \dots, n \tag{2.4}$$

After the embedding layer, we follow [**how_discrete**] to use attention mechanism to sum all the embedding from all the layers to a single embedding E with shape $T \times D$:

$$a_{k,t} = \frac{\exp(\hat{E}_{k,t})}{\sum_{j=1}^{n} \exp(\hat{E}_{j,t})}, \quad E_t = \sum_{i=1}^{n} a_{i,t} \cdot \hat{E}_{i,t} \in \mathbf{R}^D \tag{2.5}$$

$$E = [E_1, E_2, \dots, E_T] \in \mathbf{R}^{T \times D} \tag{2.6}$$

where a is the attention weight. This summation keeps the information of each layer while reducing the dimension of layers and system complexity from N to 1.

Unlike [**mask_sr**], which uses a direct average among all the layers. This attention-based summation makes the model to focus on different layers based on different inputs, and it is proved to be more efficient than average summation [**how_discrete**].

We apply attention embedding to both the reference discrete tokens and the mixture discrete tokens, resulting in reference embedding E_r and mixture embedding E_m, respectively.

2.1.4 Cross Attention

After obtaining the mixture embedding E_m and the reference embedding E_r, we apply cross-attention module and Feature-wise Linear Modulation (FiLM) similar in [**usef_tes**] to inject the reference embeddings into the mixture. It has been demonstrated that cross-attention and FiLM can inject the reference speaker information and outperforms the other TSE models that do not use this mechanism [**usef_tes**]. The details are shown in **Figure 2.2**.

We use the E_m as Query (Q) and E_r as the Key (K) and Value (V) so that:

$$Q = Linear(E_m), K = Linear(E_r), V = Linear(E_r) \tag{2.7}$$

After obtaining Q, K, V, we apply the Attention layer as described in **Section 1.5.1**. After that, a Multi-Layer Perception (MLP) is applied. This module is repeated L times, and the output embedding is denoted as E_{spk}.

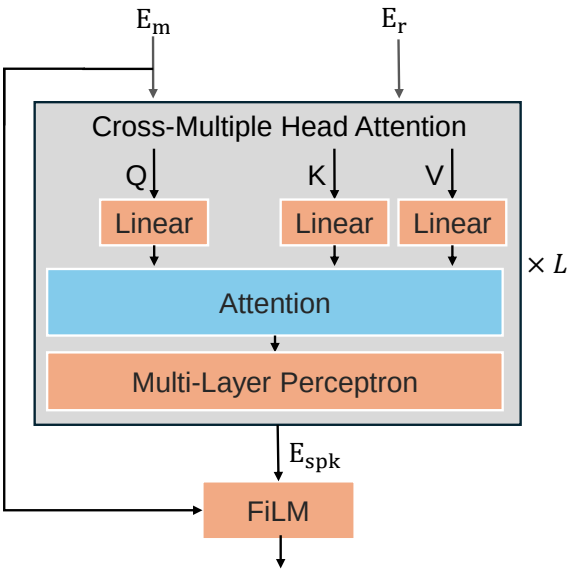

Figure 2.2: Details of the Cross-Attention mechanism and Feature-wise Linear Modulation (FiLM) in modeling.

The FiLM layer works as followed:

$$E_f = FiLM(E_m, E_{spk}) \tag{2.8}$$

$$FiLM(E_m, E_{spk}) = \lambda E_{spk} \cdot E_m + \beta \cdot E_{spk} \tag{2.9}$$

where λ and β are learnable parameters, denoting the scaling and shifting vectors, respectively.

2.1.5 Language Modeling (LM)

We use transformer encoder language models containing multiple self-attention modules to model the embedding E_f. Due to the encoder-only style, the language models are able to learn from all the positions, producing:

$$E_l = LM(E_f) \in \mathbb{R}^{T \times D} \tag{2.10}$$

Finally, n linear classifiers $[Linear_1, Linear_2, \dots, Linear_n]$ each with dimension K is used to produce the logit scores p for the tokens.

$$\hat{p}_i = Linear_i(E_f) \in \mathbf{R}^{T \times K} \tag{2.11}$$

$$\hat{p} = [\hat{p}_1, \hat{p}_2, \dots, \hat{p}_n] \in \mathbf{R}^{n \times T \times K} \tag{2.12}$$

where $\hat{p}_{i,j} \in \mathbb{R}^K$ refers to the probability distribution at layer i and time j.

Regarding the ground truth of the model, we apply the same WavLM and Kmeans structure on the clean speech, producing n layers of discrete tokens $y \in \mathbb{R}^{n \times T \times K}$. $y_{i,j}$ is a one-hot vector where the index of the ground label is 1, and the rest $K - 1$ positions are 0.

Similar as pixel-wise classification, we use Cross-Entropy Loss on each of the position $(i, j), i \in 1, 2, \ldots, n, and j \in 1, 2, \ldots, T$ and average them to get our total loss:

$$L = -\frac{1}{n \times T} \sum_{i=1}^{n} \sum_{j=1}^{T} \sum_{c=1}^{c=K} y_{i,j,c} \log \hat{p}_{i,j,c} \tag{2.13}$$

2.1.6 Model Details

We select the output from hidden layers 1, 3, 7, 12, 18, 23 from WavLM Large and Kmeans model with $K = 1000$. Since WavLM Large uses embedding dimension 1024, we select our D to be 1024 as well. Regarding the cross-attention module, it consists of 4 transformer encoders, each with 16 attention layers and an MLP with a hidden dimension of 1024. Layer normalization is applied after the cross-attention module.

We have done three different scales regarding the LM part of the TSELM system. The small version TSELM-S uses embedding dimension $d = 256$, absolute sinusoidal positional embedding, and conformer encoders as the backbone of the LM. The conformer encoder consists of 6 layers with a kernel size of 31, each with 4 attention heads, and an MLP with a hidden dimension of 2048. We apply GELU as the activation function, and dropout of 0.1. The medium version TSELM-M uses $d = 512$ with 8 layers and 8 heads, and the large version TSELM-L uses $d = 768$ with 12 layers and 16 heads. For all the experiments, we use AdamW as the optimizer with β from 0.9 to 0.98, and ϵ with 1×10^{-8}, and weight decay 0.01. For TSELM-S, the learning rate is 5×10^{-4}. For TSELM-M and TSELM-L, the learning rate is 5×10^{-5}.

All the models are trained using 8 GPUs with 32GB of RAM, each with batch size 16 for a total number of 40,000 steps.

2.1.7 Training Data

We use the publicly available WavLM-Large pretrained on roughly 94,000 hours of audio data [**chen2022wavlm**]. Regarding Kmeans tokenizer and scalable HiFi-GAN, we use the ones trained from [**dasb**]. The Kmeans tokenizer is trained on 960 hours of `train_clean_[100,360,500]` of LibriSpeech [**librispeech**]. LibriSpeech is a dataset containing single-channel 16kHz clean speeches from different speakers. The scalable HiFi-GAN is trained on `train_clean_100` of single-channel 16kHz LibriTTS [**libritts**]. LibriTTS is a high-quality clean speech from different speakers that are mostly used for Text-To-Speech tasks.

Regarding our experiments on training the modeling stage, the training data are generated on they fly from `train_clean_[100,360]` of LibriSpeech using dynamic mixing where we randomly select a target speaker and his/her reference speech, and then we mix it with another interference speaker. The resulting mixture audio has relative SNR between 0 to 5 dB. The mixture audio and the reference audio is clipped to 3 and 4 seconds, respectively.

2.1.8 Evaluation

All the models are evaluated using Lbri2Mix [**librimix**] and WSJ0-2mix [**wsj0**]. We follow the recipes[1] to form the reference speeches for WSJ0-2mix. We use the clean testset of Libri2Mix[2], and the reference speeches are randomly selected.

Metrics such as PESQ, SI-SNR, and STOI have been shown to inadequately reflect the speech quality of vocoder outputs, as vocoders do not strictly prioritize frame alignment [**tokensplit**, **selm**]. This conclusion is further supported by the SI-SNR formula in **Section 1.3**, which

[1] https://github.com/xuchenglin28/speaker_extraction
[2] https://github.com/JorisCos/LibriMix

explicitly minimizes the distance between the estimated and ground-truth audio. However, because HiFi-GAN incorporates an additional loss term via a discriminator during training, its output is not strictly aligned at the frame level.

Therefore, we use DNSMOS [**dnsmos**] to measure the speech quality and the diffreential Word Error Rate (dWER) [**dwer**] using the base model of Whisper [**whisper**] to measure the speech intelligibility. For speaker similarity, we use the `ResNet221_LM` from the public WeSpeaker [**wespeaker**] to calculate the embedding cosine speaker similarity.

- **DNSMOS** [**dnsmos**] is a neural network-based model for estimating objective speech quality. Its predictions are highly influenced by background noise. Additionally, it includes SIG, BAK, and OVRL scores, which represent the signal quality, background noise quality, and overall quality, respectively.
- **Whisper** [**whisper**] is a speech recognition model used to transcribe speech into text. To compute dWER, we use the base Whisper model to extract text from both the estimated and ground-truth speech, then compare the transcriptions.
- **WeSpeaker** [**wespeaker**] is a neural-network model that extracts unique speaker embeddings. Speaker similarity is measured by computing the cosine similarity from speaker embeddings between the estimated speech and the target speech.

2.2 TSE using codec and decoder-only language models

TSE using decoder-only language models remains an underexplored area. Some existing works, such as SpeechX [**wang2024speechx**], employ multi-task decoder-only models capable of handling various tasks beyond TSE, including Text-to-Speech, Speech Enhancement, and Clean Speech Editing. This multi-task capability is achieved through the use of a special task embedding. However, a key limitation of SpeechX is that its TSE evaluation is conducted on relatively simple datasets where the primary speaker is dominant in the mixture. While such models demonstrate versatility across multiple tasks, their ability to handle more challenging TSE scenarios remains uncertain.

In this work, we investigate the potential use of decoder-only language models for single TSE task. We propose Do-TSE, a Decoder-only TSE model based on FunCodec [**du2024funcodec**] and LauraGPT decoder-only model backbone [**du2023lauragpt**].

2.2.1 Overview

The overall framework of Do-TSE is illustrated in **Figure 2.3**. Following [**du2023lauragpt**], we extract the log-compressed Mel-spectrograms of both the reference speech and the mixture speech. These spectrograms are then processed by a conformer-based encoder, which encodes them into high-dimensional embeddings, denoted as E_r and E_m, respectively.

Next, E_r and E_m are fed into our decoder-only transformer. The decoder-only transformer will autoregressively generate the output tokens of the **first** layer of RVQ one by one following the probability chain as stated in **Equation 1.5**. Once the decoder-only transformer predicts the first layer of discrete tokens, an encoder-only transformer is applied. This encoder-only Transformer takes E_r, E_m, and V as inputs and predicts the dense embeddings of FunCodec corresponding to the clean speech. The target for this encoder-only transformer is the dense FunCodec embeddings of the clean speech. Finally, the FunCodec decoder reconstructs the output speech from these embeddings.

It is important to note that FunCodec Decoder remains frozen throughout both training and inference.

2.2.2 Encoding

For both the reference and the mixture speech, we first apply a log-compressed Mel-spectrogram, a common practice in speech processing for feature extraction. These features are then fed into the same conformer encoder, which encodes them into high-dimensional **continuous** representations $E_r \in \mathbf{R}^{T_r \times D}$ and $E_m \in \mathbb{R}^{T \times D}$, where the D stands for the embedding dimension. The primary role of this encoder is to transform the speech signals into embeddings that are well-suited for the decoder-only transformer.

In contrast, some existing works, such as [**wang2024speechx**], opt to use a codec encoder to extract discrete features instead of a trainable conformer encoder to extract continuous features. However, this approach presents several challenges. First, most neural audio codecs are not trained on mixture speech, which can lead to information loss, as observed in FunCodec [**du2024funcodec**]. Second, even if the output features exist within the codec embedding space, they may not be optimal for the decoder-only transformer. Finally, prior work [**du2023lauragpt**] has demonstrated that a trainable encoder for continuous features generally yields superior performance compared to discrete features. Therefore, we adopt this structured approach in our model.

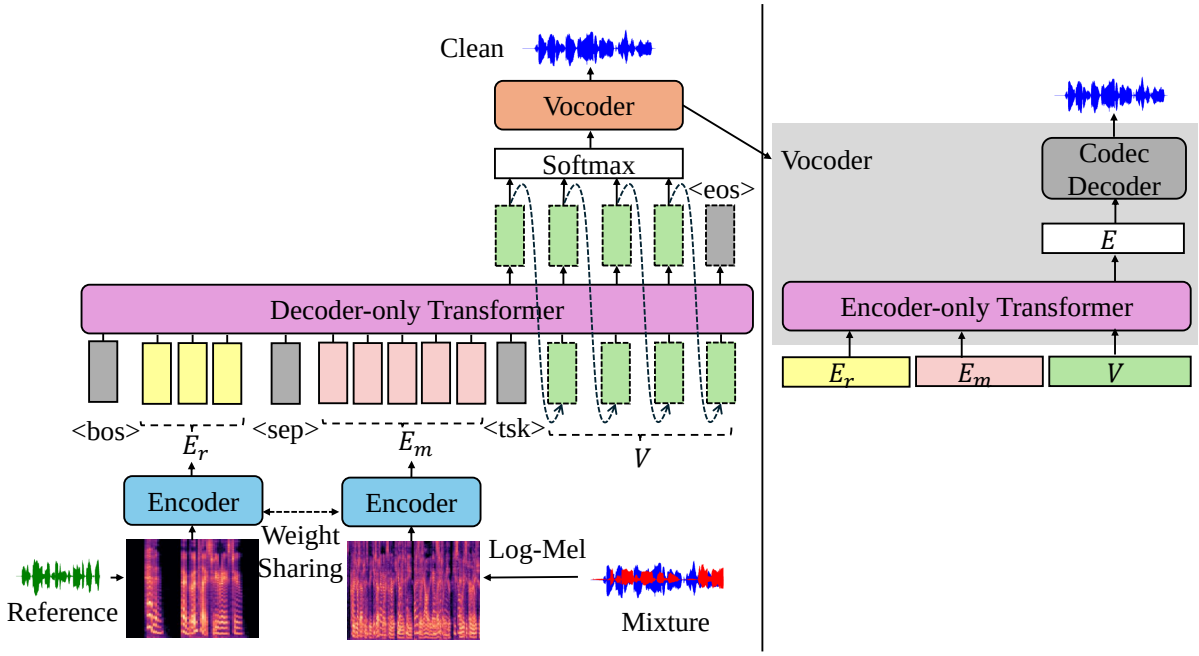

Figure 2.3: The overall framework of Decoder-only TSE (Do-TSE).

2.2.3 Decoder-only Transformer

Decoder-only transformer is a sequence-to-sequence autoregressive model, therefore, we use three special tokens $< bos >$, $< sep >$, $< tsk >$ to format our input. The $<$ bos $>$ token, commonly used in NLP, signals the beginning of a sequence. The $<$ sep $>$ token marks the boundary between the reference and mixture embeddings, while the $<$ tsk $>$ token separates the input from the output within the model. Note that since we are only performing TSE tasks, there is only one $<$ tsk $>$ embedding.

For target output generation, we utilize discrete tokens from FunCodec [**du2024funcodec**]. Specifically, we extract only the first layer output of the (RVQ) output from FunCodec as the decoder-only transformer's target. During training, the input is formatted as:

$$[< \text{bos} >, E_r, < \text{sep} >, E_m, < \text{tsk} >, E]$$

where E_{l1} represents embedding from the first layer of the FunCodec discrete tokens V corresponding to the target clean speech. We will use the causal attention module to make sure that the model can only consider the past information rather than the future. The model will output a sequence of probability distribution of the same length as the input. We only consider the probability output after the $<$ tsk $>$, which corresponds to the probability of the our target discrete tokens V.

During inference, the input is structured as:

$$[< \text{bos} >, E_r, < \text{sep} >, E_m, < \text{tsk} >]$$

It will then generate the output probability autoregressively following the chain rule as mentioned in **Section 1.5.2**.

Theoretically, the decoder-only transformer is trained to minimize the Cross-Entropy loss:

$$L_{LM} = -\frac{1}{T_v}\sum_{j=1}^{T_v}\log p_\theta(V^j \mid E_r, E_m, V^{1:j-1}) \tag{2.14}$$

where E_r and E_m are the reference embedding and the mixture embedding, respectively. T_v and V represents the sequence of target tokens with a length T_v. Note that only the outputs corresponding to V will be considered into the total loss. This is shown in **Figure 2.3** where only the part after $< tsk >$ is taken into consideration as the real output.

Only the first-layer discrete output is predicted for the following reasons. First, the first layer of discrete tokens captures most of the speech information due to the nature of RVQ. Second, predicting higher-layer tokens would significantly increase the computational load on the decoder-only transformer.

2.2.4 Vocoder

Vocoder is responsible for generating the raw audio from the first layer of the predicted first layer of discrete tokens V. It is composed of an encoder-only transformer and a pretrained codec decoder.

We first embed V using corresponding codewords of the first codebook layer from the pretrained codec:

$$E_{l1} = Emb(V) \in \mathbb{R}^{T\times D} \tag{2.15}$$

To reconstruct the summation of the dense embeddings of the target speaker based on the first layer embedding E_{l1}, We follow [**du2023lauragpt**] to an encoder-only transformer on the input sequences $[E_r, E_m, E_{l1}]$. After getting the result tensor, we take only the portion of the output corresponding to E_{l1}, called $\hat{E} \in \mathbb{R}^{T\times D}$. The Encoder-only Transformer is trained to minimize the summation of L1 and L2 loss:

$$L_{pre} = \sum_{t,i}^{T,D}|E_{t,i} - \hat{E}_{t,i}| + |E_{t,i} - \hat{E}_{t,i}|_2 \tag{2.16}$$

where E is the dense embedding of the target clean speech.

After obtaining these dense embeddings, the pretrained codec decoder will reconstruct the speech.

Unlike some other approaches [**wang2024speechx**] which predict the embedding layer by layer, this approach from [**du2023lauragpt**] which directly predicts the summation of all the embeddings save a lot of computation while having superior results.

2.2.5 Model Details

We use a hop size of 256 and a window size of 512 for both the reference speech and the mixture speech during encoding. The conformer encoder consists of 6 layers, each with 8 attention heads and a feature dimension of 512. For the decoder-only transformer, we adopt LauraGPT [**du2023lauragpt**] as the backbone, which has 10 layers, 8 attention heads, and a feature dimension of 512. The encoder-only transformer used in the vocoder is composed of 6 layers, 8 attention heads, and a 512-dimensional feature space. Our Do-TSE model are trained from scratch. The total number of parameters for Do-TSE is 77M, with 36M parameters corresponding to the decoder-only transformer. We use the Adam optimizer with a learning rate of 1×10^{-3}. A warmup scheduler is applied with 10,000 warmup steps. Do-TSE is trained on 16 GPUs, each with 32GB of RAM, for 115K steps.

2.2.6 Training and Evaluation

We utilize Libri2Mix [**librimix**] as our training data without any dynamic mixing for 50 epochs (115K steps). As for evaluation, we utilize the same test set as TSELM, which is Libri2Mix testset and WSJ0-2mix. Same as TSELM, we utilize DNSMOS, dWER, and Speaker Similarity as our evaluation metrics.

2.3 Baseline models

All baseline models are presented in **Table 3.1** and **Table 3.3**. Both TSELM and Do-TSE are compared with a discriminative TSE model Spex+ [**spex_plus**] trained on Libri2Mix. They are also compared with SkiM-UniCATS(vq-wav2vec) [**gen_tse**] on WSJ0-2mix, which uses discrete tokens from vq-wav2vec

To fully understand the TSELM model which uses the discrete representations, we conduct three other experiments: Continuous-WavLM-L6, TSELM-L-Hybrid, and TSELM-S-NoCat using the same training and evaluation data. For Continuous-WavLM-L6, we directly pass the embeddings from the 6th hidden layer output of WavLM-Large to the cross-attention module without discretization and attention embedding. The concatenation strategy is still applied to the mixture speech. Mean Square Error loss is applied between the output embeddings and the clean embeddings. HiFi-GAN in [**knn_vc**] is applied to reconstruct the clean speech from WavLM embeddings. For TSELM-L-Hybrid, inspired by [**mask_sr**], we only discretize the reference speech while keeping the continuous WavLM embeddings for the mixed speech. For TSELM-S-NoCat, we did not apply the concatenation strategy for the mixture speech in the encoding step.

Chapter 3

RESULTS

3.1 TSELM and Do-TSE with other baseline models

In **Table 3.1** and **Table 3.2**, we have compared our TSELM and Do-TSE on Libri2Mix and WSJ0-2mix, respectively. In the "Category" column, "G" refers to generative models, while "D" refers to discriminative models. The "Type" column categorizes methods as "D" (discrete), "H" (hybrid), or "C" (continuous). DNSMOS is computed over the output since this metric is reference-free. dWER is calculated with respect to the clean speech. For continuous methods, speaker similarity is directly computed against the clean speech. However, since Kmeans discretization inherently results in some loss of speaker information—Target-Kmeans shows a speaker similarity score of 0.654 on Libri2Mix -we follow [14] to assess the speaker similarity of the output from the discrete methods against the target audio produced by discretizing the clean speech. This comparison is denoted as "_d". Note that for Do-TSE which uses codec for discretization, we directly compare the speaker similarity with the original clean speech. "Mixture" refers to the unprocessed mixture speech. Model Size refers to the learnable parameters in the models. For TSELM, it is the language model, and for Do-TSE, it refers to the conformer encoder, decoder-only transformer, and encoder-transformer.

Table 3.1: Results on Libri2Mix clean.

System	Category	Type	Model size	Libri2Mix Clean				
				DNSMOS ↑			dWER ↓	Spk Sim ↑
				SIG	BAK	OVL		
Mixture	-	-	-	3.38	3.10	2.65	79.2	0.762
Target-Kmeans	G	-	-	3.47	4.03	3.19	11.8	0.654
Spex+[**spex_plus**]	D	-	11M	3.38	3.77	3.00	19.0	0.922
Continuous-WavLM-L6	G	C	141M	3.57	4.06	3.28	14.6	0.870
TSELM-L-Hybrid	G	H	138M	3.49	4.05	3.22	20.0	0.917_d
TSELM-S-NoCat	G	D	24M	3.48	4.02	3.19	71.5	0.854_d
TSELM-S	G	D	24M	3.50	4.06	3.23	28.1	0.883_d
TSELM-M	G	D	59M	3.49	4.04	3.21	29.0	0.892_d
TSELM-L	G	D	138M	3.49	4.04	3.21	27.5	0.895_d
Do-TSE	G	D	77M	3.65	4.10	3.36	24.1	0.847

Table 3.2: Results on WSJ0-2mix.

System	Category	Type	WSJ0-2Mix				
			DNSMOS ↑			dWER ↓	Spk Sim ↑
			SIG	BAK	OVL		
Mixture	-	-	3.42	3.28	2.81	63.6	0.824
Target-Kmeans	G	-	3.56	4.09	3.30	10.1	0.657
Spex+[**spex_plus**]	D	-	3.49	4.00	3.21	15.0	0.943
SkiM-UniCATS[**gen_tse**]	G	D	3.62	4.10	3.37	-	-
Continuous-WavLM-L6	G	C	3.61	4.08	3.35	8.0	0.892
TSELM-L-Hybrid	G	H	3.57	4.10	3.31	12.6	0.915_d
TSELM-S-NoCat	G	D	3.55	4.08	3.28	64.5	0.888_d
TSELM-S	G	D	3.57	4.10	3.32	19.4	0.915_d
TSELM-M	G	D	3.57	4.10	3.32	18.8	0.921_d
TSELM-L	G	D	3.57	4.10	3.31	17.8	0.924_d
Do-TSE	G	D	3.63	4.12	3.38	17.1	0.870

Table 3.3: Performance of different SSL models and layer selections on Libri2Mix clean.

SSL-Model	Type	DNSMOS ↑			dWER ↓	Spk Sim ↑
		SIG	BAK	OVL		
HuBERT [**hsu2021hubert**]	Discrete	3.57	4.09	3.31	82.2	0.854_d
	Hybrid	3.57	4.10	3.32	36.1	0.900_d
WavLM-L6	Discrete	2.08	2.07	1.64	122.14	0.589_d
	Hybrid	3.54	3.93	3.18	29.3	0.838_d
WavLM	Discrete	3.49	4.04	3.21	27.5	0.895_d
	Hybrid	3.49	4.05	3.22	20.0	0.917_d

3.2 Ablation study on TSELM

To comprehensively evaluate the impact of different SSL models, varying numbers of discretization layers, and the effectiveness of the hybrid mode—where the mixture remains continuous while the reference is discretized—we conducted ablation studies, as shown in **Table 3.3**.

In the "Hybrid" mode, the mixture is not discretized using Kmeans; instead, its WavLM embeddings are directly fed into the language model, while the reference speech undergoes discretization. In contrast, the "Discrete" mode applies Kmeans discretization to both the mixture and the reference.

"WavLM" refers to our TSELM-L model utilizing six hidden layers. "WavLM-L6" specifically employs only the sixth hidden layer output of the WavLM Large model. "HuBERT" replaces WavLM with the HuBERT [**hsu2021hubert**] SSL model.

Chapter 4

DISCUSSION

4.1 TSELM

Firstly, as we can see that the speaker similarity of Target-Kmeans is relatively low. One of the possible reason is because of the discretization process (Kmeans), which loses a lot of information, especially speaker information because we use the center clusters to represent the all the nearby embeddings. This approach tends to focus more on the semantic information rather than acoustic information which is necessary for speaker similarity. Another reason might be because of the scale of the dataset. The LibriSpeech and the LibriTTS is relatively small, and TSELM might perform better with more diverse dataset. One of the future works should be exploring other techniques for discretization using SSL models to preserve better speaker identity.

We observe that TSELM-L outperforms Spex+ in terms of DNSMOS scores, indicating better speech quality, but performs slightly worse in dWER, suggesting lower speech intelligibility. One potential reason for this could be the discretization process applied to the mixed speech. Our Kmeans algorithm is trained on clean speech rather than mixed speech, which is advantageous for speech enhancement as it likely aids in denoising because that the noisy embedding is not that different from the clean embedding, as proved in [**selm**]. However, when applied to mixture speech with multiple speakers, this discretization might lead to the model focusing on the wrong speaker, as it is most likely to retain only the dominant speaker information, causing a reduction in intelligibility. This hypothesis is supported by the results of TSELM-L-Hybrid, where continuous embeddings from the mixture speech are used without discretization, achieving dWER scores comparable to Spex+ on Libri2Mix and even better than Spex+ on WSJ0-2mix. Another contributing factor could be the limitations of our current language model. The encoder-only language model achieves around 55% accuracy for token prediction on Libri2Mix testset, and we believe using more advanced models, such as masking-based language models, could lead to improved performance in future iterations.

We observe a significant increase in dWER when the mixture audio is not concatenated with the reference in the encoding step, as seen in the TSELM-S-NoCat results in **Table 3.1**. For WavLM to effectively perform TSE, the input audio must follow specific conditions: the mixture should be less than 50% of the total length, and the first utterance should be the target speaker. Under these conditions, WavLM outputs a slightly denoised embedding that emphasizes the target speaker. When the entire input is a mixture without concatenation, however, we found that WavLM sometimes extracts the wrong speaker at some frames, where the wrong speaker has a louder voice. Our current concatenation strategy, inspired by SELM's [**selm**] success in speech denoising, reframes the TSE task as a more challenging speech enhancement problem,

utilizing WavLM's denoising capabilities by concatenation. However, this approach remains suboptimal because firstly if WavLM can focus on target speaker already, why not fine-tune it in TSE task rather than training another language model? Secondly, the reference speech is used twice in this approach, and thus adds extra computation. We believe that our future work should prioritize developing a speaker-aware tokenization method for SSL models.

In **Table** 3.3, we compare the performance of HuBERT and WavLM as SSL models and examine the effects of using either one or multiple layers for discretization. Our findings indicate that using HuBERT as the SSL model results in slightly better DNSMOS scores but much worse dWER compared to our WavLM baseline. The improved DNSMOS scores likely stem from the vocoder performance, yet they do not adequately reflect speech intelligibility, which is crucial for speech separation tasks. This is the reason why we think SkiM-UniCATS might not accurately reflect the speech intelligibility and speaker similarity as it only considers the DNSMOS score.

The poorer dWER scores observed with this HuBERT model may be attributed to its pretraining on clean speech, which might not equip it to capture the complexity and richness of mixed speech like WavLM. Moreover, our results from WavLM-L6 suggest that when performing speech separation, discretizing across multiple layers provides better results than relying on a single layer. This might be because that using multiple layer outputs can better tolerate errors compared to using just one layer output.

4.2 Do-TSE

Do-TSE outperforms the TSELM models across all evaluation metrics while using only half the parameters of TSELM-L. Additionally, it achieves near-native speaker similarity, rivaling TSELM. This improvement stems from the use of neural codec models, which are specifically designed to reconstruct speech from discrete tokens while preserving speaker information. As a result, neural codecs may better preserve acoustic features compared to Kmeans clustering applied to SSL models. However, Do-TSE's speech intelligibility remains lower than that of TSELM-L-Hybrid, which benefits from WavLM embeddings extracted from the reference speech. A promising future direction could be integrating neural audio codecs with SSL models.

One limitation of our work is that we employ relatively small decoder-only models due to time and computational constraints, as well as a limited amount of training data. To fully leverage the power of language models, significantly larger datasets beyond 500 hours in this study would be necessary. Furthermore, we do not conduct ablation studies to compare the effectiveness of Mel-spectrogram representations versus audio codecs for speech encoding. Moreover, we only use FunCodec as our neural audio codec rather than exploring other neural audio codecs. Lastly, we do not explore the potential benefits of using pretrained decoder-only transformers and ASR-conformers, which could further enhance performance.

Another promising direction for future work is integrating multimodal information into our models, as the current approach relies solely on reference audio. Beyond speech, reference information can be derived from video and text data.

For instance, instead of using only the target speaker's voice as a reference, we could incorporate lip movement features extracted from video input. Lip-reading models have demonstrated strong potential in speaker identification and speech enhancement, particularly in noisy environments where audio quality is degraded. By leveraging visual information, our model might also achieve better results. Additionally, textual context can serve as valuable reference information. We could integrate transcriptions of the speech or use textual prompts to guide the model toward extracting the desired speaker. For example, instead of relying solely on an audio reference, a user might provide a text prompt such as, "I want to separate the man's

voice in the mixture." This method can be more convenient for the user as they do not need to provide auxiliary audio or video information to assist TSE. Combining multimodal information—audio, video, and text—could might enhance our model's ability to extract target speakers with higher accuracy and robustness under many different scenarios, making it more adaptable to real-world applications, which can be a future direction.

4.3 Discrete vs. Continuous

Finally, we observe a performance gap between discrete and continuous methods, as demonstrated by the superior results of Continuous-WavLM-L6. Among all experiments, Continuous-WavLM-L6 achieves the highest performance in terms of DNSMOS and dWER while utilizing only the 6th-layer output of WavLM Large. This performance gap may stem from the inherent information loss introduced during the discretization process, whether through Kmeans or neural audio codec.

One key advantage of discretization is its ability to transform complex regression problems into classification problems, potentially simplifying the task. However, a possible reason for the performance gap is the limited amount of training data used in our experiments. Classification-based approaches typically require a comprehensive understanding of the full data distribution, which demands significantly larger datasets for effective generalization. Scaling up the dataset might help bridge this gap and improve the effectiveness of discrete representations. Our future research will also work on scaling data.

Chapter 5

CONCLUSIONS

In this work, we explore the use of discrete tokens from both self-supervised learning (SSL) models and neural audio codecs for Target Speaker Extraction (TSE). For SSL models, we employ Kmeans clustering to discretize multiple layers of WavLM, while for neural audio codecs, we utilize FunCodec for discretization alongside a LauraGPT decoder-only transformer backbone.

Our experiments demonstrate that these generative approaches achieve excellent performance in terms of speech quality and competitive results in speech intelligibility. However, we observe a notable performance gap between discrete and continuous methods, particularly in speech intelligibility and speaker similarity. In future work, we aim to bridge this gap by further refining discretization techniques and exploring hybrid approaches that better preserve acoustic and semantic features.

REFERENCES

Chapter 6

SIGNATURE WORK NARRATIVE

6.0.1 three thematic courses

COMPSCI 308: Advanced Software Design and Implementation (Duke)

I took this course during my time at Duke, and it provided me with a solid foundation in programming high-quality software. Throughout the semester, I learned essential design paradigms such as abstraction, encapsulation, handling null values, and adopting an agile coding style. These concepts have been crucial in shaping my approach to software development. We also worked extensively on coding projects, with nearly every two weeks requiring us to submit a new assignment. This rigorous schedule taught me how to efficiently manage my time and prioritize tasks, as the course demands significant time commitment.

One of the most valuable lessons from this course was learning how to collaborate with others on designing clean, efficient code. This is especially critical in a professional setting, particularly when working in large companies where teams must coordinate on complex projects. Even though my research in software development is more focused on algorithms for machine learning than on traditional software development, the skills gained from this course have been incredibly beneficial.

In my work, I often need to design frameworks for training machine learning models, and the software engineering techniques I learned, such as writing clear, readable functions, providing meaningful comments, and using good abstraction, have helped me tremendously. Thanks to the hands-on coding experience from this course, I am now able to implement ideas more quickly and effectively.

Additionally, this course has taught me how to read and understand others' code, which is an invaluable skill. In research and development, we often rely on algorithms and code written by others, which can be structured and written in ways that differ from our own approach. It would be inefficient and counterproductive to start writing new code without first checking whether existing solutions work as intended. I recall working through a repository that contained excessive layers of unnecessary abstraction. Applying the knowledge from this course, I was able to navigate and make sense of it, ultimately enhancing my ability to collaborate and contribute to larger projects.

6.0.2 ECE 689: Advanced Topics in Deep Learning (Duke)

I took this graduate course when I was in my senior year at Duke. This course has been one of the most challenging in my entire academic career. I chose to take it because my research is

directly related to Deep Learning. Notably, I skipped the introductory course and went straight to the advanced level, as Duke only offered this course at the time, and I didn't want to miss such a valuable opportunity. I initially expected the course to focus on designing complex neural network structures. However, it turned out to be highly theoretical, emphasizing innovative approaches that integrate mathematics and statistics. I still remember the first lecture on Normalizing Flows—an advanced topic in Deep Learning—where I was immediately met with an overwhelming number of mathematical formulas. At first, I struggled to grasp the theoretical concepts, and the dense mathematical formulations felt daunting. However, I was determined not to give up. I dedicated significant time to self-study, regularly sought guidance from teaching assistants, and gradually built a stronger understanding of the material. Through persistence and hard work, I ultimately excelled in the course, earning an A+ grade. Taking this course was one of the best academic decisions I have made. It not only strengthened my foundational knowledge in Deep Learning but also reinforced my perseverance in tackling complex problems. It made me realize that Deep Learning is more than just stacking models and sparked my interest in conducting research in Statistical Machine Learning and Deep Learning in the future. The course introduced me to key topics such as Attention and Language Models, which are directly related to my signature work. Additionally, the well-structured homework assignments provided hands-on experience with PyTorch, a deep learning framework that I utilized extensively in my research. Furthermore, this course allowed me to connect with the instructor, who has offered invaluable guidance throughout my application journey.

6.0.3 COMPSCI 572: Introduction to Natural Language Processing (Duke)

I took this graduate course during my time at Duke, and it provided a comprehensive introduction to Natural Language Processing (NLP). The course covered both traditional models, such as n-grams, neural networks, CNNs, RNNs, and LSTMs, as well as modern approaches, including decoder-only language models for NLP tasks. This course not only sparked my interest in NLP but also inspired me to apply NLP strategies to other domains, such as speech processing. Through this course, I developed a solid understanding of how decoder-only and encoder-only models work and how to implement them—models that later became central to my research. Additionally, the course delved into key concepts such as causal attention and masking strategies, which are highly relevant to my field. To reinforce learning, the course incorporated regular quizzes that tested my understanding of key topics. The homework assignments provided hands-on experience with PyTorch, which not only made me proficient in the framework but also gave me a strong foundation that later proved invaluable in my research. The assignments were particularly engaging, as each one included a competitive component where students could challenge each other to build the best-performing model. I learned a lot of optimization techniques and new model ideas. Moreover, the professor and teaching staff were highly responsive to my questions, creating a supportive and engaging learning environment that further enhanced my experience. Overall, this course is one of the key courses to provide me with fundamental knowledge of NLP which later becomes essential in my Signature Work.

6.1 Two capstone courses

Capstone 495 and 496 provided me with dedicated time to focus on my research. While I had prior research experience before these courses, my progress was slow as I could only work on it during my limited free time. During my senior year, balancing research alongside preparing application materials for graduate school made it even more challenging to find time. However, since these capstone courses were part of my official coursework, they allowed me to allocate more time to research.

Additionally, these courses gave me the opportunity to communicate regularly with my professor.

Dr. Ming Li and I held weekly meetings to discuss my progress, ensuring that my signature work stayed on track.

Moreover, after completing the two capstone courses and finishing my signature work, I developed a strong interest in Deep Learning. These experiences deepened my passion for the field and inspired me to explore it further, ultimately motivating me to pursue a Ph.D. in the future.