音樂生成 Music Generation

陳奕羶
資訊工程學系
國立暨南國際大學
南投,臺灣
s110321015@ncnu.edu.tw

張簡雲翔
資訊工程學系
國立暨南國際大學
南投,臺灣
s110321018@ncnu.edu.tw

劉德權
資訊工程學系
國立暨南國際大學
南投,臺灣
s110321064@ncnu.edu.tw

Abstract

在現在生成式 AI 蓬勃發展的年代，無論是在文字或者影像的生成式 AI 都有卓越表現的情況下，慢慢也開展對於音樂生成式 AI 的研究，並分成 Symbolic 跟 Audio 的不同資料類型進行研究與生成。透過了解到 Transformer 以及 EnCodec 的原理以及架構後，進而推展並組成 MusicGen 的模型，去實作一個音樂生成的 AI 模型。使用 MusicCap 資料集作為訓練以及測試資料集，並成功實作出一個 MusicGen 的模型架構，在參數量約 1 億左右的情況下得到 FAD 10.82677 的成績。

Keywords—Audio process, Transformer, Music generation, Auto Encoder

I. INTRODUCTION

文化的載體有很多種，有文字、圖片或者音樂，這些載體都是承載人類文明的進步以及傳承，這些年來，各式各樣的研究也正在朝向這樣的方向進行。在人工智慧領域，首先著手與文字，努力製造出一個可以完成 text to text 的人工智慧模型，這些年來也有突破性的進步，像是 ChatGPT-4.0o、Gemini 等各式各樣的聊天機器人問世。

在文字已經有一定程度的突破的同時，圖片的生成也不甘落後，在面對 text to image 的模型，近年來也蓬勃發展，生成出來的圖片都已經足彌以假亂真的地步，像是 Stable Diffusion、Dalle 等模型，在圖片能夠被生成後，自然也慢慢有影片的生成，足見人工智慧對於影像的理解能力與程度。

與此同時，音樂方面的研究也慢慢興起，不可否認的是，音樂是一種複雜的載體，人類在面對文字與影像時，會有遺失補全的能力，但是面對音樂卻是非常挑剔，因此，音樂方面的研究十分緩慢，近年來能夠在 text to music 上產生一定效能的模型只有 MusicLM 以及 MusicGen。

音樂卻是文化載體中，最能夠跨越國界在不同國度中傳遞的美妙樂章，為此我們非常好奇且感到興趣在這個領域，想要嘗試實作出一個基於 MusicGen 的 text to music 的模型。

II. BACKGROUND

電腦處理音樂的方式有兩種，一種是透過 MIDI 協議與樂器進行互動，將音訊中的音高、音量、音色與節奏，轉化成一個 MIDI token，並存在電腦中，檔案副檔名為.midi。

另一種電腦儲存音樂的方式，將類比訊號的波形進行量化分析，轉換成數位訊號，並傳輸給電腦將數位訊號進行儲存。為此，在音樂生成領域上可以從輸入的資訊，將所有的研究分成兩大類，第一類輸入的資料為 MIDI 的檔案格式，因為本身就是 token 的方式儲存，所以稱為 symbolic music generation，另一類則是輸入的資料為.wav 或.mp3 等，是由類比訊號轉換成數位訊號方式儲存的，稱之為 audio generation。

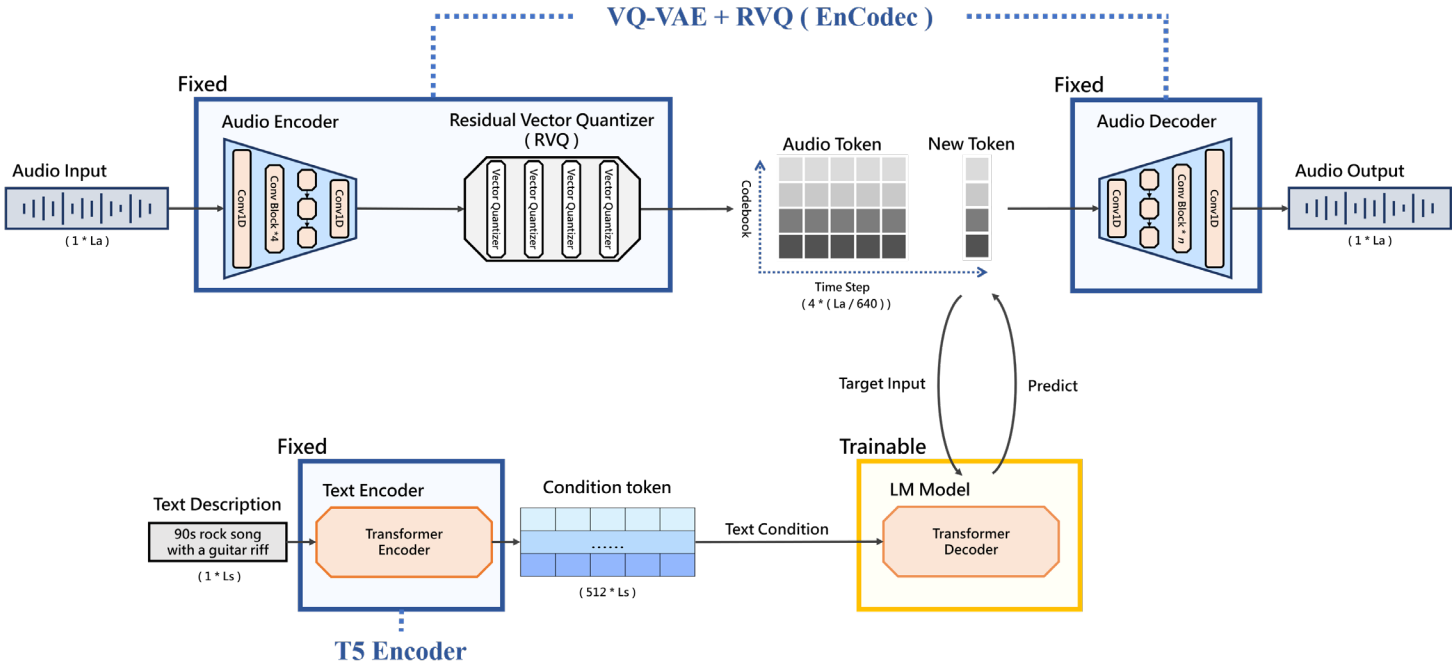
A. Symbolic Music Generation

Symbolic music 的處理相對於 audio 簡單很多，因為本身儲存的方式已經是 token，而單一 token 即可代表一種音高、音量、音色與節奏，所以可以類推成 text to text 的模式進行處裡，所以這方面的研究比較早，且因為 text to text 的研究十分成熟，所以可以利用的模型比較多。所以很常在 NLP(Natural language process)有一定成效的模型都會被拿來試驗看看，像是早期 NLP 研究相對比較風行的 LSTM 模型，也有被用在音樂生成上方，並且生成效果在當時相當不錯[1]。在 2017 年，Google 提出 self-attention 以及 Transformer 的架構[2]，並在自然語言領域上取得非常好的成績，奠定了現在大部分 text to text 的模型基礎，後來 Transformer 也被應用在影像辨識等不同的領域上方，自然也包含音樂生成的領域。

在面對長文字上，原先的 Transformer 的 position encoding 表現略有力不從心的情況，所以有人將原本的 absolute position 的機制改變成 relative position，並在取得一定程度的效能提升[3]，因為音樂本身就是一種很長的字串，所以後來有人提出嘗試用加了 relative position 的 Transformer 進行音樂生成，也取得相當不錯的試驗成效[4]，但是因為 midi 的使用領域較為狹隘，所以應用層面相對比較少。

B. Audio Generation

相較 symbolic music 而言，audio 的處理上就顯得較為困難，因為 audio 是從類比訊號的波形進行量化後儲存的，所以單一的取樣點不能代表音高、音量、音色與節奏，需要由多個取樣點組合才能代表。此外，取樣點的數量除了跟音樂時長有關係之外，也跟 sampling rate 有關聯，sampling rate 越大，每秒儲存的取樣點也就越多。對於 sampling rate 為 32kHz，長度為 10 秒的 audio，則取樣點一共會有 320k 個取樣點，如果我們把這 320k 的取樣點當成字串，並採用 text to text 模式訓練，對於



圖一：MusicGen 架構

硬體的負荷會非常大，且生成效果也不好。所以在處理 audio 的時候，一般都會先經過壓縮，再進行訓練。

如今比較主流的壓縮方法是透過 Auto Encoder 的 Encoder 進行壓縮，得到 latent code 後再進行訓練。由於圖片或音樂的性質是離散而非連續的，所以這些資料在壓縮上較常被採用的是使用 VQ-VAE 的模型架構[5]，將資料經由 Encode 壓縮至連續的 latent code 後，會再經由 Quantizer 量化成離散的 latent code。這些離散的 latent code 分別對應到 codebook 中某個 centroid 的值。在音訊壓縮的領域上，為了減少硬體的負荷，所以會使用 residual vector quantizer 的架構，經由多個 Quantizer 將連續的 latent code 由多個 codebook 中 centroid 的值組合。例如近期的 Google 的 SoundStream[6]跟 Meta 的 EnCodec [7]就是採取此方法對音訊進行壓縮。將音訊壓縮過後，text to text 的訓練難度就降低了許多，隨後也接著出現了 MusicLM[8]及 MusicGen[9]等音樂生成模型，並在此領域獲得了不錯的成效。

III. Model Architecture

A. Overview

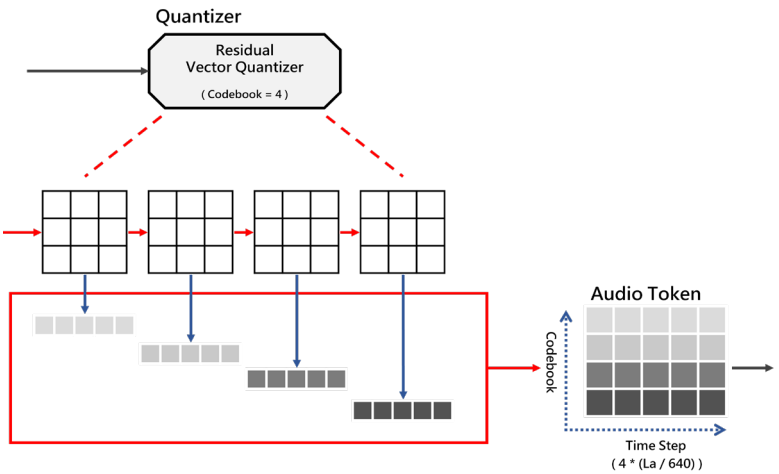
模型所使用的架構是基於 MusicGen 架構所設計的（圖一），由以下模組組成：Text Encoder、VQ-VAE + RVQ (EnCodec)（Audio Encoder、Quantizer、Audio Decoder）、LM Model。部分模組是 Trainable（圖一黃色框框），部分則是 Fixed（圖一藍色框框），其中 Fixed 的部分有：Text Encoder、VQ-VAE + RVQ (EnCodec)，Trainable 的部分有：LM Model。

B. Text Encoder

Text Encoder 主要負責將 text condition 轉成 condition token。本次的模組使用 T5 Encoder，主要架構為 Transformer Encoder，為 pretrained model，參數量為 50M，產生的 condition token 的 shape 為 512 * *Ls*，*Ls* 為 text condition 的長度。為避免模型參數量太大，所以將 T5 Encoder 的參數 fix，不讓其參與訓練。

C. VQ-VAE + RVQ (EnCodec)

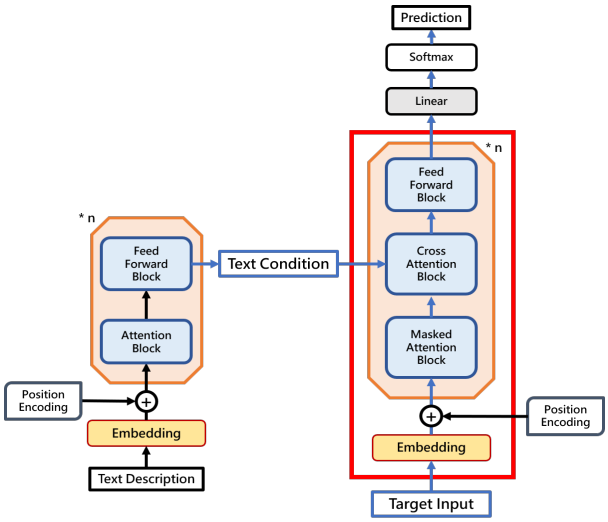
VQ-VAE + RVQ (EnCodec)分成兩個部分，第一部分：audio encoder、quantizer 主要負責將 audio input 轉成 audio token，第二部分：audio decoder 主要負責將 audio token 轉成 audio output；audio input 與 audio output 皆為音訊檔案，sampling rate 為 32kHz，副檔名為.wav。本次的模組使用 EnCodec，為 pretrained model，壓縮倍率為 640 倍，例如：長度 1 秒，sampling rate 為 32kHz 的音訊，經由 audio encoder、quantizer 壓縮過後，會產生 frame rate 為 50 的 audio token。為了避免運算量及記憶體使用量過大，所以 quantizer 採用的是 residual vector quantizer（RVQ），使用的 codebook 一共有 4 個（圖二），每個 codebook 有 2048 個 centroid。產生的 audio token，值為對應 codebook 的 centroid 的 index；shape 為 4 * (*La* / 640)，4 為 codebook 的數量，*La* 為 32k * 音訊秒數，640 為音訊壓縮倍率。EnCodec 的參數是 fixed，沒有參與訓練過程。



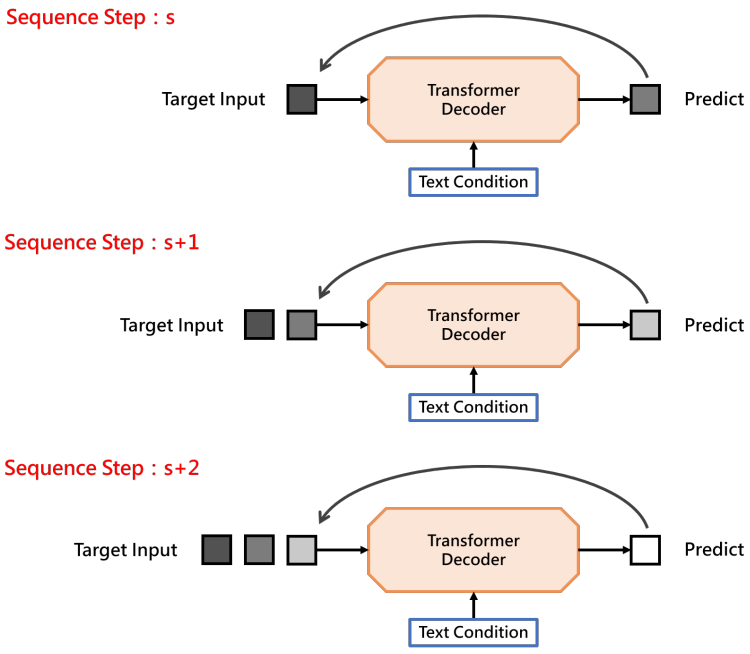
圖二：RVQ 架構

D. LM Model

LM model 主要負責預測 audio token 4 個 codebook 的機率分布，並且根據這 4 個機率分布，從 RVQ 中對應的 codebook 取出對應的 index 產生新的 audio token。本次的模組使用 Transformer Decoder（圖三紅色框框），為 autoregressive decoder 架構：在 sequence step s 時 autoregressive decoder 會產生 output，如果是在 training state，則會把產生 output 對應的 ground truth 當作 input 預測 sequence step $s+1$ 的 output，反之則將產生的 output 當作 input（圖四）。共有兩個 input：target input、text condition。模組架構由以下幾個部分組成：embedding layer、positional encoding layer、masked mutihead self-attention block、多個 Decoder block、linear layer 及 softmax layer。



圖三：Transformer 架構



圖四：Autoregressive Decoder 運作方式

Embedding layer、positional encoding layer 負責獲取 target input 每個 entry 的位置訊息與 embedding。Masked mutihead self-attention block 由 masked mutihead self-attention layer 與 feedforward block 組成，負責對 target input 進行 masked self-attention 運算。decoder block 由 cross-attention layer 與 feedforward block 組成，負責對 target input 與 text condition 進行 cross-attention 運算。Linear layer 及 softmax layer 負責計算機率分布。Feedforward block 由兩層 fully connected layer 組成。

E. Codebook Pattern

由於本次模型使用的 EnCodec 模組，codebook 有四個，所以 LM model 在預測機率分布的時候需要分別針對 4 個 codebook 進行計算，為此，此模型引入了 Codebook Pattern 的概念。為方便說明，先進行符號定義：如果符號上方帶有~，例如$\tilde{V}$，則此符號屬於 ground truth；反之則為模型產生的 output。此外，下面皆假設是在 training state，也就是模型在 sequence step s 產生 output V，並且把$\tilde{V}$當成 input 重新輸入回去。

假設只有一個 codebook 的情況下，一個 audio token，用$\tilde{V}_t$表示，代表這個 audio token 在第 t 個 time step 下，codebook 中對應到的 centroid index。由於 codebook 只有一個，所以可以當成一般模型預測 sequence 的方式來實作：LM model 在 sequence step s 時預測 time step $s+1$ 的 audio token，也就是$\tilde{V}_{s+1}$。此時 LM model 的兩個 input：target input，用$\tilde{V}$表示，text condition，用 U 表示。如果在 sequence step 0 並要預測$\tilde{V}_1$時，會輸入 SOS token 給 LM model。LM model 預測V_s的機率分布如下：

$$P(V_s \mid \mathrm{SOS}, \tilde{T}_{s-1}, U) \tag{1}$$

$$\tilde{T}_{s-1} = \{\tilde{V}_1, \tilde{V}_2, \ldots, \tilde{V}_{s-1}\} \tag{2}$$

$$U = \{U_i, 0 < i < Ls\} \tag{3}$$

在 sequence step 0 的時候，LM model 會根據 SOS 與 U 預測 V_1，sequence step 1 的時候，LM model 會根據 SOS、$\tilde{V}_1$與 U 預測 V_2，在 sequence step 2 的時候，LM model 會根據 SOS、$\tilde{V}_1$、$\tilde{V}_2$與 U 預測 V_3，在 sequence step 3 的時候，LM model 會根據 SOS、$\tilde{V}_1$、$\tilde{V}_2$、$\tilde{V}_3$與 U 預測 V_4，以此類推。

當 codebook 數目增加到 4 個的時候，第 t 個 time step 下的 audio token，用$\tilde{V}_t = \{\tilde{V}_t^c, c = 1 \sim 4\}$ 表示，$\tilde{V}_t^c$ 代表這個 audio token 在 time step t 下，第 c 個 codebook 中對應到的 centroid index。這個情況下，如果要產生一個完整的 audio token，需要 4 個不同 codebook 的值，而 LM model 需要預測 audio token 在不同 time step，不同 codebook 下的值。比較直觀的做法是參考 Eq.(1)：將 codebook 攤平，此時的 codebook 則稱之為 flattening pattern（圖五 (a)）。讓 LM model 用 4 個 sequence step 預測在某個 time step 下 audio token 的 4 個 codebook 值，在 sequence step s 時，對第$(s+1)/4 + 1$ time step 的 audio token 的第$(s+1) \bmod 4 + 1$ 個 codebook 進行預測，也就是$\tilde{V}_{(s+1)/4+1}^{(s+1) \bmod 4 + 1}$。此時 LM model 的兩個 input：target input，用$\tilde{V}$表示，text condition，用 U 表示。如果

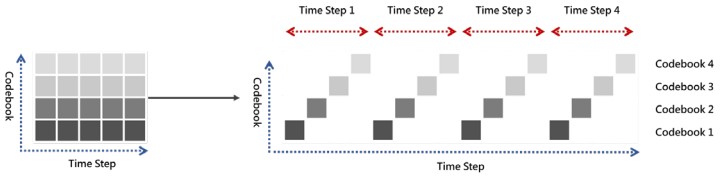
在 sequence step 0 並要預測 V_1^1 時，會輸入 SOS token 給 LM model。LM model 預測 $\tilde{V}_{s/4+1}^{s \bmod 4+1}$ 的機率分布如下：

$$\mathrm{P}\left(V_{s/4+1}^{s \bmod 4+1} \mid \mathrm{SOS}, \tilde{V}_{s-1}, U\right) \tag{4}$$

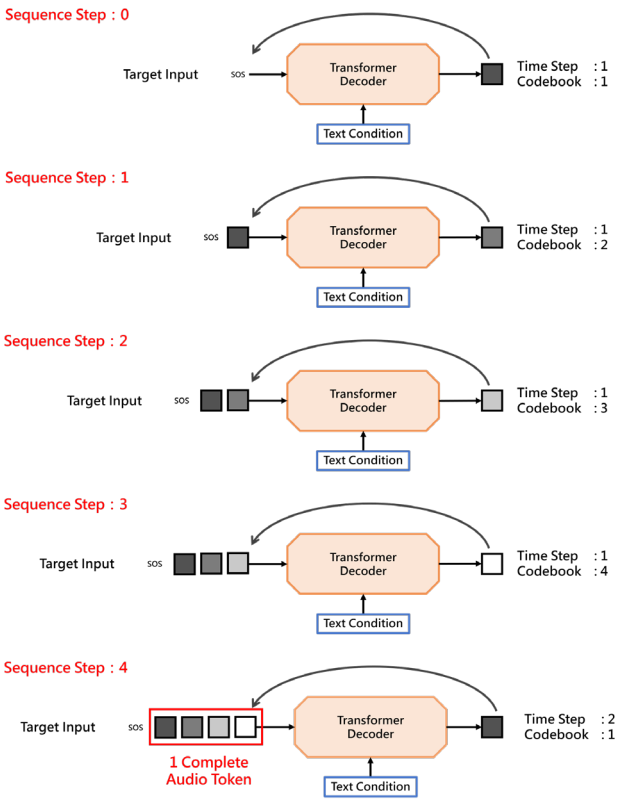
$$\tilde{V}_{s-1} = \{\tilde{V}_1^{1\sim4}, \tilde{V}_2^{1\sim4}, \ldots, \tilde{V}_{(s-1)/4+1}^{(s-1) \bmod 4+1}\} \tag{5}$$

$$U = \{U_i, 0 < i < Ls\} \tag{6}$$

在 sequence step 0 的時候，LM model 會根據 SOS 與 U 預測 V_1^1，sequence step 1 的時候，LM model 會根據 SOS、$\tilde{V}_1^1$ 與 U 預測 V_1^2，在 sequence step 2 的時候，LM model 會根據 SOS、$\tilde{V}_1^1$ 、$\tilde{V}_1^2$ 與 U 預測 V_1^3，在 sequence step 3 的時候，LM model 會根據 SOS、$\tilde{V}_1^1$ 、$\tilde{V}_1^2$、$\tilde{V}_1^3$ 與 U 預測 V_1^4，這時在 time step 1 的 audio token 才會是完整的，並以此類推（圖五 (b)）。然而這種做法有個缺點：如果要預測 n 個完整的 audio token，則總共要花 $n * 4$ 個 sequence step 才能預測完，相對耗時。



(a) Fattening Pattern



(b) Inference

圖五：Flattening Pattern (a) 與 Inference 過程 (b)

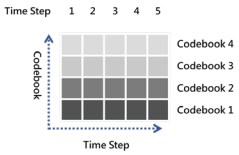
另一個做法是同時對某個 time step 下的 audio token 的 4 個 codebook 進行預測：LM model 在 sequence step s 時預測 time step $s + 1$ 下 audio token 的四個 codebook 的值，也就是 $\tilde{V}_{s+1} = \{\tilde{V}_{s+1}^1, \tilde{V}_{s+1}^2, \tilde{V}_{s+1}^3, \tilde{V}_{s+1}^4\}$。此時 LM model 的兩個 input：target input，用 $\tilde{V}$ 表示，text condition，用 U 表示。如果在 sequence step 0 並要預測 $\tilde{V}_1$ 時，會輸入 SOS token 給 LM model。因此，LM model 在 sequence step $s - 1$ 所預測 $\tilde{V}_s$ 的機率分布如下：

$$\mathrm{P}\left(V_s \mid \mathrm{SOS}, \tilde{V}_{s-1}, U\right) \tag{7}$$

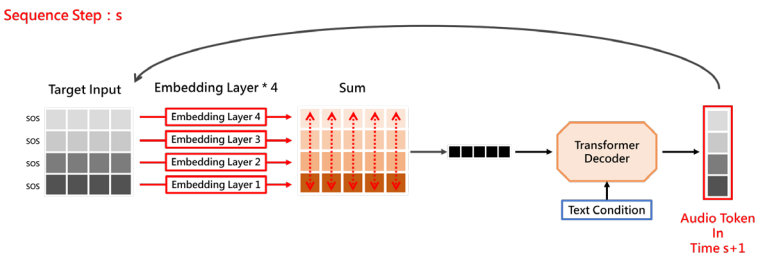
$$\tilde{V}_{s-1} = \{\tilde{V}_1^{1\sim4}, \tilde{V}_2^{1\sim4}, \ldots, \tilde{V}_{(s-1)}^{1\sim4}\} \tag{8}$$

$$U = \{U_i, 0 < i < Ls\} \tag{9}$$

在 sequence step 0 的時候，LM model 會根據 SOS 與 U 預測 V_1，sequence step 1 的時候，LM model 會根據 SOS、$\tilde{V}_1$ 與 U 預測 V_2，在 sequence step 2 的時候，LM model 會根據 SOS、$\tilde{V}_1$ 、$\tilde{V}_2$ 與 U 預測 V_3，在 sequence step 3 的時候，LM model 會根據 SOS、$\tilde{V}_1$ 、$\tilde{V}_2$、$\tilde{V}_3$ 與 U 預測 V_4，以此類推。此時的 codebook 稱之為 parallel pattern（圖六 (a)）。因為 output 有 4 個，每個對應不同的 codebook。如果將這 4 個 output 當成新的 target input 重新輸入給 LM model 並預測 sequence step $s + 1$ 的 output，此時要做一些處理：target input 的不同 codebook 的值會分別經過不同 embedding layer 的，並把四個 embedding layer 的 output 做加總（圖六 (b)）。



(a) Parallel Pattern



(b) Inference

圖六：Parallel Pattern (a) 與 Inference 過程 (b)

對於 parallel pattern，由於模型的效能可能太強導致生成的效果不好：在相同 time step 下的 audio token，其四個 codebook 的值是有高度關聯性的，如果採用 parallel pattern，可能會有 overfitting 的情況，也就是相同 time step 下 audio token 的四個 codebook 的值的多樣性不高，導致生成的音樂品質不夠好。所以引入了 delay pattern 的概念[10]：對 parallel pattern 中 4 個 codebook 的 code 向右平移 c - 1，c 為 code 對應的 codebook，例如

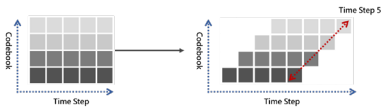
codebook 4 的 code 會向右移動 3 格，此時的 codebook 稱之為 delay pattern（圖七 (a)）。LM model 在 sequence step s 時預測值就變成 $\tilde{C}_{s+1}=\{\tilde{V}^{c}_{(s+1)-c+1}, c=1\sim4\}$，換句話說，在 sequence step s 時 time step s - 3 的 audio token 的四個值才會全部預測完畢。此時 LM model 的兩個 input：target input，用 $\tilde{C}$表示， text condition，用 U 表示。如果在 sequence step 0 並要預測$\tilde{V}_1$時，會輸入 SOS token 給 LM model。因此，LM model 在 sequence step $s-1$ 所預測$\tilde{C}_s$的機率分布如下：

$$P(C_s \mid \text{SOS}, \tilde{C}_{s-1}, U) \tag{10}$$

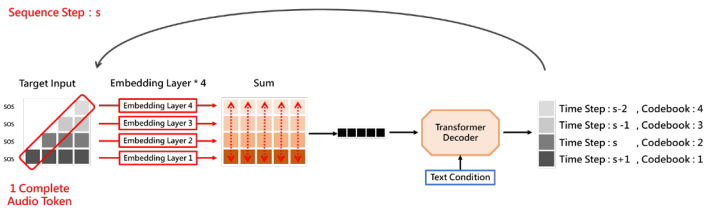
$$\tilde{C}_{(s-1)}=\{\tilde{V}_1^{1\sim4}, \tilde{V}_2^{1\sim4}, \ldots, \tilde{V}_{s-4}^{1\sim4}, \tilde{V}_{s-3}^{1\sim3}, \tilde{V}_{s-2}^{1\sim2}, \tilde{V}_{s-1}^{1}\} \tag{11}$$

$$U=\{U_i, 0<i<Ls\} \tag{12}$$

在 sequence step 0 的時候，LM model 會根據 SOS 與 U 預測 C_1， sequence step 1 的時候，LM model 會根據 SOS、C_1 與 U 預測 C_2，在 sequence step 2 的時候，LM model 會根據 SOS、C_1 、C_2 與 U 預測 C_3，在 sequence step 3 的時候，LM model 會根據 SOS、C_1 、C_2、C_3 與 U 預測 C_4，以此類推。同樣，target input 的不同 codebook 的值會分別經過不同 embedding layer 的，並把四個 embedding layer 的 output 做加總（圖七 (b)）。



(a) Delay Pattern



(b) Inference

圖七：Delay Pattern (a) 與 Inference 過程 (b)

F. Loss Function

本次模型使用的 loss function 為 cross entropy，LM model 預測的機率分布，會與 ground truth 計算 loss。LM model 在 sequence step s 時的產生的四個 codebook 的分布$P_{c=1}\sim P_{c=4}$會跟 ground truth $\tilde{C}_{s+1}=\{\tilde{V}^{c}_{(s+1)-c+1}, c=1\sim4\}$計算 loss， loss term 如下：

$$\text{Loss}=-\frac{1}{4}\log\left(P_{c=1}(\tilde{V}^{1}_{s+1})+P_{c=2}(\tilde{V}^{2}_{s})+P_{c=3}(\tilde{V}^{3}_{s-1})+P_{c=4}(\tilde{V}^{4}_{s-2})\right) \tag{13}$$

四個 codebook 的值皆會跟其對應的 audio token 的 codebook ground truth 計算 loss 並相加平均。

IV. Dataset and Training

A. Introduction

MusicCap 資料集包含 5521 首音樂，檔案格式均為無壓縮的.wav 檔，sampling rate 為 48kHz，雙聲道，曲風包羅萬象，從柔和抒情到搖滾重金屬均有，內容部分有些為包含人聲之內容，亦有單純只有音樂內容，資料集內容主要收集 YouTube 上的影音資料。這份資料集被用在 MusicLM 中當作 training set 以及 test set[8]，並在 MusicGen 中當作 test set 之一[9]。每首音樂均有相對應的英文關鍵字，如「pop, tinny wide hi hats, mellow piano melody, high pitched female vocal melody, sustained pulsating synth lead」及相對應的 caption，如「A low sounding male voice is rapping over a fast paced drums playing a reggaeton beat along with a bass」，透過上述兩點便可用來形容這首音樂的特徵。因為 MusicGen 所使用的 Dataset 皆需要付費，為了成本考量，選用本份資料集作為 training set 以及 test set。

B. Data Splitting

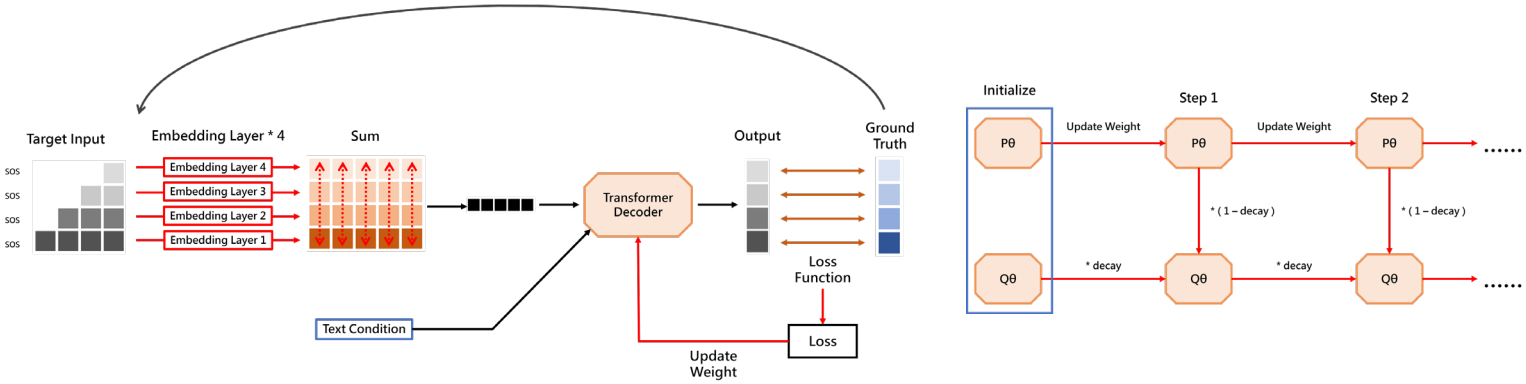
在下載下來的資料集中，只有 5419 筆資料，所以就用這 5419 筆資料作為我們資料集。藉由資料有無經過過濾，分成兩個版本，第一版就是沒經過任何過濾，所有 Dataset 中的資料都拿來使用的，先將所有的資料進行打亂後，其中先切出 80%的資料作為 training set，剩下的 20%作為 test set，在 training set 中再切出 20%作為 validation set，換言之，最後的資料集分布，所有資料的 65%作為 training set、15%作為 validation set、20%作為 test set。

第二版資料集是有經過人工篩選後的資料集，將第一個版本的 training set、validation set、test set，逐一篩選掉音檔中有出現人聲、雜訊或者明顯不是音樂的內容，經果篩選後，training set 中有 1343 筆資料、validation set 中有 302 筆資料、test set 中有 360 筆資料。

C. Data Preprocessing

Data Preprocessing 分為兩個部分，一個 text description，使用 dataset 的 caption；另一個是 audio input，使用 dataset 的音樂。由於 T5 Encoder 與 EnCodec 的參數是 fixed，為了減少訓練時間，所以本次實驗將 text description 轉成 condition token，audio input 轉成 audio token。本次模型採用 classifier free guidance，所以在 text description 上，會先根據 word drop rate 隨機 drop 一些 text（圖十二），取前 250 個 text，不足的 padding 0（Ls = 250），並輸入進 text encoder 產生 condition token，word drop rate 為 0.2。Audio input 的部分則只取前 5 秒並從雙聲道轉成單聲道，sample 降至 32kHz（La = 5*32000）並透過 EnCodec 轉成 audio token。因為本次實驗以 delay pattern 進行訓練和生成，所以以 delay pattern 的形式儲存以便後續訓練。training set 的 audio token 會被當成 ground truth 用於訓練中計算 loss，validation set 與 test set 的 audio token 則會被用於計算 performance metrics。

對 training set、validation set、test set 資料集獨立處理，為避免處理及訓練過程中記憶體使用量過大，所以



(a) 模型訓練　　(b) 參數更新

圖八：模型訓練 (a) 與 參數更新過程 (b)

採分批處理的方式：每次處理 30 筆資料並以 1 個 batch 的形式儲存。

D. *Training*

訓練過程中，從每個 batch 固定取前 n 筆資料，此時的 batch size 為 n，為避免訓練中 loss 變動過大，於是採用 gradient accumulation 的方法，每累計 b 個 batch 的 loss 才進行一次 back backpropagate，如此一來，就可以確保只會占用 n 筆資料的記憶體使用量，但是 batch size 可以提高到 $n * b$。

訓練過程中，training set 的 condition token 會當成 text condition，audio token 會當成 ground truth。在 sequence step s 的時候，LM model 會根據 target input 及 text condition 產生 4 個 output：V_s^1、V_{s-1}^2、V_{s-2}^3、V_{s-3}^4，此時這 4 個 output 會分別與對應的 ground truth $\tilde{V}_s^1$、$\tilde{V}_{s-1}^2$、$\tilde{V}_{s-2}^3$、$\tilde{V}_{s-3}^4$帶入 loss function 計算 loss 並更新模型參數，此外$\tilde{V}_s^1$、$\tilde{V}_{s-1}^2$、$\tilde{V}_{s-2}^3$、$\tilde{V}_{s-3}^4$會被當成新的 target input 重新輸入給 LM model（圖八 (a)）。

為了確保訓練過程的隱定性，本次模型的參數還會透過 exponential moving average 維護：有兩組參數 P_θ、Q_θ，P_θ，二者參數在初始化的時候是一樣的，訓練過程中會以 P_θ 產生的 output 計算 loss 並以此更新 P_θ 的參數。在更新完 P_θ 後，Q_θ 的值會以下列式子更新

$$Q_\theta = Q_\theta * \text{decay} + P_\theta * (1 - \text{decay}) \quad (14)$$

，decay 為 0.99（圖八 (b)）。最終模型所採用的參數為 Q_θ。

V. Experiment

A. *Inference*

Inference 的過程，首先模型會 load 參數 Q_θ，validation set 與 test set 的 condition token 會被當成 text condition。模型會根據 text condition 產生 output，此時 output 的四個機率分布會各自進行 sample，隨機取出對應 codebook 的 index 並當成新的 target input 重新輸入給模型。sample 的方式採用 top-k，k=250，temperature=1.0 的方式 sample：先選出機率前 250 大的 index，再搭配 temperature 進行 sample 出對應 codebook 的 index。此外，本次 inference 過程也加入 classifier free guidance 機制：

在 sequence step s 的時候，LM model 會根據 target input 及 text condition 生成機率分布，$P_{conditional}$，根據 target input 生成機率分布，$P_{unconditional}$，接著對 $P_{conditional}$、$P_{unconditional}$ 進行下列運算

$$P = \gamma * P(C_{s+1} \mid SOS, C_s) + (1 - \gamma) * P(C_{s+1} \mid SOS, C_s, U) \quad (15)$$

γ 為 guidance scale，設定為 0.2，之後對 P 進行 sample。

B. *Model and Hyperparameter*

模型中有需多可以調整的 hyperparameter，根據調整不同的 hyperparameter，會有不同的參數量以及不同的效果，分別介紹不同的 hyperparameter 對於模型架構上的影響，以及兩種不同的 hyperparameter 調整出來的模型參數。

在訓練過程中，可以設定訓練的 epoch 數量，我們所有模型統一設定為 epoch 數量為 300。d_model 是代表模型輸入的時候，經過的 embedding layer 後輸出的維度，以及 masked mutihead self-attention layer 與 cross-attention layer。nheads 是代表在模型中 self-attention 的 head 數量。nlayer 代表在模型中，合計堆疊多少次 attention block 的數量（圖三紅色框框中的 * n）。d_hid 代表在模型中 feed forward 中第一層 fully connected layer 輸出的維度大小。

基於上述的 hyperparameter，我們設計了三種不同的模型，主要是想要測試在小參數量輸入較少的資料進入訓練，大參數量輸入較多的資料進行訓練。主要會先調整 nlayer 去拉高層數，並適度的增加 d_hid 去增加每個 attention block 各層傳遞的資訊，並在資料量更大，更加複雜後去將增加 d_model 以及 nheads 去增加整體模型的維度。

	d_model	nheads	nlayer	d_hid	Total parameter
MusicGen-1	1024	8	8	2048	119,171,072
MusicGen-2	2048	16	16	4096	844,689,408

表一：各種模型 Hyperparameter 數值以及參數量

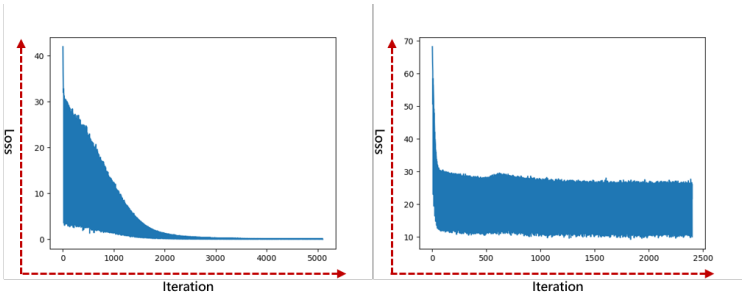
C. *Training Result*

資料分為有經過人工篩選以及沒有人工篩選，有經過人工篩選的資料稱作 Filtered Data，沒有經過人工篩選的資料稱作 Non-Filtered Data。模型有經過不同的 hyperparameter 調整出來的模型。資料數量透過取預先

處理好的 batch 中取出前 n 筆去控制，分成 300 筆跟 600 筆資料。

在訓練過程中先以 Non-Filtered Data 進行訓練，在進行 Filtered Data 的訓練。並以 300 筆資料去訓練 MusicGen-1 以 600 資料去訓練 MusicGen-2，訓練後以 loss 繪製 learning curve，得到兩張不同的 learning curve（圖九），左側的為 300 筆資料訓練 MusicGen-1，右側的為 600 筆資料訓練 MusicGen-2，縱軸為 loss，橫軸則為 iteration，每 18 iterations 等於 1 epoch。

可以發現兩個模型訓練上的極大差異，在 300 筆資料訓練 MusicGen-1 的 learning curve（圖九 (a)）可以發現，loss 雖然有一定程度的震盪，但都是有再慢慢收斂直到小於 1。但在 600 筆資料訓練 MusicGen-2 的 learning curve（圖九 (b)），可以發現在 loss 下降到 30 左右後，就會一直在 30 跟 10 之間震盪，並沒有在收斂，且整體 loss 的數值很高，考量到算力資源，就沒有讓他繼續練完 300 epoch。

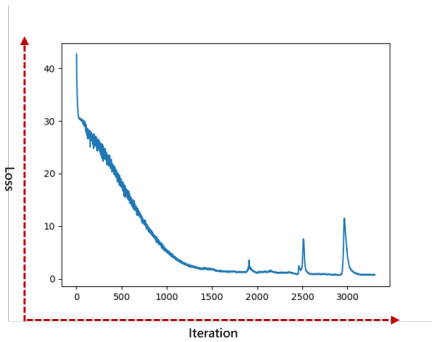


(a) MusicGen-1 300 筆資料　　(b) MusicGen-2 600 筆資料

圖九：Learning Curves using Non-Filtered Data

在測試將 300 筆資料輸入到 MusicGen-2 的模型中，我們發現基本上 learning curve 跟原來幾乎一樣，可能是因為模型層數增加導致梯度消失，或者因為參數量太大，導致於模型本身可能非常難以收斂情況。但在增大 learning rate 效果仍舊十分不明顯，所以就放棄以 600 筆資料訓練 MusicGen-2 的模型。

在經過 Non-Filtered Data 的測試後，我們針對於 Filtered Data 只使用 300 筆資料訓練 Filtered Data，並在以 loss 繪製 learning curve（圖十），縱軸一樣是 loss 橫軸，是 iteration，並以 11 iteration 等於 1 epoch。可以發現這次震盪的情況，相較於 Non-Filtered Data 明顯降低很多。可能是因為經過過濾後的資料較為純粹，所以使模型在學習音樂上，比較沒有受到其他聲音的干擾。



MusicGen-1 300 筆資料

圖十：Learning Curve of MusicGen-1 using Filtered Data

VI. Result

A. Performance Metric

在音樂生成領域，要比較一首音樂好聽與否，一種做法是透過人耳主觀評斷，而在客觀評價上，一種常見的做法是參考 FID Score 的做法：將原始音樂與模型生成的音樂分別計算分布，並且計算二者分布的距離[11]。為此，本次採用的 Performance Metric 為 FAD score：會有兩組資料，一組是原始資料，來自 test set 和 validation set，不參與訓練過程，另一組是模型生成的音樂。將兩組資料個別輸入給 VGGish 模型，這個模型會個別對兩組資料的每首音樂進行特徵擷取並算出分布，接著對兩組資料的機率分布計算 frechet-audio-distance，得到兩組分布的距離。FAD score 越小代表兩組的分布越近，也代表模型的生成表現越好。

B. Melody evaluation

在經過模型訓練後，最終兩種不同版本的資料都選定以 MusicGen-1 訓練出來的模型進行測試，將兩個訓練出來的模型個別與兩個不同版本的資料的 validation set 計算 FAD。在 Non-Filtered Data 上 FAD 得到 7.70155，在 Filtered Data 上 FAD 得到 11.05481。相比於同樣使用 MusicCap 作為 Test 的 Riffusion 的 FAD 14.8 優秀不少，但仍舊遜色於 Mousai 的 FAD 7.5、3 億參數量 MusicGen 的 FAD 3.1 以及 15 億參數量的 FAD 3.4 [9]。由於本次實驗所使用的資料數量僅有 600 筆，且參數量也較少，所以 FAD 的分數沒有非常好，但就實驗結果而言，此次模型生成的也已經可以產生清晰可辨識的音樂了。因為 Filtered Data 與我們主題目標較為接近，且生成效果較好，所以選定以 Filtered Data 訓練出來的模型最為最終模型。

C. Demo Music

選定 Filtered Data 的 MusicGen-1 的模型為最終模型後，經過與 test set 計算 FAD 得到 10.82677。並將 test set 的 text condition 輸入到模型中生成出多首音樂，經過專題成員為主觀投票後，選出三首較為不錯的音樂，作為前三首實驗成果展示。之後再自行設計一些 text condition 輸入到模型中，並經過專題成員主觀投票後，選出兩較為不錯的音樂，並為後兩首實驗成果展示。有標底線的段落，是專題成員主觀認定上，模型生成的音樂與文字描述較有相關的部分。

Text Condition	Music
The low quality recording features a drum & bass song that consists of a repetitive female vocal sound effect, growl synth bass, manic percussions, shimmering hi hats, punchy kick and snare hits and echoing synth pad. It sounds energetic, aggressive and manic - like something you would hear in underground clubs during the 00s.	
This is an advertisement jingle music piece. It is an instrumental piece. The main theme is being played by the piano while there is a synth string sound in the melodic background. There is an emotional, heart-touching atmosphere. This piece could be used in the soundtrack of a drama movie during scenes of tragedy. It could also work well as an advertisement jingle where there is an attempted appeal to emotion.	

The low quality recording features a live performance that consists of bagpipes melodies. It sounds soulful, passionate, cultural and the recording is in mono and noisy.	
It sounds energetic, aggressive and manic - like something you would hear in underground clubs during the 00s.	
It sounds soulful, passionate, cultural and the recording is in mono and noisy.	

表二：展示用音樂

VII. Conclusion

透過本次實驗，我們從 Transformer 開始，一路了解到後面音訊壓縮常用到的模型 EnCodec ，並進而推展到後來的 MusicGen 整體架構，對於無論是生成式 AI，還是音訊處理上都有更深刻的認識以及體會，並在最後成功實作出 MusicGen 的整體架構，並使用 MusicCap 資料集進行訓練以及測試，並得到 FAD 為 10.82677 的成績，雖然主觀上聆聽音樂內容還是多有可以進步的空間，但是總體成果仍舊有一定表現。

在這次實驗的過程中，沒有成功改善對於大參數量上的訓練，loss 無法降低的問題，礙於算力以及時間沒有辦法做更深入的測試，但我們推測應該是跟資料集的選擇，或者是 hyperparameter 的調整上，以及部分 fixed 的 model 需要去改成 trainable 有關，如若在未來有人對於本項研究有更多興趣，並且具備更多實驗資源以及時間，可以嘗試朝向這樣的方向著手。

生成式 AI 仍舊是現在人類一再攻克的主題之一，在面對音樂生成上更加複雜且難以突破，但是這些生成式 AI 都是在為了讓人類可以在生活上、創造上有更加便捷的工具，音樂生成在現階段雖然不像是文字或影像生成一般可以達到以假亂真的地步，仍有很高的進步空間，但是現在的我們跨出了第一步，象徵的未來逐漸成長的人工智慧時代，並期許在未來可以看到完成以假亂真等級的音樂生成 AI 出現。

References

[1] Oore, S., Simon, I., Dieleman, S., Eck, D., & Simonyan, K. (2018). This time with feeling: learning expressive musical performance. Neural Computing and Applications, 32, 955 - 967.

[2] Vaswani, A., Shazeer, N.M., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., & Polosukhin, I. (2017). Attention is All you Need. Neural Information Processing Systems.

[3] Shaw, P., Uszkoreit, J., & Vaswani, A. (2018). Self-Attention with Relative Position Representations. North American Chapter of the Association for Computational Linguistics.

[4] Huang, C.A., Vaswani, A., Uszkoreit, J., Shazeer, N.M., Simon, I., Hawthorne, C., Dai, A.M., Hoffman, M.D., Dinculescu, M., & Eck, D. (2018). Music Transformer: Generating Music with Long-Term Structure. International Conference on Learning Representations.

[5] Oord, A.V., Vinyals, O., & Kavukcuoglu, K. (2017). Neural Discrete Representation Learning. ArXiv, abs/1711.00937.

[6] Zeghidour, N., Luebs, A., Omran, A., Skoglund, J., & Tagliasacchi, M. (2021). SoundStream: An End-to-End Neural Audio Codec. IEEE/ACM Transactions on Audio, Speech, and Language Processing, 30, 495-507.

[7] D'efossez, A., Copet, J., Synnaeve, G., & Adi, Y. (2022). High Fidelity Neural Audio Compression. ArXiv, abs/2210.13438.

[8] Agostinelli, A., Denk, T.I., Borsos, Z., Engel, J., Verzetti, M., Caillon, A., Huang, Q., Jansen, A., Roberts, A., Tagliasacchi, M., Sharifi, M., Zeghidour, N., & Frank, C.H. (2023). MusicLM: Generating Music From Text. ArXiv, abs/2301.11325.

[9] Copet, J., Kreuk, F., Gat, I., Remez, T., Kant, D., Synnaeve, G., Adi, Y., & D'efossez, A. (2023). Simple and Controllable Music Generation. ArXiv, abs/2306.05284.

[10] Kharitonov, E., Lee, A., Polyak, A., Adi, Y., Copet, J., Lakhotia, K., Nguyen, T., Rivière, M., Mohamed, A., Dupoux, E., & Hsu, W. (2021). Text-Free Prosody-Aware Generative Spoken Language Modeling. ArXiv, abs/2109.03264.

[11] Kilgour, K., Zuluaga, M., Roblek, D., & Sharifi, M. (2018). Fréchet Audio Distance: A Metric for Evaluating Music Enhancement Algorithms. ArXiv, abs/1812.08466.