



COMP4801 Final Year Project
Interim Report

Autoustic: An AI model for generating
acoustic adaptations of musical compositions

Supervisor: Dr. Choi Yi King

Group: fyp23028

Student: Cheng Wai Fung (3035705427)

Copyright Acknowledgment

Copyrighted songs were used in this project for training purposes. Due to copyright issues, the dataset used in this project will not be published and the complete list is available in Appendix 1. Note that the models published in this project are only for individual research purposes.

Table of Contents

Copyright Acknowledgment	2
Table of Figures	4
Table of Tables	5
Abbreviations	6
1. Introduction	7
2. Objectives	7
3. Background	7
3.1 Related Works	7
3.1.1 Jukebox.....	7
3.2 Adopted/Experimented Models	8
3.2.1 Audiocraft EnCodec.....	8
3.2.2 Audiocraft MusicGen.....	8
3.2.3 Descript Audio Codec (DAC).....	10
4. Works Accomplished	10
4.1 AI Models	10
4.1.1 Approach 1 – MusicGen with DAC	10
4.1.2 Approach 2 – Audio Splitting.....	14
4.1.3 Models Summary	15
4.2 Web App	15
4.2.1 Frontend	15
4.2.2 Backend	15
4.2.3 List of AWS resources	16
4.2.4 User Flow.....	17
5. Future Works	20
Appendix 1	22
References	25

Table of Figures

Figure 1 Architecture of EnCodec [2]	8
Figure 2 Architecture of MusicGen [3].....	9
Figure 3 Generate music with MusicGen with text and melody conditioning	9
Figure 4 Approach 1 model architecture	10
Figure 5 Cross-entropy loss of the model	12
Figure 6 Perplexity of the model.....	13
Figure 7 Model architecture of approach 2	14
Figure 8 AWS architecture	15
Figure 9 Sign in page	18
Figure 10 Create account page	18
Figure 11 User home page	19
Figure 12 Project timeline	20

Table of Tables

Table 1 Example metadata of Take On Me.....	11
Table 2 Dataset parameters and configurations	11
Table 3 Language model parameters and configurations	12
Table 4 Optimizer parameters and configurations (Unchanged)	12

Abbreviations

ABBREVIATIONS	DEFINITION
AI	Artificial Intelligence
Amazon EC2	Amazon Elastic Compute Cloud
Amazon S3	Amazon Simple Storage Service
Amazon SQS	Amazon Storage Queue Service
AWS	Amazon Web Services
CE	Cross Entropy
CI/CD	Continuous Integration and Continuous Delivery/Deployment
DAC	Descript Audio Codec
EMA	Exponential Moving Average
FaaS	Function as a Service
FIFO	First In First Out
GPU	Graphics Processing Unit
LM	Language Model
NVMe	Nonvolatile Memory Express
OTP	One-Time Password
SSD	Solid-State Drive
UAT	User Acceptance Testing
UI	User Interface
vCPU	Virtual Central Processing Unit
VQ-VAE-2	Vector Quantized Variational AutoEncoder
VRAM	Video Random Access Memory

1. Introduction

Acoustic music, which is often characterized by its use of instruments without electronic amplification, tends to capture the purity of sound through instruments like violins, pianos, guitars, and vocals, resulting in a soothing and calming atmosphere that can lead to relaxation and improved sleeping quality.

In today's music landscape, the availability of acoustic renditions of songs is rather limited. Listeners either rely on original artists to release acoustic versions of their songs or listen to independent artists' covers, both of which can be unpredictable and limited in availability. Moreover, for music creators, creating acoustic adaptations often requires a proper recording studio for experimenting with different musical instruments. Independent artists who don't have access to resources and facilities often encounter obstacles in trying different instrumentations to transform their compositions into different styles.

With the emergence of generative AI and the success in image style transfer with generative image models like DALL·E and Midjourney, this project is motivated to create a generative AI model specifically for creating acoustic adaptations of musical compositions to bridge the gap between human-created musical compositions and AI-generated music. Thus, expanding the availability of acoustic renditions of songs and contributing to the field of AI-assisted music composition as well as style transfer.

2. Objectives

The main objective of this project is to create a generative AI model specializing in producing acoustic interpretations of musical compositions. Firstly, this project aims to harness the potential of generative AI models to generate quality acoustic renditions of musical compositions in a reasonable amount of time, given the original versions of songs and style requirements during the input stage. Secondly, this project aims to provide a web interface to accompany the model for users to generate musical compositions easily and quickly.

3. Background

In the field of generative AI, there are many existing works done with regard to audio style transfer. However, few of them are capable of handling both vocals and musical instruments with the exception of Jukebox, and models specializing in acoustic renditions are almost nonexistent. The sections below will briefly introduce related generative audio models and the models used or experimented in this project.

3.1 Related Works

3.1.1 Jukebox

Jukebox is an open-source generative AI model developed by OpenAI. It is a neural net that is specifically trained for music generation and rudimentary singing in various artist styles [1]. Jukebox was trained on a huge dataset of 1.2 million songs alongside the corresponding lyrics and metadata [1]. Therefore, it is capable of generating lyrics and music in different genres while mimicking a particular artist, and song completion given the first part of an audio sequence [1].

Jukebox leverages Vector Quantized Variational AutoEncoder (VQ-VAE-2) as its autoencoder model to compress 44 kHz 32-bit raw audio waveform by up to 128 times into a discrete representation of the waveform [1]. Then, the audio can be reconstructed by decoding it back to the raw audio space while conditioned on textual information supplied during the input phase [1].

In 2020, when the model was first proposed, it represented a huge leap in AI music generation with the capability to specify artists, genres, and lyrics. However, it has some noticeable flaws. Firstly, noise similar to that found in radio broadcasts is introduced during the downsampling and upsampling phase of the raw audio which leads to a subpar listening experience that does not live up to the current music streaming standards [1]. Additionally, due to the relatively slow autoregressive sampling process, it usually takes around 9 hours to generate 1 minute of audio sequence [1], which renders the model unusable in interactive applications like the one this project aims to create.

3.2 Adopted/Experimented Models

3.2.1 Audiocraft EnCodec

EnCodec is a high-fidelity, real-time audio codec in the AudioCraft family developed by Meta [2]. It utilizes neural networks to compress various kinds of audio and reconstruct the original audio signal with exceptional fidelity [3].

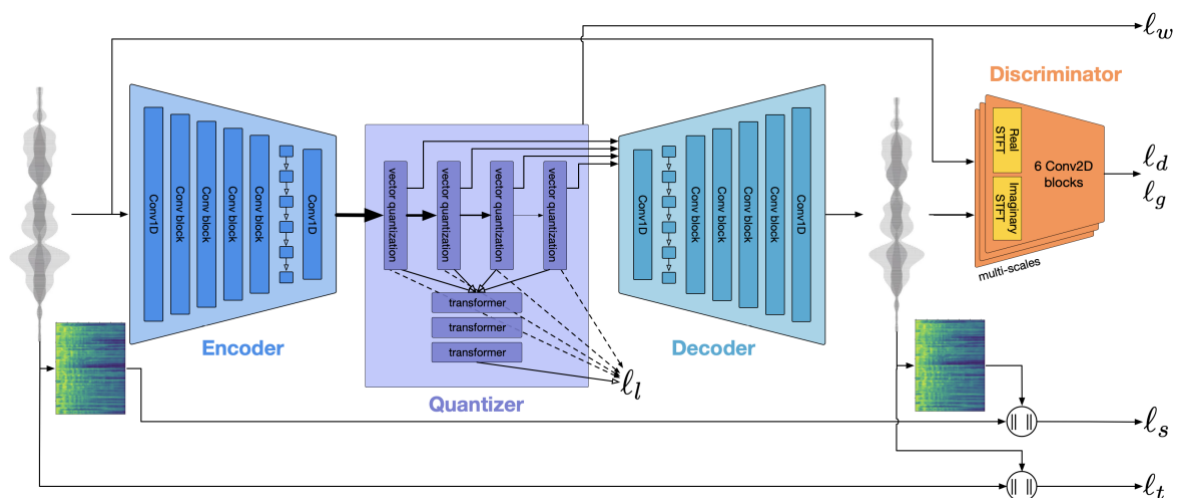


Figure 1 Architecture of EnCodec [2]

The architecture of EnCodec is a streaming encoder-decoder architecture with a quantized latent space [2]. It is trained with reconstruction (l_f and l_t) alongside adversarial losses (l_g for the generator and l_d for the discriminator) [2]. Additionally, the encoder is applied with the residual vector quantization commitment loss (l_w). Moreover, a small Transformer language model can be trained optionally for entropy coding over the quantized units with l_l , which further reduces bandwidth [2].

3.2.2 Audiocraft MusicGen

MusicGen is an open-source generative AI model in the AudioCraft family developed by Meta. It is a generative audio model trained with Meta-owned and specifically licensed music for music-generation tasks [4]. It is capable of generating music conditioned on melodic and textual features, allowing finer controls over the generated audio output [4].

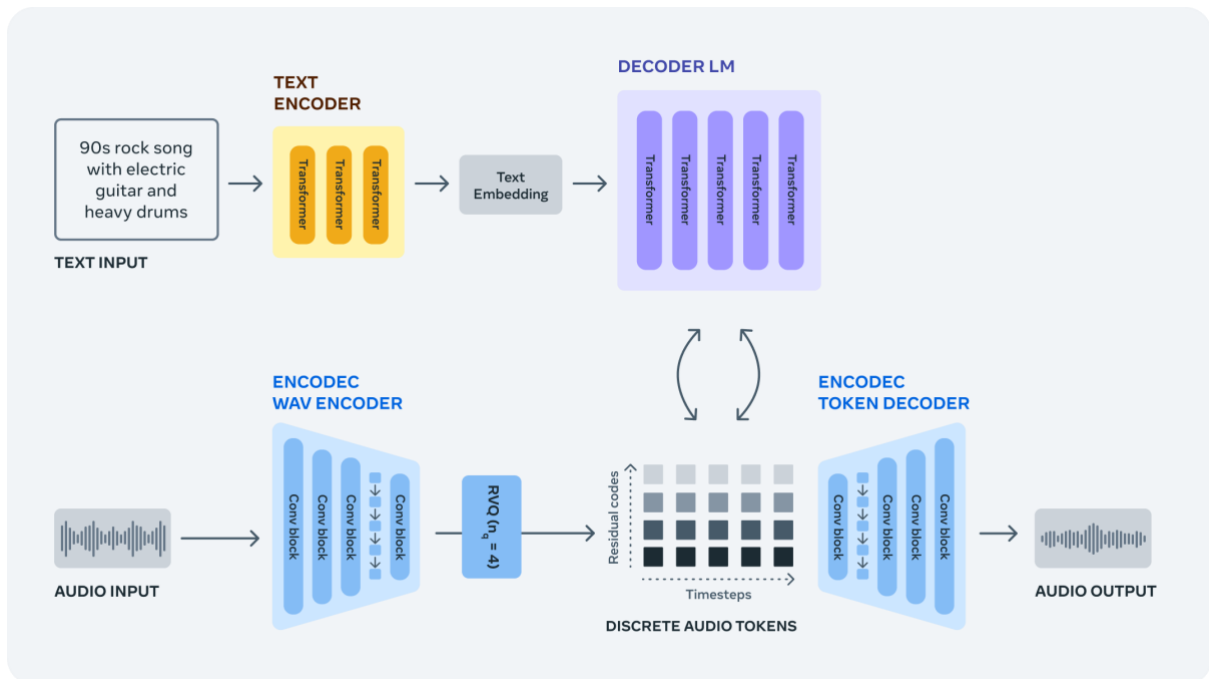


Figure 2 Architecture of MusicGen [3]

MusicGen contains one single autoregressive language model (LM) that processes over streams of tokens, which are compressed discrete music representations [3]. It introduces a simple approach that effectively models audio sequences, capturing the long-term dependencies in the audio simultaneously by utilizing the internal structure of the parallel streams of tokens [3]. As a result, it can generate high-quality audio with a single model and token interleaving pattern [3].

MusicGen utilizes the EnCodec neural audio codec to learn the discrete audio tokens from the raw waveform [3]. EnCodec transforms the audio signal into one or more parallel streams of discrete tokens [3]. Then, a single autoregressive language model is employed to iteratively model the audio tokens from EnCodec [3]. Afterward, the resulting tokens are input into the EnCodec decoder to map back to the audio space to generate the output waveform [3]. Lastly, various types of conditioning models, such as a pre-trained text encoder can be applied to condition the generation process, particularly for applications involving text-to-music transformations [3].

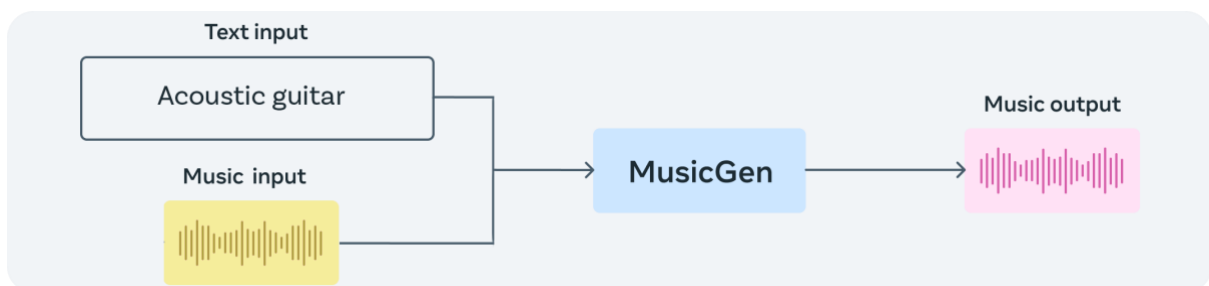


Figure 3 Generate music with MusicGen with text and melody conditioning

MusicGen can accept both waveform and textual input for conditioning the music output. However, MusicGen was not trained on music involving vocals. Thus, it is unable to handle vocals in songs. Instead of producing vocals alongside instrumentals, the model ignores the

vocals completely and generates instrumental music only. Moreover, the output audio degrades significantly if the output duration is longer than 30 seconds. Lastly, the pre-trained models of MusicGen can only output audio of up to 32kHz sampling rate because of the limitations posed by EnCodec.

3.2.3 Descript Audio Codec (DAC)

DAC is a high-fidelity general neural audio codec that was used as a replacement for EnCodec in part of this project [5]. The model is based on the VQ-GANs using the identical pattern as EnCodec [5]. Same as EnCodec, it also utilizes a fully convolutional encoder-decoder network that performs temporal downscaling with a given striding factor [5]. One of the benefits of using DAC includes a higher compression ratio. It is able to compress 44.1 kHz audio into discrete codes at an 8 kbps bitrate achieving around 90 times compression with fewer artifacts and minimal loss in quality [5]. Additionally, it fixes an important issue found in existing codecs which full bandwidth is not utilized because of codebook collapse by using more advanced codebook learning techniques [5].

4. Works Accomplished

4.1 AI Models

There are two approaches when it comes to creating a model for generating acoustic adaptations. The first approach is training MusicGen with DAC, and the second approach involves audio splitting before feeding into a pre-trained MusicGen model.

4.1.1 Approach 1 – MusicGen with DAC

4.1.1.1 Model Architecture

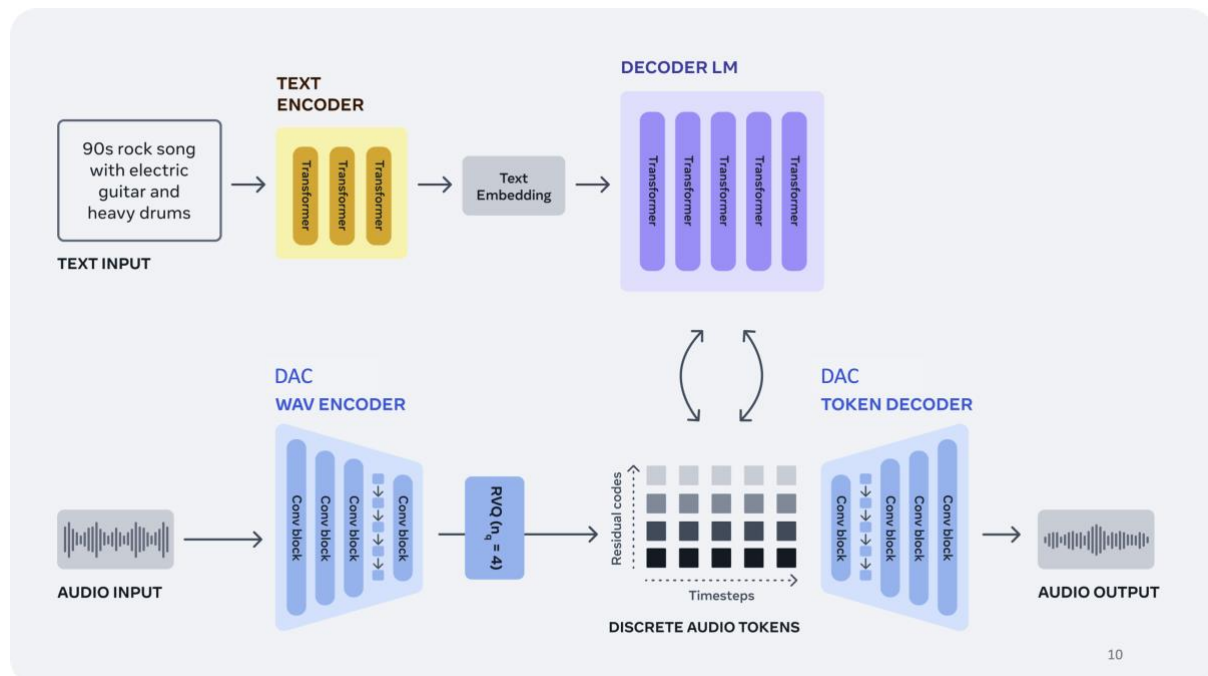


Figure 4 Approach 1 model architecture

The model architecture is the same as MusicGen’s architecture except that the neural audio codec is replaced by DAC instead of EnCodec. With DAC, audio can be compressed from 44.1 kHz sampling rate into discrete codes as low as 8 kbps bitrate while maintaining high fidelity and little artifacts [5].

4.1.1.2 Dataset

The dataset used consists of 103 English songs which are official acoustic versions of released songs across different genres and artists like Taylor Swift and Sam Smith (see Appendix 1 for the complete list). The songs were downloaded from YouTube Music at 44.1 kHz dual channels and their respective metadata are fetched through Spotify API. The metadata collected were then stored in JSON format and the metadata includes:

KEY	EXAMPLE VALUE
KEY	1
ARTIST	a-ha
SAMPLE_RATE	44100
FILE_EXTENSION	wav
KEYWORDS	acoustic
DURATION	184.33015873015873
BPM	103.883
GENRE	new romantic
TITLE	Take On Me - 2017 Acoustic
NAME	a-ha - Take On Me - 2017 Acoustic
INSTRUMENT	guitar

Table 1 Example metadata of Take On Me

Originally, line-synced lyrics, which means a timestamp is only available at the start of each sentence, were included in the JSON files. However, the songs were trimmed down to only 5 seconds each during the training phase which rendered line-synced lyrics useless. Due to the absence of a free word-synced lyrics provider, lyrics were not included in the JSON files as a result.

The songs together with their respective JSON files were split into training, validation, and testing sets randomly where 70% were placed in the training set, 20% were placed in the validation set, and 10% were placed in the testing set.

4.1.1.3 Model Training

Due to the high computation requirements of MusicGen, everything had to be downsized to avoid overloading a local NVIDIA RTX 4080, which only had 16GB of VRAM. The most notable one is to trim every single song in the dataset down to only 5 seconds. Apart from the adjustments in the dataset, there were also changes in the model hyperparameters and configurations. The tables below illustrate some of the hyperparameters and configurations adopted when training the model.

DATASET PARAMETER/CONFIGURATION	
CHANNELS	1
SAMPLE RATE	44100
BATCH SIZE	3
SEGMENT DURATION	5

Table 2 Dataset parameters and configurations

LANGUAGE MODEL PARAMETER/CONFIGURATION	
--	--

LM	transformer-lm
LM MODEL SCALE	small
TOP_K	250
TOP_P	0
CONDITIONERS	t5-base (text conditioning), chroma-stem (waveform conditioning)

Table 3 Language model parameters and configurations

OPTIMIZER PARAMETER/CONFIGURATION	
LOSS FUNCTION	cross entropy (CE)
OPTIMIZER	DADAM
EPOCHS	500
EMA	0.99 decay every 10 updates

Table 4 Optimizer parameters and configurations (Unchanged)

4.1.1.4 Training Result

The final model was made up of 321.77 million parameters. Figure 5 shows how the CE, which measures the performance of the model (the lower the better), and is defined as:

$$H(p, q) = -E_p[\log q]$$

where $E_p[\cdot]$ is the expected value operator to the distribution p [6], changes over time.

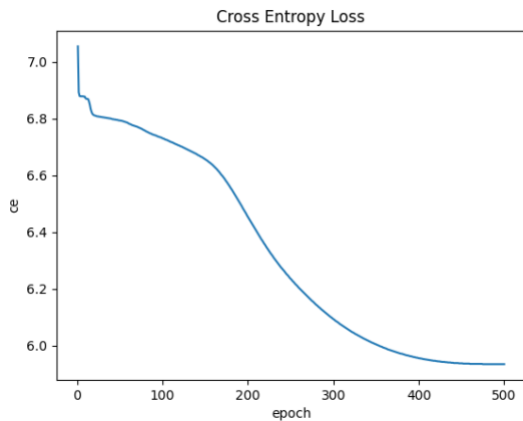


Figure 5 Cross-entropy loss of the model

As illustrated in Figure 5, the CE was reducing at a very slow rate after epoch 400 and the CE was at 5.935 at epoch 500. That means the distribution predicted by the model could not align with the actual distribution of the target data well. As a result, the model cannot predict accurately and meaningfully given an audio sequence.

Figure 6 shows how the perplexity, which measures the uncertainty in the value of a sample from a discrete probability distribution (the lower the better), and is defined as:

$$PP(p) := 2^{H(p)} = 2^{-\sum_x p(x) \log_2 p(x)} = \prod_x p(x)^{-p(x)}$$

where x ranges over the events, and $H(p)$ is the entropy (in bits) of the distribution [7], changes over time.

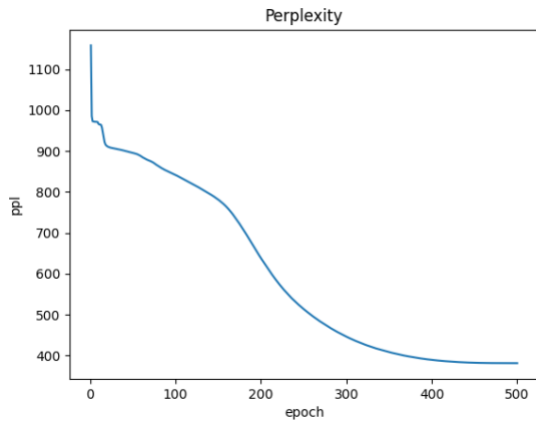


Figure 6 Perplexity of the model

As illustrated in Figure 6, the perplexity was reducing at a very slow rate after epoch 400 and it was at 381.606 at epoch 500. That means the model could not identify and extract the structure and patterns within an audio signal [7]. As a result, it is less likely to predict the next frame in an audio sequence correctly.

As suggested by the metrics, the model performed very poorly. Although the output of the model can follow the melody of a given audio sequence, the output lacks clarity and enunciation. It also introduces a mumbling human voice instead of maintaining the original lyrics which further degrades the overall auditory experience. As a result, the output is a mix of muddled vocals and subpar musical elements which renders the model unusable for producing acoustic adaptations based on a source material.

4.1.1.5 Limitations

The unsatisfactory output quality could be attributed to a relatively small training dataset and using a small-scale LM for training as the pre-trained MusicGen model uses a medium-scale LM for its normal melody-conditioned (1.5 billion) model whereas a large-scale LM is used for its large melody-conditioned (3.3 billion) model. Moreover, as suggested by the implementation of Jukebox, the presence of lyrics could be crucial for generating songs. The lack of lyrics in the metadata file could lead to muddling vocals in the output audio.

Apart from the significant limitations of the output quality of the model, the model cannot generate audio sequences longer than 5 seconds and it is limited to generating mono audio sequences. Moreover, the training of the model was very compute-intensive which required approximately 10 days with over 95% utilization rate on the GPU to complete the training on a local NVIDIA RTX 4080. This makes the model training and evaluation lifecycle so long that hyperparameter tuning or configuration changes cannot be done and evaluated promptly.

4.1.2 Approach 2 – Audio Splitting

4.1.2.1 Model Architecture

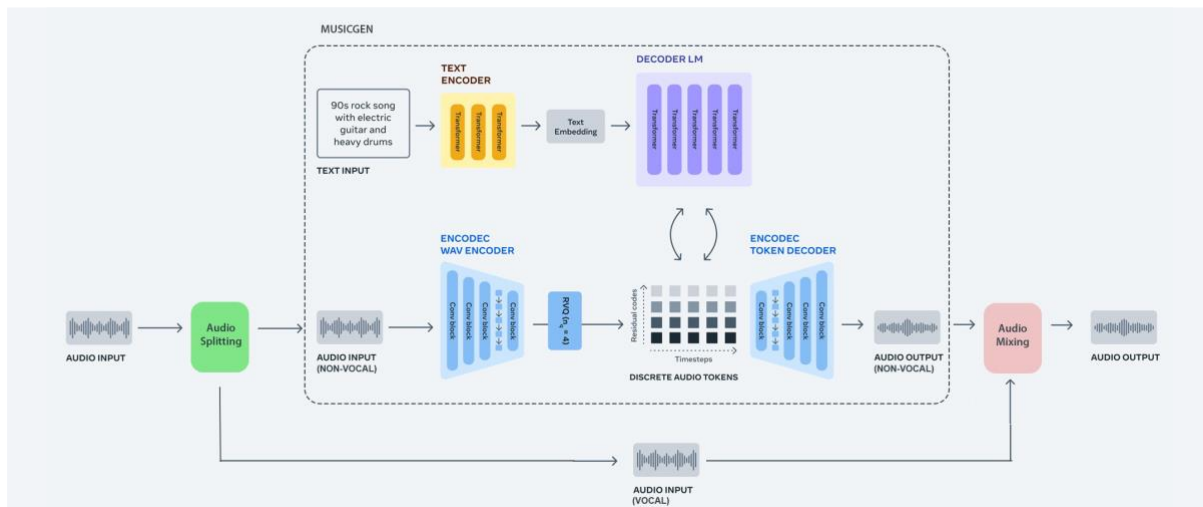


Figure 7 Model architecture of approach 2

The idea of this approach is to create a model with minimal changes to MusicGen for quick testing and evaluation. Other than MusicGen, this approach introduces another model, that is lalal.ai for audio splitting. This pre-trained model helps to split an audio input into vocals and non-vocals, resulting in two separate audio sequences leaving only the non-vocal track to be processed by MusicGen. The MusicGen model used in this approach is a pre-trained “musicgen-stereo-melody-large” which is a 3.3 billion model capable of melody conditioning and stereo audio generation.

The flow of the model starts with a piece of song that contains both instrumentals and vocals. The song is then passed to the audio-splitting model to split the song into vocals and non-vocals. The non-vocals are extracted and passed to the pre-trained MusicGen model. As the audio input passed to MusicGen only consists of instrumentals, the audio output also consists of instrumentals only and does not contain any vocals. After MusicGen has generated the non-vocal part of the audio based on the melody of the input audio file and the style specified in the text prompt, the output is ready to be mixed with the vocal part of the original song, which was separated earlier by lalal.ai. The final audio output is a mix of the original vocals with melodies played by acoustic instruments.

4.1.2.2 Results

The generated audio showcases great musicality with clear and well-rendered vocals. The model can generate music based on the melody of a given audio sequence, and the instruments and style specified in the text prompt in a harmonious arrangement. However, despite these strengths, the generated audio suffers from a flaw that is significantly noticeable in faster and more energetic songs, that is the lack of coherence with regard to beats and tempo. This results in a sense of inconsistency in the rhythm as the vocals are not modified in the generation process whereas the melody and instruments change to a more acoustic setting by removing the punchiness of the song and introducing softer transitions in the rhythm. This incoherence negatively impacts the auditory experience for acoustic adaptations for fast songs.

4.1.2.3 Limitations

Apart from the limitations when generating acoustic renditions for faster and more energetic songs, this approach also suffers from the limitations carried from the audio splitting model and MusicGen. Firstly, there will be a slight drop in quality after splitting the audio sequence into vocals and non-vocals, which will be propagated through the MusicGen stage. Secondly, as limited by the capabilities of EnCodec, the generated audio can only support a sampling rate of up to 32 kHz. Finally, as inherited from the limitation of MusicGen, the quality degrades significantly if the duration of the output audio sequence is longer than 30 seconds.

4.1.3 Models Summary

In conclusion, approach 1, which utilizes MusicGen with DAC, encountered obstacles in achieving satisfactory audio output quality as the model struggled with producing clear vocals and musical instruments. The high computation costs when training the model also limited the scale of the LM and blocked further experimentations with the current hardware setup. On the other hand, approach 2, which utilizes lala.ai for audio splitting alongside MusicGen, exhibits satisfactory results in generating music with clear vocals and musical instruments. However, it suffers from incoherence with tempo and beats, particularly in faster songs. While both approaches suffer from certain flaws and limitations, approach 2 demonstrated better musicality and clearer vocals despite challenges in rhythm consistency and coherence. Therefore, approach 2 was chosen to be implemented in the web app.

4.2 Web App

4.2.1 Frontend

ReactJS is used as the framework for the frontend of the app, and the UI is coded and stored as components for reusability. The frontend is hosted on AWS Amplify and connected to the backend database, which is Amazon DynamoDB, via GraphQL.

4.2.2 Backend

All of the backend services are running on AWS for easy deployments and management. The diagram below illustrates the architectural design of this web app on AWS.

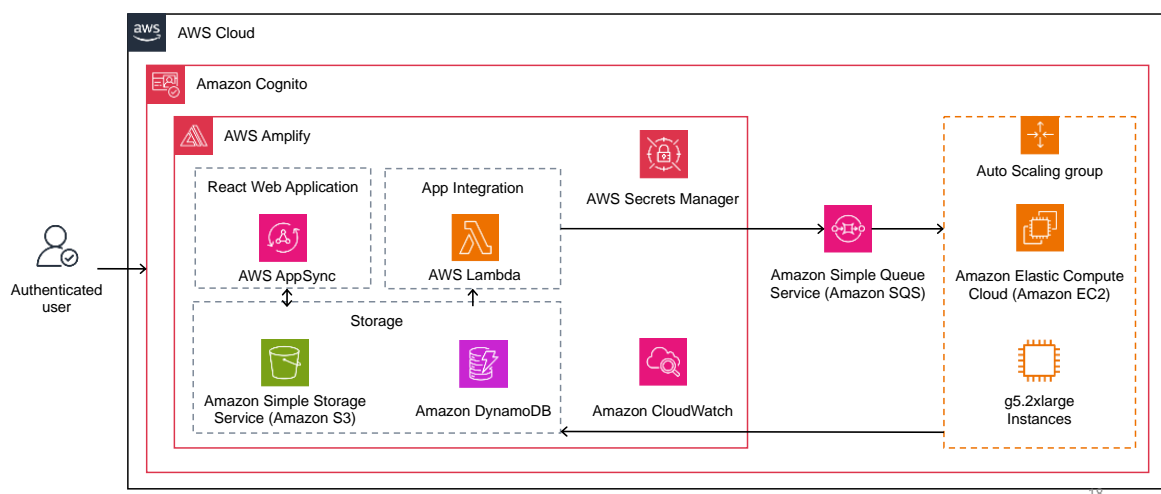


Figure 8 AWS architecture

4.2.3 List of AWS resources

4.2.3.1 Amazon Cognito

It is an identity and access management service [8] that is used to authenticate users to the web app and authorize users to access his/her audio files.

4.2.3.2 AWS Amplify

It is a platform for hosting full-stack web and mobile apps [9] that are used to manage the web app easily, set up CI/CD, and host the web app. Features like storage and authentication can also be deployed and managed easily on Amplify Studio.

4.2.3.3 AWS AppSync

It is an API service that is used to query, create, update, and delete items stored in the Music table inside the Amazon DynamoDB via GraphQL and Pub/Sub APIs [10].

4.2.3.4 Amazon S3

It is a cloud object storage service [11] that is used to store the original audio files and the generated audio files from Autoustic. The bucket and objects are not accessible publicly, that is the bucket only allows the owner of the uploaded audio files to view and download the corresponding generated file.

4.2.3.5 Amazon DynamoDB

It is a NoSQL, serverless, and fully managed database [12] that is used to store the data related to music generation. For example, the text prompts used to condition the output audio, the S3 IDs and URLs of the audio files stored on Amazon S3, the username that initiated the generation, the status of the generation, etc. Below is the GraphQL schema:

```
type Music @model @auth(rules: [{allow: public}]) {  
  id: ID!  
  prompt: String!  
  s3uid: String!  
  s3durl: String  
  username: String!  
  vocal: Boolean!  
  lalad: String  
  status: String!  
  name: String!  
}
```

where “!” represents mandatory fields.

4.2.3.6 AWS Lambda

It is a serverless compute service [13] that is used to run a custom JavaScript for preprocessing before being handled by Autoustic. This Lambda function is triggered whenever there is an update on the Music table on Amazon DynamoDB. If a newly created item triggers it, the function will start by downloading the original audio file from S3, then it will upload the file to lalal.ai to split the audio file into vocals and non-vocals. Once the splitting has been

completed, it will send the download URL, text prompt, item (DynamoDB) ID, and original audio file's (S3) ID to the queue in Amazon SQS for further processing by Autoustic.

4.2.3.7 AWS Secrets Manager

It is a service for managing the lifecycle of secrets centrally [14]. It currently stores the API key for lambda to communicate with lalal.ai APIs.

4.2.3.8 Amazon CloudWatch

It is a monitoring service for logging and debugging [15]. It collects logs from AWS Lambda and Amazon EC2 and stores them in a workplace which allows logs can be easily viewed and traced when unexpected errors occur.

4.2.3.9 Amazon SQS

It is a fully managed message queuing that allows the decoupling and scaling of microservices [16]. Due to the huge differences in processing time between audio splitting and generation, it may be better to decouple these two processes to cope with sudden changes in demand. Amazon SQS decouples the audio splitting process, which AWS Lambda handles, and the audio generation process, which is handled by Amazon EC2, by placing a queue where AWS Lambda can send messages to the queue, and Amazon EC2 instances can consume messages from the queue. As AWS Lambda is Function as a Service (FaaS) which scales automatically according to the demand and Amazon EC2 instances are put under an auto-scaling group, they can also scale out according to the demand. Note that this is a first-in-first-out (FIFO) queue in which order is maintained and message duplication is not allowed to maintain the priority between requests.

4.2.3.10 Amazon EC2

It is a compute service [17] that is mainly used for audio generation. Multiple g5.2xlarge EC2 instances are placed into an auto scaling group, each of them is equipped with one NVIDIA A10G Tensor Core GPU that comes with 24 GiB of GPU memory, 8 vCPUs, 32 GiB of memory, and 450 GB of local NVMe SSD storage, resulting in high performance for stereo audio generation [18].

4.2.4 User Flow

1. For new user registration, he/she creates an account with a unique username, email, and password, then he/she will receive a one-time password (OTP) for email verification. Once the email has been verified, the registration is completed, and the user is added to the user pool on Amazon Cognito. Then, the user will be directed to his/her home page. As for existing users, he/she simply log in with his/her usernames and passwords, and Amazon Cognito will verify the login credentials against the user pool. If the user passes through the authentication, he/she will be directed to his/her home page.

Sign In **Create Account**

Username

Password

[Forgot your password?](#)

Figure 9 Sign in page

Sign In **Create Account**

Username

Password

Confirm Password

Email

Create Account


Figure 10 Create account page

2. On the home page, there is a form that allows the user to initiate a music generation task and a “Your music” table that allows the user to view and listen to previously generated music.

Name

Text Prompt

Music File



Drop files here or

Vocal Extraction

Your music



Name	Uploaded music	Text prompt	Generated music	Status
Idol	 0:00 -1:45	guitar and piano	 0:00 -1:30	Completed

Figure 11 User home page

To generate an acoustic version of a piece of music, the user simply needs to fill in a friendly name for identification like the name of the music, a text prompt, for example, acoustic guitar, upload the original music, and turn on vocal extraction if the music contains human voice and turn it off if it is solely instrumentals. This switch determines whether the uploaded music needs to be preprocessed by lalal.ai.

- Once the user has submitted the form and the audio file is uploaded to Amazon S3, a new item is created with form data in DynamoDB, and this triggers the Lambda function. The function starts by updating the status field on DynamoDB, which will then be reflected on the corresponding row under the “Your music” table. Then, it will download the original music from Amazon S3 and upload it to lalal.ai if vocal extraction is enabled.
- The Lambda function checks on the progress of audio splitting and updates the status if necessary. Once the audio splitting is completed, it will send the data to Amazon SQS, which details are mentioned in section 4.2.3.6.
- When the message has arrived at the front of the queue, an Amazon EC2 instance will consume the message and retrieve the data. The function running on EC2 will download the audio files, which include a file that only contains the vocals and another file that only contains the non-vocals, from lalal.ai if vocal extraction is enabled. Then, it will change the sampling rate of both files to 32 kHz which is the input format of MusicGen. Then, the non-vocal file and the text prompt will be fed to the model for inferencing. Finally, it will combine the generated audio with the vocals and upload the mix to S3 as well as update the status and the S3 download URL of the corresponding item. Once everything has been completed, it will delete the message in the queue to avoid the message being re-consumed by other EC2 instances.
- The “Your music” table reflects the latest changes on Amazon DynamoDB and generates signed S3 URLs for the user to download and listen to both original and generated music.

5. Future Works

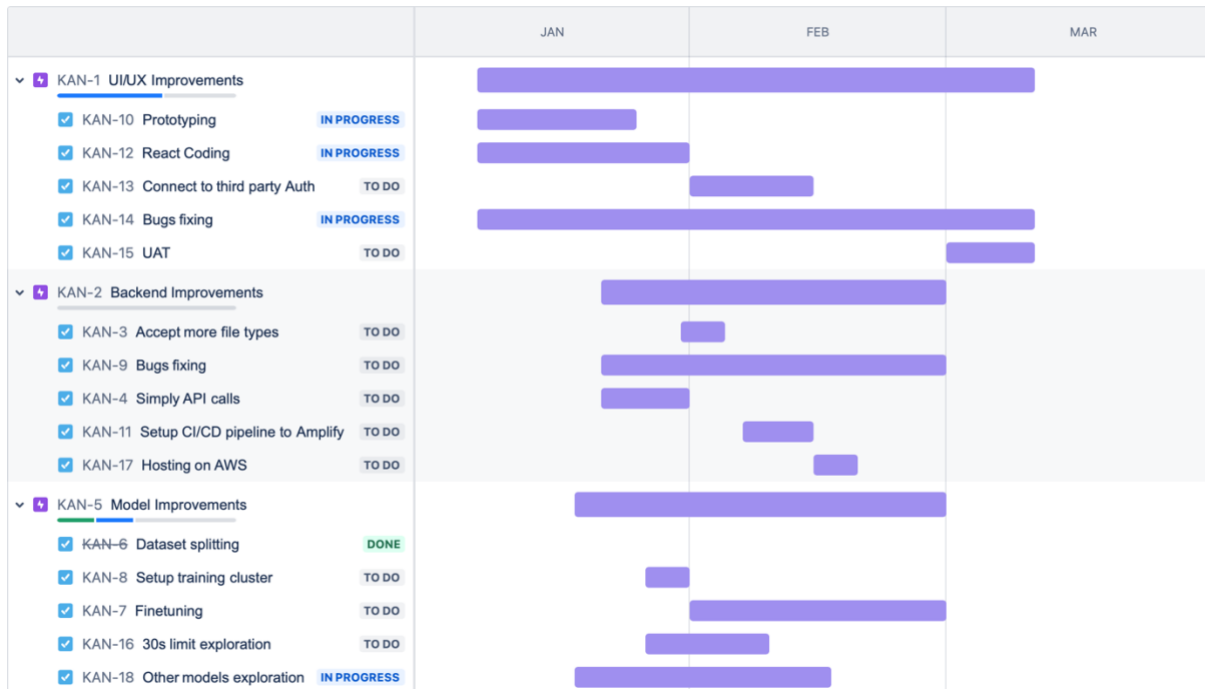


Figure 12 Project timeline

To improve the web app, both the frontend and backend will be revamped and improved in the coming months to provide a more seamless user experience for users to interact with the model. For the frontend of the web app, as the current UI is quite simple and only provides some of the basic features to the users, the UI will be completely revamped. Prototyping will be done by 25 January and coding of the UI components will be done by the end of January. Moreover, to provide a more seamless experience to the users, third-party identity providers like Google and Facebook will be added by mid-February to provide more options for users to authenticate and register with the app. Finally, user acceptance tests (UAT) need to be carried out before deploying and submitting the web app to avoid major flaws and logical errors from happening in the app.

Secondly, for the backend of the web app, there may be some architectural problems that are incurring a huge amount of costs on AWS. Further investigation needs to be carried out and redesign of the client-to-server and server-to-server communications might be needed to find a solution that is both cost-effective and incurs minimal impact on the user experience. Moreover, to speed up the development and testing of the web app, a CI/CD pipeline will be created by early February and the web app is expected to start hosting on AWS Amplify in mid-February.

Finally, regarding the music generation model, the pre-trained model from MusicGen will be finetuned to better prepare the model for acoustic rendition tasks by feeding more acoustic music data into the model. The training dataset is readily available whereas the training infrastructure will be set up by the end of January on AWS and it is expected to continue finetuning and evaluating the model until the end of February. In the meantime, investigations on methods such as sliding windows to overcome the 30-second generation limit inherited from MusicGen will start immediately as this determines the capability of the model. Finally,

there are a few existing models which are capable of audio style transfer. Comparison and evaluation of these models will be done in parallel with MusicGen finetuning tasks to determine which model is best suited for acoustic rendition purposes.

Appendix 1

Song Name	Artist
Youngblood - Acoustic	5 Seconds of Summer
Take On Me - 2017 Acoustic	a-ha
Wild Things - Acoustic Version	Alessia Cara
No One - Acoustic	Alicia Keys
Lonely Together - Acoustic	Avicii
Meant to Be - Acoustic	Bebe Rexha
Eastside (with Halsey & Khalid) - Acoustic	benny blanco
You (with Marshmello & Vance Joy) - Acoustic	benny blanco
Grenade - Acoustic	Bruno Mars
Dancing On My Own - Acoustic	Calum Scott
Attention - Acoustic	Charlie Puth
One Call Away - Acoustic	Charlie Puth
I'll Be Waiting - Acoustic	Cian Ducrot
Solo (feat. Demi Lovato) - Acoustic	Clean Bandit
Symphony (feat. Zara Larsson) - Acoustic Version	Clean Bandit
My Universe - Acoustic Version	Coldplay
Yellow - Live from Spotify London	Coldplay
Bad Day - Acoustic	Daniel Powter
Sorry Not Sorry - Acoustic	Demi Lovato
Be the One - Acoustic	Dua Lipa
Blow Your Mind (Mwah) - Acoustic	Dua Lipa
IDGAF - Acoustic	Dua Lipa
New Rules - Acoustic	Dua Lipa
Castle on the Hill - Acoustic	Ed Sheeran
Happier - Acoustic	Ed Sheeran
I Don't Care - Acoustic	Ed Sheeran
Lego House - Acoustic	Ed Sheeran
Perfect - Acoustic	Ed Sheeran
Shape of You - Acoustic	Ed Sheeran
South of the Border (feat. Camila Cabello) - Acoustic	Ed Sheeran
Cold Heart - Acoustic	Elton John
Most Girls - Acoustic	Hailee Steinfeld
Starving - Acoustic	Hailee Steinfeld
Quite Miss Home - Acoustic	James Arthur
Train Wreck - Acoustic	James Arthur
Please Keep Loving Me - Acoustic	James TW
Here's Your Perfect - Acoustic	Jamie Miller
Ridin' Solo - Acoustic	Jason Derulo
Price Tag - Acoustic Version	Jessie J
Fast Car - Acoustic	Jonas Blue

Mama - Acoustic	Jonas Blue
Perfect Strangers - Acoustic	Jonas Blue
Rise - Acoustic	Jonas Blue
No Air (feat. Chris Brown) - Acoustic Version	Jordin Sparks
Passport Home - Piano Acoustic	JP Cooper
September Song - Piano Acoustic	JP Cooper
Boyfriend - Acoustic Version	Justin Bieber
Intentions - Acoustic	Justin Bieber
Lonely (with benny blanco) - Acoustic	Justin Bieber
What Do You Mean? - Acoustic	Justin Bieber
The One That Got Away - Acoustic	Katy Perry
Thinking Of You - Acoustic Version	Katy Perry
Can't Get You out of My Head - Live from Spotify, London	Kylie Minogue
Before You Go - Guitar Acoustic	Lewis Capaldi
Strip That Down - Acoustic	Liam Payne
No More Sad Songs - Acoustic Version	Little Mix
Shout Out to My Ex - Acoustic	Little Mix
Harder To Breathe - Acoustic	Maroon 5
Never Gonna Leave This Bed - Acoustic Version	Maroon 5
She Will Be Loved - Acoustic	Maroon 5
Sunday Morning - Acoustic	Maroon 5
Won't Go Home Without You - Acoustic Version	Maroon 5
FRIENDS - Acoustic	Marshmello
Scared to Be Lonely - Acoustic Version	Martin Garrix
Beautiful Scars - Acoustic	Maximillian
Colour - Acoustic	MNEK
So Sick - Acoustic	Ne-Yo
Slow Hands - Acoustic	Niall Horan
Too Much To Ask - Acoustic	Niall Horan
Find You - Acoustic	Nick Jonas
Remember I Told You - Acoustic	Nick Jonas
Sunroof - Acoustic	Nicky Youre
La La Lost You - Acoustic Version	NIKI
Night Changes - Live Acoustic Session	One Direction
One Thing - Acoustic	One Direction
Perfect - Stripped	One Direction
Steal My Girl - Live Acoustic Session	One Direction
If I Lose Myself - Acoustic	OneRepublic
Mariposa - Acoustic	Peach Tree Rascals
Your Song - Acoustic	Rita Ora
These Days (feat. Jess Glynne, Macklemore & Dan Caplen) - Acoustic	Rudimental
This City - Acoustic	Sam Fischer

Dancing With A Stranger (With Normani) - Acoustic	Sam Smith
How Do You Sleep? - Acoustic	Sam Smith
Latch - Acoustic	Sam Smith
Too Good At Goodbyes - Acoustic	Sam Smith
Lost In Japan - Recorded at Spotify Studios NYC	Shawn Mendes
Mercy - Acoustic	Shawn Mendes
There's Nothing Holdin' Me Back - Acoustic	Shawn Mendes
Wonder - Acoustic	Shawn Mendes
Youth (feat. Khalid) - Acoustic	Shawn Mendes
Electricity - Acoustic	Silk City
Love Is Gone - Acoustic	SLANDER
Back To December - Acoustic	Taylor Swift
Delicate - Recorded at The Tracking Room Nashville	Taylor Swift
The Man Who Can't Be Moved - Acoustic	The Script
My My My! - Acoustic	Troye Sivan
YOUTH - Acoustic	Troye Sivan
A Thousand Miles - Acoustic	Vanessa Carlton
Shut Up and Dance - Live Acoustic - 2015	WALK THE MOON
Fallin,Äô (Adrenaline) - Acoustic	Why Don't We
Beep Me - Acoustic	Will Heard
Stay - Acoustic	Zedd

References

- [1] P. Dhariwal, H. Jun, C. M. Payne, J. W. Kim, A. Radford and I. Sutskever, "Jukebox," OpenAI, 30 April 2020. [Online]. Available: <https://openai.com/research/jukebox>. [Accessed 21 January 2024].
- [2] A. Défossez, J. Copet, G. Synnaeve and Y. Adi, "High Fidelity Neural Audio Compression," 2022.
- [3] Meta, "AudioCraft: generating high-quality audio and music from text," Meta, 2023. [Online]. Available: <https://ai.meta.com/resources/models-and-libraries/audiocraft/>. [Accessed 21 January 2024].
- [4] J. Copet, F. Kreuk, I. Gat, T. Remez, D. Kant, G. Synnaeve, Y. Adi and A. Défossez, "Simple and Controllable Music Generation," 2023.
- [5] R. Kumar, P. Seetharaman, A. Luebs, I. Kumar and K. Kumar, "High-Fidelity Audio Compression with Improved RVQGAN," 2023.
- [6] Wikipedia, "Cross-entropy," Wikipedia, 8 December 2023. [Online]. Available: <https://en.wikipedia.org/wiki/Cross-entropy>. [Accessed 20 January 2024].
- [7] Wikipedia, "Perplexity," Wikipedia, 15 January 2024. [Online]. Available: <https://en.wikipedia.org/wiki/Perplexity>. [Accessed 20 January 2024].
- [8] AWS, "Amazon Cognito," AWS, 2024. [Online]. Available: <https://aws.amazon.com/cognito/>. [Accessed 19 January 2024].
- [9] AWS, "AWS Amplify," AWS, 2024. [Online]. Available: <https://aws.amazon.com/amplify/>. [Accessed 19 January 2024].
- [10] AWS, "AWS AppSync," AWS, 2024. [Online]. Available: <https://aws.amazon.com/appsync/>. [Accessed 19 January 2024].
- [11] AWS, "Amazon S3," AWS, 2024. [Online]. Available: <https://aws.amazon.com/s3/>. [Accessed 19 January 2024].
- [12] AWS, "Amazon DynamoDB," AWS, 2024. [Online]. Available: <https://aws.amazon.com/dynamodb/>. [Accessed 19 January 2024].
- [13] AWS, "AWS Lambda," AWS, 2024. [Online]. Available: <https://aws.amazon.com/lambda/>. [Accessed 19 January 2024].
- [14] AWS, "AWS Secrets Manager," AWS, 2024. [Online]. Available: <https://aws.amazon.com/secrets-manager/>. [Accessed 20 January 2024].
- [15] AWS, "Amazon CloudWatch," AWS, 2024. [Online]. Available: <https://aws.amazon.com/cloudwatch/>. [Accessed 19 January 2024].
- [16] AWS, "Amazon Simple Queue Service," AWS, 2024. [Online]. Available: <https://aws.amazon.com/sqs/>. [Accessed 19 January 2024].
- [17] AWS, "Amazon EC2," AWS, 2024. [Online]. Available: <https://aws.amazon.com/ec2/>. [Accessed 19 January 2024].
- [18] AWS, "Amazon EC2 G5 Instances," AWS, 2024. [Online]. Available: <https://aws.amazon.com/ec2/instance-types/g5/>. [Accessed 19 January 2024].