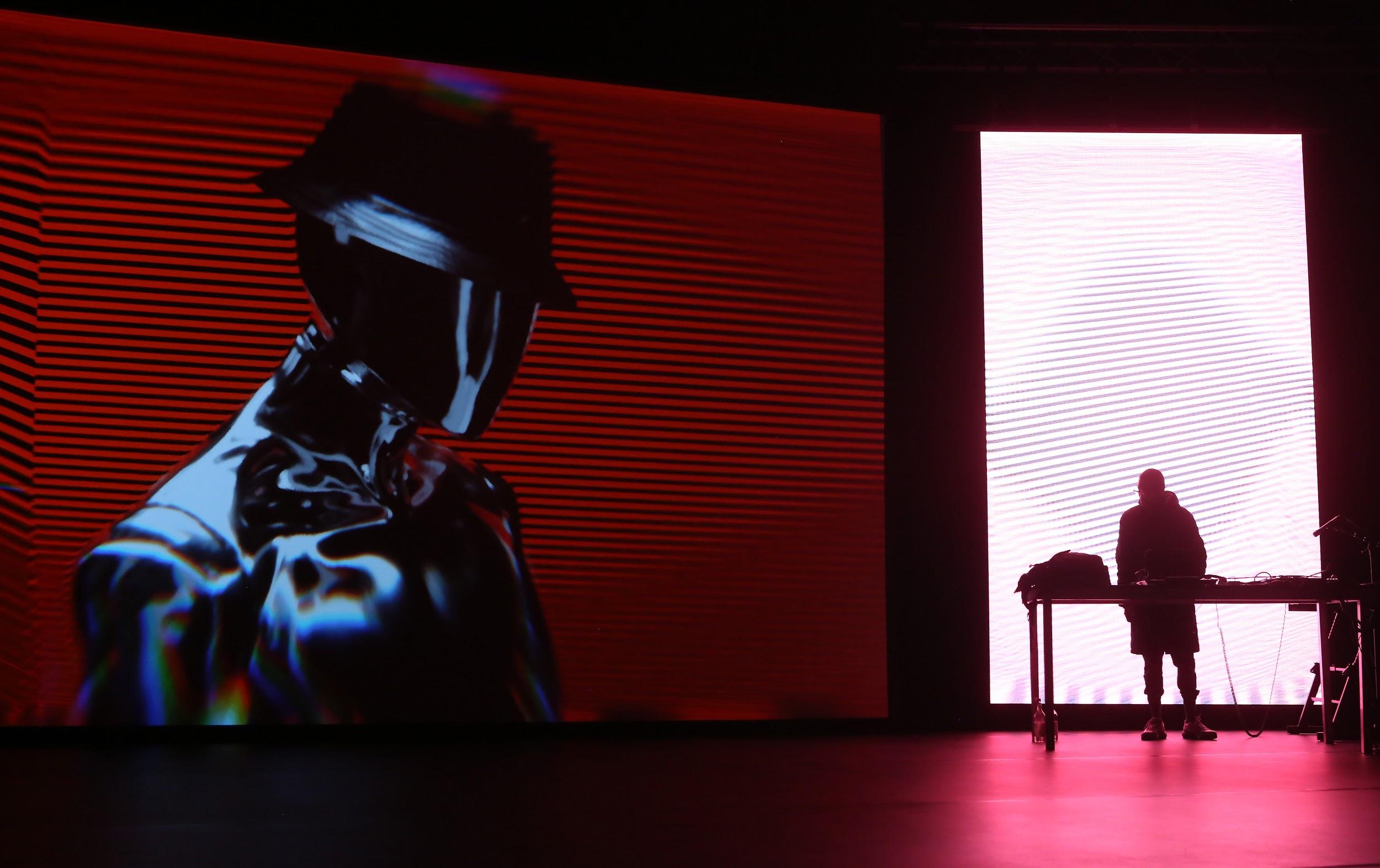
# Final Project Plan – gen m



Adam Berry, transmediale, CC BY NC-SA 4.0

**PROBLEM STATEMENT**

The purpose of this project is to generate new electronic music using a variety of state-of-the-art machine learning methods.

**MOTIVATION**

It may seem counterintuitive to utilize machine learning for creative endeavors, however recent innovations have opened new pathways to music creation. Music theory is deeply intertwined with mathematics. Dating as far back as the sixth century B.C., scientific minds such as Pythagoras began to study the theory of harmonics and their relationship to numerical intervals. As Western Music Theory evolved, maths continued to play a pivotal role in music, albeit a subtle one. Applying machine learning for music generation tests our knowledge and understanding of what constitutes music and allows us to stretch our musical imagination by exploring composition through a computational lens.

As artists, musicians seek inspiration from all around them. AI-based music generation, once perfected, can be a conduit for novel musical sounds, structures and melodies. We hope this project provides for us an outlet to expand our artistic and scientific horizons, and helps foster musical creativity not only for us but for those who wish to build upon our work.

We have particular interest in Electronic music generation, as it is a genre often passed on in AI music generation in favor of more structured forms, such as Classical and Rock music. Additionally, Electronic music is defined by its sound being produced by circuit-based machines, as opposed to traditional instruments. In that vein, we believe that Electronic music is uniquely appropriate to machine learning-based music generation.

**What is Music?**

To understand machine learning-based music generation, it is important to understand how we actually experience “music”. Sound at its essence is a sensation that the brain perceives when our eardrums vibrate. When those vibrations occur at consistent frequencies over time, we recognize them not as just noise but as musical tones or notes. Over the last millennium, musicians have codified specific frequencies as pitches, represented today by letters of the Roman alphabet. Each note represents a specific frequency. For example, “middle-C” is approximately 256 Hz, and doubling each frequency produces the same note one octave higher. By understanding the inherently numeric foundations of music, we can begin to formulate music composition not only as creative but as a computational endeavor as well.

A close up of a piano

Description automatically generatedA close up of a logo

Description automatically generated

https://www.slideshare.net/mikhailvink/alexey-fridman-piano-design-and-sound-generation

https://casparmenkman.typepad.com/caspar-menkman/2011/08/soundwaves.html

**TEAM ROLES**

This is how the project will be divided, part-wise

Data exploration and subjective model evaluation will be done by Zinoo Park

Experimentation and Application of VAEs will be carried out by H

GANs will be performed by M

However, it is important to note and therefore we stress that no individual part will be solely executed by that said member. We realize that our project is a collaborative process as a whole that may require

adjustments and fixes that propagate backward as well as forwards.

**DATA GATHERING & PREPROCESSING**

In order to perform calculations on music data, we need to process our music sources in a format that our algorithms understand. This generally involves representing the music numerically or symbolically by using some conversion process. Music files in MIDI format already represent music numerically and are widely available online. However, we will likely have to supplement MIDI data with other music sources, and we will need to process those files accordingly.

**MIDI Files**

**MelSpectrograms**

**Raw Audio Waveform**

**METHODOLOGY**

To use machine learning to generate music, we will rely on Artificial Neural Network structures. Neural Networks aim to replicate the way the human brain works: we pass data through a set of algorithms, and in that process attempt to learn patterns about the data. Neural Networks have many successful applications in Machine Learning and AI, and have recently been used successfully for new music generation.

The two primary forms of Neural Networks we plan to explore are Vector-Quantized Variational AutoEncoders (VQ-VAE), traditional and quantum Generative Adversarial Networks (GAN). Both methods have demonstrated success in generating coherent, multi-track music in various contexts. GANs provide a framework for generating music symbolically - i.e. representing various timesteps instruments as separate output layers. VQ-VAE takes a more holistic approach by accepting and outputting full pieces of raw audio. Through our study, we will determine which model is most effective for generating tracks of electronic music.

**VQ-VAE**

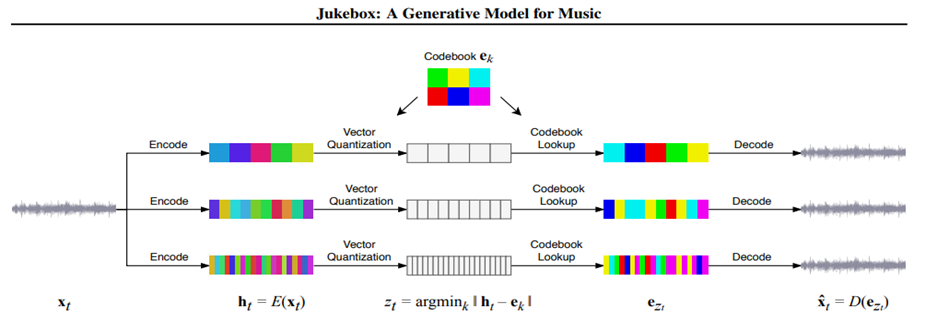
The first method we plan on using for music generation is a type of Variational AutoEncoder (VAE) called Vector-Quantized VAE. VAEs have been used successfully in new data generation, initially for images but more recently for music as well.

AutoEncoders are models consisting of two Neural Networks with the goal of accurately reconstructing compressed data. The first network, or “encoder,” compresses input data into a lower dimensionality, “latent-space” representation. The “decoder” network then reconstructs the input based on data drawn from the latent space.

In Variational AutoEncoder frameworks, rather than compressing the input data into single values, the encoder compresses data into a probability distribution of values. The decoder then randomly samples from that distribution and uses those values as input. Unlike traditional AutoEncoders, VAEs allow for generation of brand new data.

We plan to explore the VQ-VAE model similar to the one used by OpenAI’s state-of-the-art music generator, “Jukebox.” While other music generation models have been successful at generating multi-track music, most of those models generate music symbolically, and therefore constrain outputs to fixed lengths and instrument sets. And while attempts had previously been made to output generated music as a holistic piece of raw audio, maintaining long-range dependencies with data of that high dimensionality is extremely computationally complex.

VQ-VAE models have provided a successful workaround to the computational cost of generating raw audio by encoding input into lower dimensionality embeddings while retaining the most important aspect of the music.



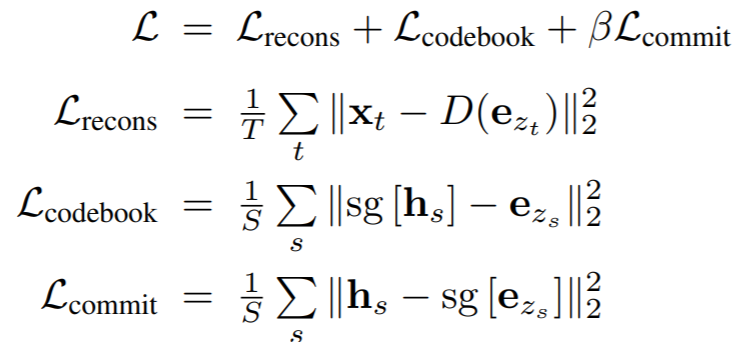
*Vector- Quantized Variational AutoEncoder*

From “Jukebox: A Generative Model for Music”

More specifically, VQ-VAE accepts raw waveform audio as input. Its encoder outputs discrete, categorical data, which are then embedded based on their similarity to an existing codebook value. The index from that codebook is passed into the Decoder and the decoder then outputs a short snippet of sound.

The loss function for VQ-VAE is composed of 3 separate measures:

* **Reconstruction Loss** penalizes the difference between the reconstructed output and the original input
* **Codebook Loss** penalizes the distance between encodings and their nearest neighbor in codebook
* **Commit Loss,** weighted by a factor β, prevents encodings from fluctuating too much by penalizing the distance between hidden representations and the codebook vectors



From “Jukebox: A Generative Model for Music”

The final step after generating these sound snippets to combine them into full tracks. To do so, after the VQ-VAE model is trained, we learn prior distributions for the latent space of Vector Quantizations. The Jukebox model learns these distributions using transformers with sparse attention, but due to their computational complexity, we will likely explore other methods of doing so (using VAEs, for example). To generate a full track, we sample sound snippets from the learned prior distributions based on a “most likely next token” algorithm. We hope to fine-tune these VQ-VAE implementations so that we can successfully generate coherent electronic music.

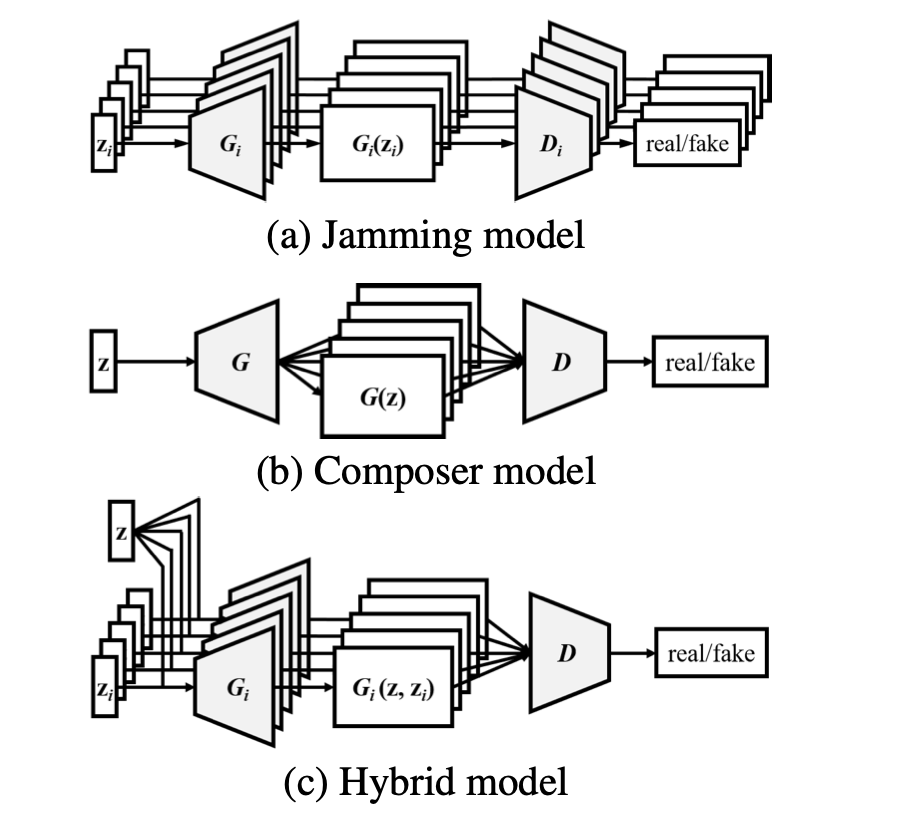
**GANs**

The second method we shall be looking into is Generative Adversarial Networks (GANs). Originally conceived for images, researchers have struggled to implement them for more sequential data such as audio and music, which necessitate a temporal model.

For some context, let’s briefly examine what a GAN actually is. GANs consist of two neural networks with conflicting goals, namely a discriminator and a generator. The discriminator has the task of determining whether or not input it is given is “real” or “fake”.

Encouraging results with GANs in past research projects lead us to believe that GANs are good candidates for melody generation. Firstly, they are capable of generating polyphonic music, moreover, the ability to include a drum kit as one of the generated tracks is a major plus. Seeing how drums are different from more traditional instruments where they are not pitch based, but rhythm based.

Several Implementations and architectures have been proposed to generate multitrack musical pieces using GANs:



*Different GAN Structures*

*From ‘MuseGAN’, Yang, et al.*

Jamming Model:

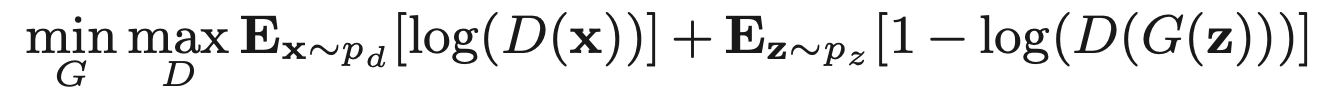
Multiple generators work independently and generate music of its own track from a private random vector , where denotes the number of generators (or tracks). These generators receive critics (i.e. back propagated supervisory signals) from different discriminators. So, to generate music of tracks, we need generators and discriminators.

Composer Model:

One single generator creates a multitrack bar of music. This model requires only one shared random vector and one discriminator, which examines the tracks collectively to tell whether the input music is real or fake. Regardless of the value of , we always need only one generator and one discriminator.

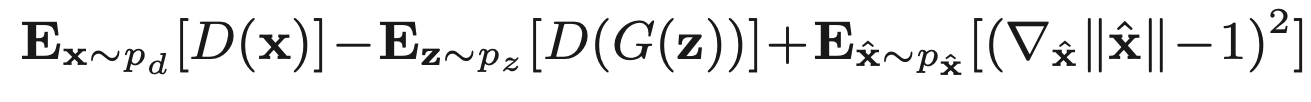
Hybrid Model:

Combining the idea of jamming and composing, we further propose the hybrid model. As we see in the figure, each of the generators take as inputs an inter-track random vector and an intra-track random vector . We expect that the inter-track random vector can coordinate the generation of different musicians, namely , The generator maps a random noise sampled from a prior distribution to the data space. The discriminator is trained to distinguish real data from those generated by the generator, whereas the generator is trained to fool the discriminator. The training procedure can be formally modeled as a two-player minimax game between the generator G and the discriminator D:



A major difference between the composer model and the hybrid model lies in the flexibility—in the hybrid model we can use different network architectures (e.g., number of layers, filter size) and different inputs for the generators. Therefore, we can, for example, vary the generation of one specific track without losing the inter-track interdependency. This is especially useful if we want to finetune one specific generator to one track to better accommodate it whilst leaving the other tracks untouched.

In 2017 (Arjovsky, Chintala, and Bottou 2017), reported that using the Wasserstein distance instead of the Jensen-Shannon divergence used in the original formulation, can stabilize the training process and avoid model collapsing. To enforce a *K*-Lipschitz constraint, weight clipping is used in Wasserstein GAN, while it is later found to cause optimization difficulties. An additional gradient penalty term for the objective function of the discriminator is then proposed in (Gulrajani et al. 2017). The objective function of D becomes



This provides us with a good starting point to proceed with training our models and testing how applicable the aforementioned loss functions are to the genre of music we are trying to produce.

Moreover, we shall be looking to encourage the generator to make use of certain musical characteristics by implementing auxiliary classification loss (Odena et al., 2017) to generate tracks that are more authentic to the electronic music genre. For example, trance music is usually produced in a D minor key – implementing an auxiliary classification loss to influence to predict the correct key could aid the generator in creating music more consistent with trance. We could extend the same procedure throughout the length of the project depending on the results we achieve during our preliminary trials.

**Temporal Structure**

Now, it is important to note that the aforementioned models can only generate multi-track music bar by bar, with possibly no temporal coherence between the generated bars, in other words, they do not sound good sequentially. To overcome this, several potential solutions have been proposed.

A close up of a logo

Description automatically generated

Generation from Scratch:

The first method is an adaptation of an idea used by (Saito, Matsumoto and Saito, 2017) for video generation. It aims to generate fixed-length musical phrases by viewing bar progression as another dimension to grow the generator. The generator consists of two sub-networks, the *temporal structure generator*  and the *bar generator* . maps a noise vector to a sequence of some latent vectors, . The resulting , which is expected to carry temporal information, is then be used by to generate piano-rolls sequentially (i.e. bar by bar):

A picture containing table

Description automatically generated

Track-conditional Generation:

The second method assumes that the bar sequence of one specific track is given by a human and tries to learn the temporal structure underlying that track and to generate the remaining tracks (and complete the song) The track-conditional generator G0 generates bars one after another with the conditional bar generator, *Gbar0*. The multi-track piano-rolls of the remaining tracks of bar *t* are then a generated by *Gbar0.* The whole procedure can be formulated as:

A close up of a logo

Description automatically generated

A picture containing screenshot

Description automatically generated

**Quantum GANs**

In classical machine learning, GAN have proven useful for generative modeling. These algorithms employ two competing neural networks - a generator and a discriminator - which are trained alternately. This has inspired quantum researchers to propose a quantum variant, QGAN. Replacing either the generator, the discriminator, or both with quantum systems translates the framework to the quantum computing context (Lloyd, S. & Weedbrook, 2018). The hybrid quantum-classical (HQC) framework has been adopted as the *de facto* strategy to design practical algorithms in the near term. The basic idea behind the HQC framework is that a computational problem can be divided into several subtasks, several of which can be executed more efficiently using a quantum computer while the rest can be deployed to a classical computer.

Quantum algorithms have the potential to outperform their classical counterparts in a variety of tasks. The realization of the advantage often requires the ability to load classical data efficiently into quantum states. However, the best-known methods require gates to load an exact representation of a generic data structure into an *n*-qubit state. This scaling can quickly predominate a quantum algorithm's complexity and, thereby, impair potential quantum advantage.

For our project we shall aim to implement a hybrid quantum-classical algorithm using, initially, quantum generators and classical discriminators. More precisely, we use quantum Generative Adversarial Networks (QGANs) to facilitate efficient learning and loading of generic probability distributions - implicitly given by data samples - into quantum states. Through the interplay of a quantum channel, such as a variational quantum circuit, and a classical neural network, the QGAN can learn a representation of the probability distribution underlying the data samples and load it into a quantum state. The loading requires ) gates and can enable the use of potentially advantageous quantum algorithms, such as Quantum Amplitude Estimation. We implement the QGAN distribution learning and loading method with Qiskit and test it using a quantum simulation and actual quantum processors provided by the IBM Q Experience.

With Grover's algorithm, we know the target quantum state right from the start, whereas in QGAN, we do not know the quantum state explicitly, *a-priori* but only have access to classical data samples. QGANs have been applied to classical data, for example, in finance, where they were used to facilitate financial derivative pricing. Ultimately, we hope to replicate the same network architecture that we’ve implemented with classical GANs to produce multitrack music.

**PROJECT TIMELINE**

**RISKS & CHALLENGES**

**Data Availability & Preprocessing Bottlenecks**

There are lots of audio data on the Internet. But securing data that pertains to the genre we would like to train our model to will always be a challenge. For example, data for some iconic songs may only exist partially

Converting midi files to audio data or converting audio data to a format that our models can train on may take long as the model may require numerous songs to train on

These two points may become bottlenecks

Bottleneck 1: Getting the right audio data for iconic music pieces (ie. saving a song as a midi file)

Bottleneck 2: Mass conversion may be necessary for the project to move forward

**Computation**

Dependencies

Pipeline

Most models may require intense computation, utilizing multiple GPUs

**Electronic Music**

Another risk lies in dealing with electronic music in particular. Electronic music is an extremely broad genre, as it is technically defined only in that the sounds produced are circuit-based. The lack of inherent musical structure in electronic music may make it difficult for neural networks to properly learn its characteristics. To deal with this, we may need to narrow our scope to certain subsets of electronic music, perhaps subsets with more explicit musical structure than others.

**Evaluation Metrics**

It is particularly difficult to evaluate models that write music, as works of art are inherently subjective in their appreciation. Certain studies have assigned objective metrics to evaluate AI-generated music, such as measures of polyphonicity and tonal distance. These metrics help identify structure within generated music, however, there is no accurate way to quantify how “good” a song is.

As other studies have done, we hope to rely on a combination of objective and subjective metrics. Subjective metrics will involve distributing generated tracks to objective and diverse listeners and asking them to rate certain aspects of the track. This can be a complicated task, and even if successful, it relies on opinion rather than numbers.

**Loss Functions**

We may need to revise/test it continuously, to improve it

**Q GANs**

Difficult data gathering

Rigid structure

**FINAL OUTCOME**

Our goal is to generate a brand new piece using new sounds and consisting of compatible layers. Hopefully this process will spark our creativity as both artists and data scientists. And we hope that other artists can use these processes and results to foster greater creativeness in their own work.

**REFERENCES**

van den Oord, Aaron and Dieleman, Sander. “WaveNet: A generative model for raw audio.” *DeepMind.com*, 8 Sep. 2016, https://deepmind.com/blog/article/wavenet-generative-model-raw-audio.

Putz, Volkmar, and Karl Svozil. “Quantum Music.” Soft Computing 21.6 (2015): 1467–1471. Crossref. Web.

Jesse Engel, Cinjon Resnick, Adam Roberts, Sander Dieleman, Douglas Eck, Karen Simonyan, and Mohammad Norouzi. "Neural Audio Synthesis of Musical Notes with WaveNet Autoencoders." 2017.

Engel, Jesse, et al. “GANSynth: Adversarial Neural Audio Synthesis.” *ArXiv.org*, 15 Apr. 2019, arxiv.org/abs/1902.08710.

“5.” *Guide to Unconventional Computing for Music*, by Eduardo Reck Miranda, Springer, 2017.

Adam Roberts, Jesse Engel, Colin Raffel, Curtis Hawthorne and Douglas Eck. “A Hierarchical Latent Vector Model for Learning Long-Term Structure in Music.” *ArXiv.org*, 11 Nov. 2019, arxiv.org/abs/1803.05428.

Prafulla Dhariwal, Heewoo Jun, Christine Payne, Jong Wook Kim, Alec Radford and Ilya Sutskever. “Jukebox: A Generative Model for Music.” *ArXiv.org*, 30 Apr. 2020, https://arxiv.org/abs/2005.00341.

Velardo, Valerio. “Deep Learning (For Audio) With Python.” *Github Repository*. 5 Feb. 2020, https://github.com/musikalkemist/DeepLearningForAudioWithPython.

Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang and Yi-Hsuan Yang. “MuseGAN: Multi-track Sequential Generative Adversarial Networks for Symbolic Music Generation and Accompaniment.” *ArXiv.org*, 19 Sep. 2017, https://arxiv.org/abs/1709.06298.