Utilizing Advanced ML Techniques for Structured MIDI Generation

Daniel Foulkes-Halbard
Candidate number: 246736
Supervisor: Dr Kingsley Sage
Degree Course: Computer Science and Artificial Intelligence

September 11, 2024

1 Abstract

This report details a project that leverages transformer technologies and advanced machine learning techniques to produce convincing generative music from MIDI training data. A music generation model was created using the Mistral 7B transformer model that through evaluation was proven to be capable of producing music that listeners found difficult to distinguish between the model generated and human generated music. The project explored different ways to optimise the pipeline to enable working with hardware bottlenecks introduced by the increased computational power required for large sequence generation tasks. Different hyper-parameters for generation were also explored and evaluated with user testing to determine what a sensible configuration was for the model to output musically correct pieces.

Contents

2 Introduction

There has been significant interest and development in the field of automated music generation over the last sixty years. With the recent advancements in AI and machine learning technology, the prospect of completely AI generated pieces of music is slowly becoming a feasible reality. Large scale organisations such as Google and OpenAI recognise the demand by the general public, hobbyists, professional musicians and organisations looking for good quality automated music. Googles Magenta project is a plugin for Ableton Live that lets users utilize machine learning to generate or augment pieces using transformer-based models. [22] OpenAI have their own project named MuseNet which aims to produce coherent and structurally sound musical output based on a sparse transformer architecture. [14] Significant research and development has gone into both projects for both symbolic and raw audio generation from scratch as both companies recognise the huge scale of possibility and demand for such technology.

2.1 Aims

The project's objective is to produce a machine learning model that can output generated MIDI files. These files will be multi-instrumental and integrate into any MIDI player or editor of choice. I envision these files serving music creators either as complete compositions or as customizable segments to enhance their personal musical projects.

As part of the core objections for this project, I will conduct background research into effective existing methods of music generation and how they work. This will be done through both research papers, reports and online sources such OpenAI's MuseNet and GoogleAI's Magenta. I will employ a transformer based machine learning architecture that will use sequence data from existing symbolic MIDI datasets to design a model that is able to generate monophonic MIDI files that will apply short and long term structure and a sensible track length. Upon testing and evaluation, the output should be difficult to distinguish between the models output and human made music.

If the project is going well and in good time, I have outlined some stretch objectives that will be implemented. I will the add ability for the model to generate multi-track MIDI outputs. These multi-track files will still have to hold the values set in the core objective of maintaining the track structure with the additional requirement of keeping musical integrity between the different instrument parts. For example, if a track with two instrument parts where the first track plays the base melody and the second track is an accompanying harmonic the two tracks must work together musically. The ability of the model will be confirmed with user testing. The tests will include tasks that ask users to identify between human made and the AI made tracks from the model and clarify what made them think tracks were AI made. It will also ask users to rate tracks from the model on how well they liked it.

If disaster strikes or unforeseen circumstances mean the project is delayed or severely impacted, I have a list of fallback objectives that should be realistic to achieve in dire circumstances. The first fallback objective is a model that can output monophonic MIDI files with short term structure but not necessarily long term length or structure. The outputted files would likely be shorter music snippets that can still be used by the user. This model will be embedded in a very simple interface that the user can download the file from.
A summary of the projects objectives are as follows:

2.2 Core Objectives

- Conduct research of existing methods used to generate music.
- Create an effective a machine learning architecture to generate a well structured MIDI track that can used as a base melody for someone who might want to use or edit it in software such as Ableton.
- Produce a model that can generate a monophonic MIDI output that maintains integrity in the following areas: Sensible track length, local and global structure.
- Generated tracks should be difficult to distinguish between the generation and human made music.

2.3 Stretch Objectives

- Add multi-track capabilities to the model, allowing extra instrument tracks to be produced on top of the original generation whilst still maintaining musical integrity between tracks.

- Conduct user testing to confirm models ability to produce music well and use the data to gain insight into how the model can be improved in the future.

2.4 Fallback Objectives

- Create a model that can produce monophonic MIDI files with short term structure.
- Create a very simple GUI for users to create and download MIDI tracks for their own purpose.

3 Professional considerations

In this section I will outline how this project relates to the four key principles of the BCS code of conduct.

3.1 Public interest

The public and environments health, privacy, security and well being will not be effected by this project. The final outcome will not deal with any user's private data and will pose no risk. All third party work such as code, software, datasets or ideas will be credited appropriately. The development and final application will be conducted to a non discriminatory standard. The final application will be accessible to anyone who wishes to use it.

3.2 Professional Competence and Integrity

The project is within my capabilities to achieve with a solid foundation from my university course. I will be continually expanding my knowledge and skill set throughout this project as I research and test different approaches and ideas. I welcome and accept alternative viewpoints and criticisms of my work in the hope I can improve the final outcome of the project after taking different opinions on board. This will be especially useful for a music generation task as music is often subjective. There is no risk of injury to others, their property, reputation, or employment by any actions taken through this project

3.3 Duty to Relevant Authority

This project will be carried out with due care and diligence in accordance to the requirements of the University of Sussex. I will avoid any conflict of interest between myself and the relevant authority. I will accept professional responsibility for the work carried out in my project and their are no colleagues working under my supervision. I will not disclose any confidential information except with permission from the University, or as required by legislation. All work and performance of the final application will be presented truthfully and as it works. And I will not take advantage of a lack of relevant knowledge or inexperience to anyone the work is presented to.

3.4 Duty to the Profession

I accept my professional duty to uphold the reputation of the profession and to improve professional standards by participating in their development, use and enforcement. I will respect and encourage all my peers and will support them in their professional development.

4 Background research

4.1 Foreword

A special mention is warranted to reference [26] as an integral part of of this projects background research. This paper is an in depth evaluation of music generation and has acted as both a great foundation of knowledge in this sector and a means in which to explore more in depth and relevant works.

4.2 Real world examples

4.2.1 OpenAI's MuseNet

MuseNet is an impressive project from OpenAI designed to generate four minute long music compositions with up to ten different instrument tracks. The model uses a sparse transformer and attention architectures to achieve this. [13], [14]

4.2.2 Google AI's Magenta

Magenta is an open source project from Google AI exploring the role of machine learning in the creative process. It offers a plug in for Ableton Live called Magenta Studio whereby users can utilize AI to create new MIDI tracks or augment existing tracks. They use a transformer based model trained on the MEASTRO dataset, which is a dataset created by google and its partners for this project. MAESTRO is available for public use. [22, 23, 1]

4.3 Symbolic music representation and frameworks

Symbolic music generation refers to the method of generating pieces of music in the form of a sequence of symbols that can capture the essence of a piece of music using machine learning techniques to learn the relationships between the symbolic features. The most widely used format for symbolic music is the MIDI (Musical instrument digital interface) protocol. [26, 43, 10]
The challenges involved in generating symbolic music from scratch include, but are not limited to: [26]

- Long and short term structure
- Natural endings
- Musical consistency between different instruments
- Capturing mood or emotion
- Producing music in a certain style or genre

4.3.1 Symbolic and raw audio generation

An alternative approach to symbolic music generation is raw audio generation. Raw audio deals with generating audio wave forms directly, usually in the form of digital audio signals. This method focuses on capturing lower level nuances such a tone, mood and feel of a piece of music whereas symbolic music is usually focused on higher level attributes such as musical structure and form. A good example of raw audio generation is WaveNet, which uses a fully probabilistic deep neural network to output raw audio.[16] Some projects have attempted to combine the two methods in the hope of capturing both high and low level characteristics in pieces of music. As this project focuses heavily on musical structure and the ability for the generated music to be accessible and editable, symbolic generation is the correct way forward.

4.3.2 MIDI

As the most commonly used and widely available symbolic music representation, this project will be working with and generating MIDI files. MIDI is a system that can contain information about a tracks musical performance regardless of what instrument is being played. This makes it perfect for this project as music can be generated and instruments applied to the track at the users whim. The main types of data that can be extracted from a MIDI track sequence of notes are note pitch(scale from 0-127), note start, end and duration times, note velocity (how hard a note was played) and pitch bend. This is commonly known as event based music.[26, 10, 43]

I will be using the Python programming language to develop my models. There are some useful Python packages that have been developed specifically to use MIDI files in the language. I have outlined some of the ones I considered using below.

4.3.3 MidiTok

MidiTok is a python package specifically made for symbolic music generation and deep learning which makes it a great candidate for this project. It can take care of the most well-known tokenization techniques and bye pair encoding of MIDI data for use in models such as transformers.[20]

4.3.4 Pretty_MIDI

Another python package that can handle MIDI data is pretty_midi. It was designed to be simple to use and offers a way explore MIDI data hierarchically. I intend to use this package in the initial stages of development to help me explore MIDI data and get a very simple prototype going. [33]

4.3.5 Music21

A Python based toolkit that allows the utilization of the fundamentals of musical theory with its application which is obviously incredibly useful. The package also comes with its own dataset designed to be used with it. Considering it comes with its own dataset and is able to extract intricate musical theory data from MIDI files it is a great package to get familiar with, although it is not specifically designed to be used with deep learning like MidiTok. [15]

4.3.6 Package decision

Package	Usability	Tokenization	Dataset	Deep learning specific
MidTok	5	yes	no	yes
Pretty_MIDI	5	no	no	no
Music21	3	limited	yes	no

Figure 1: Decision matrix for python packages. Usability scored from 1 to 5, with 5 being the best.

Based on the comparison table in figure1 it is highly likely I will use MidiTok as the main tokenizer package when designing my model workflow. It offers an easy to use package that has all standard tokenization methods inbuilt for both pre-processing and post processing and was designed with deep learning models in mind.

4.4 Deep learning Frameworks

4.4.1 Hugging Face

Hugging Face is an open source machine learning community with Python libraries available for developers for transformers, datasets and tokenizers. It also offers a a large potential for collaboration and sharing with other developers as well as discussion and support forums to help with potential problems. The site boasts a large array of GPT transformer models made by reputable companies that can easily be deployed for this project. This is a fantastic tool as the alternative would be to try and program a transformer model which could prove extremely challenging. Additionally , every technique employed through the Hugging Face libraries are well documented and include references to their respective research papers. [5]

4.4.2 MistralAI

Mistral is a small AI research and development company based in France. They have developed an AI chat bot similar to OpenAI's ChatGPT. They have also developed and open source transformer model named Mistral-7B-v0.1 which is available on the Hugging Face transformer library and is originally designed for text generation tasks. This project is very similar to text generation tasks as MIDI music generation and text generation are both sequence generation tasks meaning that it can employ this model for MIDI generation taking advantage of natural language engineering techniques. The model is decoder only meaning that instead of the usual encoder decoder stack frequently employed by transformer models this model only contains decoders as we do not have a need to encode inputs into vector representation like we would need to if we were doing classification or clustering. Instead we can input our tokenized MIDI data straight into the decoder stack. Each decoder uses multi-headed attention, multi layer perceptrons with a SiLU activation function, a normalisation layer and an output layer of the size of the vocabulary. Thanks to the Hugging Face training libraries it is very simple to employ this model for MIDI generation and set hyper-parameters such as learning rate and weight decay. [8][27]

4.5 Datasets

4.5.1 Lakh MIDI Dataset

The Lahk MIDI dataset is the largest currently available MIDI dataset you can find, comprised of 176,581 unique MIDI files. The goal of its creation was to facilitate music information retrieval on as large a scale as possible, which includes symbolic information extracted directly from the MIDI files. The dataset contains 45,129 MIDI files which have been matched to actual songs from the Million Song Dataset, meaning these files are known to be actual musical pieces that people enjoy and therefore can be reliably used as training data for this project. Some other smaller datasets that are often used such as the ADL Piano MIDI Dataset are a smaller collection of files taken from the Lakh dataset. [11]

4.5.2 BitMidi

This website allows users to browse, listen and download MIDI files curated by volunteers. As of my last visit to the site (December 2023), it boasted a collection of 113,229 files. It is an easy site to use but as all files are uploaded by volunteers they would need to be listened to as a type of quality control to ensure the integrity of the training dataset. [2]

4.5.3 Augmented Design Lab(ADL) Piano MIDI Dataset

The ADL piano dataset is a collection of files made from the Lakh dataset. It was created by extracting all the "piano family" (MIDI program numbers 1-8) parts from the matched subset in the Lahk dataset which generated 9,021 Unique files mainly in the classical and rock genres. Additionally, 2,065 more files were added to the dataset from publicly available sources on the internet. [19]

4.5.4 Classical Archives

Classical Archives is a site where you can download expertly curated classical MIDI files from all of the biggest names in classical music. After some testing, the files themselves all appear to be of good quality meaning they are faithful to the source pieces and have been expertly made. However there is a limit to how many you can download daily without paying. [9]

4.5.5 Music21 and J.S. Bach's

The python toolkit Music21 has a small inbuilt dataset known as the J.S. Bach's (JSB) dataset consisting of 382 chorales. In reality this dataset is not large enough by itself to be used for this project, but may prove useful in initial prototyping stages.[15][12]

4.6 Pre-Processing

Typically machine learning models require numerical data in a standardized form to perform efficiently. MIDI data contains information about various musical events and can vary in complexity and structure therefore the MIDI data needs to be converted to an input format that models can effectively train and learn from. Various tokenization techniques have been employed to get data from MIDI files into a format that a deep learning model can utilize effectively. [20] A common approach to encoding MIDI data is one-hot encoding, which essentially maps unique MIDI events into binary vectors with a length equal to the possible events, helping the model learn the relationship between different music events which can be fed into an RNN. [24] The following sections contain some tokenization methods that seemed the most appropriate when conducting research and are not by any means the only tokenization methods that can be used.

4.6.1 Revamped MIDI (REMI) and REMIPlus

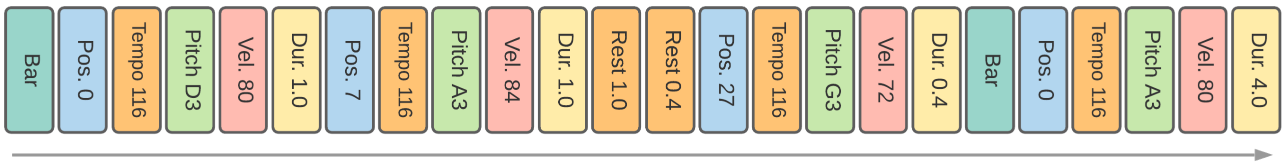


Figure 2: REMI tokenization visualisation. Source : [7]

REMI is a tokenization that represents notes as sequences of the following event data:

- **Pitch**: The note being played.
- **Velocity**: how "hard" the note is played.
- **Duration**: how long the note is played for.
- **Bar**: to indicate a new bar is beginning.
- **Position**: the current position within the current bar.
- **Tempo**: event each beat to represent expressive tempo freedom if wanted.

The main idea was to introduce bars and positions in a bar to allow the model to produce a metrical grid to model music in a beat based manner. This data representation allows the model to learn rhythmic regularity more clearly. [25] REMIPlus was introduced and built on top of REMI to enable a model to generate multi track outputs by adding program and time signature tokens as well. [42] The Midtok python library uses REMIPlus by default and seems well suited for transformer based sequence generation.

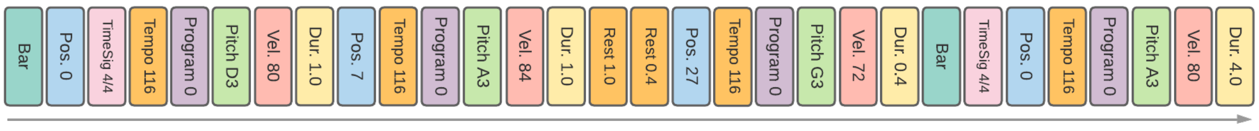


Figure 3: REMIPlus tokenization visualisation. Source : [7]

4.6.2 MIDI-Like

Like the name suggests, MIDI-Like tokenizations works by simply converting native MIDI messages such as pitch, note on, note off and time shifts into tokens that can be fed into a basic RNN architecture model as 143-dimensional one-hot vectors. [30]

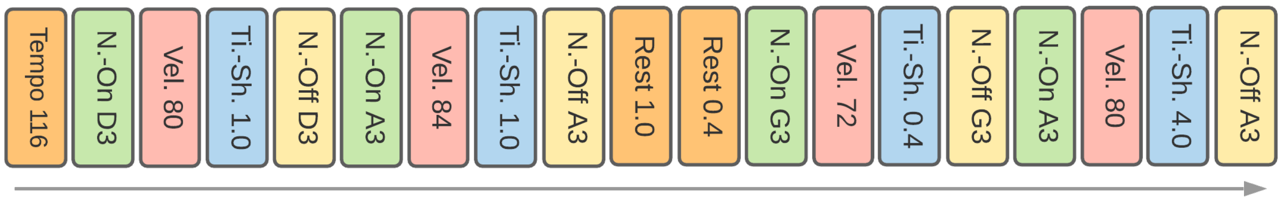


Figure 4: MIDI-Like tokenization visualisation. Source : [7]

4.6.3 OctupleMIDI

A tokenization method designed to reduce the sequence length of training data when compare to methods like REMI. It encodes each note into a tuple with 8 elements including time signature, tempo, bar, position, instrument, pitch, duration and velocity. It claims to use training sequences as much as four times shorter than REMI especially with multi track outputs. Octuple captures enough information to capture the important musical data such as time changes and note duration whilst being a universally feasible method of tokenization.[45]

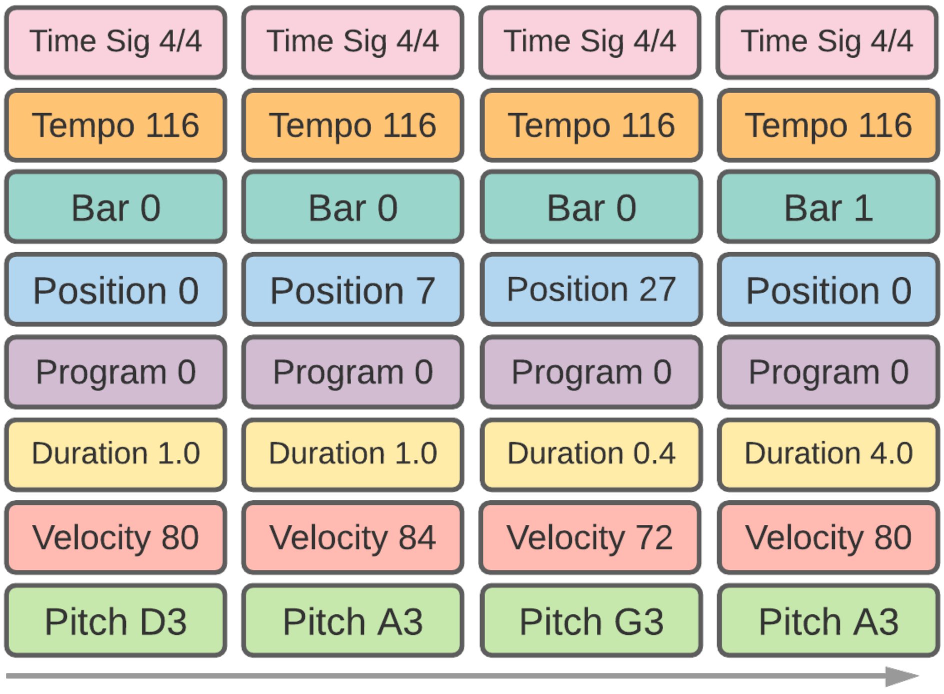


Figure 5: OctupleMIDI tokenization visualisation. Source : [7]

4.6.4 Multi-track MIDI representation (MuMIDI)

MuMIDI was made specifically for multi-track tasks using embedded pooling as a way to represent harmony between different tracks. The main objective was to encode all tracks in a file in a single token sequence. This method introduces the track token which precedes the note token to indicate what track they belong to. This outperforms REMI in multi track harmonization as well. [34]

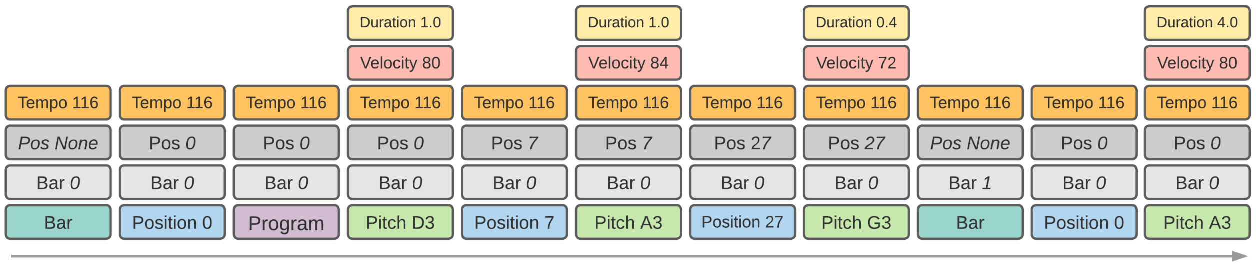


Figure 6: OctupleMIDI tokenization visualisation. Source : [7]

4.6.5 Tokenization decision

OctupleMIDI offers a tokenization method that is simple in structure, requires less resources to process and can be used for both monophonic and polyphonic music generation. However, modern transformer based sequence generators work well with one dimensional input sequences such as REMIPlus which will likely be the tokenization method favoured throughout this project.

Method	Mono/Multi	Resource Req.	Token Simplicity
REMI/REMIPlus	Both	Large	2
MIDI-Like	Monophonic	Medium	2
OctupleMIDI	Both	Small	1
MuMIDI	Multi	Medium	3

Figure 7: Decision matrix for Tokenization methods. Simplicity scored from 1 to 3, with 1 being the most simple.

4.6.6 Byte Pair Encoding (BPE)

BPE is a compression technique that maps sub-sequences of frequently occurring pairs of bytes to a different character. For example with the sequence ’abbcbbdbbe’ we can map the sub-sequence ’bb’ to another character such as ’f’ which would change the sequence to ’afcfdfe’. We can keep doing this with frequently occurring sub-sequences until our specified vocabulary length has been reached. It is a very commonly used technique in NLP tokenizations that allows rare or unknown words to be encoded as sub sequences. It is also an effective memory management device as longer sequences are compressed down to their sub-sequence sequences saving memory. BPE can be effectively deployed when using symbolic music as well such as MIDI, especially on a music generation task such as this. Just like we would if this were an NLP task, we can tokenize the MIDI events and apply BPE encoding to receive all the same benefits. [4][36]

4.6.7 One-Hot Encoding

This is a feature engineering technique commonly used in machine learning. Its main goal is to be able to transform categorical data in a numerical format. To achieve this we represent each category as a binary vector where only one element (in the case of this project one possible token in our vocabulary) is set to 1 and all the other are set to 0. This can be a better solution to the usual vocabulary encoding where our tokens might be encoded like ’vocab1:0, vocab2:1’ and assigning ID’s in ascended numerical order we can introduce a sort of ”ranking” for different vocabulary tokens that might influence the predictions. By one-hot encoding we can un-rank these vocabulary tokens to potentially help improve the overall performance of the model. This method was used in the OpenAI and Google Magenta models. However for this project this method may not be appropriate for the Mistral model used or the hardware limitations for this project. While it has advantages One-Hot encoding can also drastically increase memory usage and computation time as input dimensionality is greatly increased. [3] [35]

4.7 Deep learning techniques

To generate music we must first design a model architecture that can interpret the tokenized MIDI data and learn the relationships in the complexities of the music between notes and tracks. In the paper [26], various model architectures are discussed such as recurrent neural networks (RNNs), convolution neural networks, generative adversarial networks and transformers. For this project I am specifically interested in RNNs and transformers as they appear the most in literature and real world examples.

4.7.1 Exploding and Vanishing Gradients

The vanishing gradient problem occurs when the models parameters become so small they begin to contribute little or negligible amounts to the learning of the model so it fails to converge on a solution. The exploding gradient is the opposite problem to the vanishing gradient problem where the parameters of the model become so large that updates to the parameters during training become unstable. [31]

4.7.2 Recurrent Neural Networks (RNNs)

RNNs are purposely designed to work with sequences of data which makes it the perfect starting point for this project which will be largely dealing with data sequences from MIDI files. By maintaining and updating an internal state that acts like the models ”memory” they are able to exploit the notion that sequences can be expressed and predicted by relationships with past information. It is this internal memory state that distinguishes RNNs from normal feed forward networks. While this is very powerful when needing to predict sequences it can also be computationally expensive when presented which large sequences. There is also two commonly known issues with RNNs. The vanishing gradient problem and the exploding gradient problem which happen due to the nature of backpropogation. [21]

4.7.3 Long short-term memory (LSTM)

LSTMs were designed as a way to combat common problems in RNNs known as the vanishing and exploding gradient problems mentioned above. They do this by introducing a gating mechanism into the model architecture. There are three types of gates introduced:

- **Input gate**: determines how much information is stored in the cell state.
- **Forget gate**: determines which information gets kept or discarded in the cell state.
- **Output gate**: determines how much of the information in the cell state is exposed to the next layer.

The cell state acts as the models memory. It is the LSTMs ability to regulate its internal cell state that sets it apart from regular RNNs. [21]

4.7.4 Transformers

Transformers were first introduced in 2017 and revolutionised the way sequences of data could be processed. The core of the model is it's self attention mechanism that allows the model to dynamically focus on different parts of an input sequence which is particularly useful for music generation tasks. The self attention also allows the model to capture dependencies in the data and process the data in parallel in a more flexible manner making it suitable for very large sequences and complex tasks. The self attention mechanism is used in both encoder and decoder stack layers. The encoder turns the input sequence of tokenized data into a form that captures the context around the music by projecting the input into a higher vector space representation, and the decoder uses that data to generate an output that is informed by the encoded representation and previous decoder outputs. Another key advantage of transformers is the ability for parallelised training, so you can train multiple batches at a time utilizing the full capacity of the hardware you have available. [40] [41]

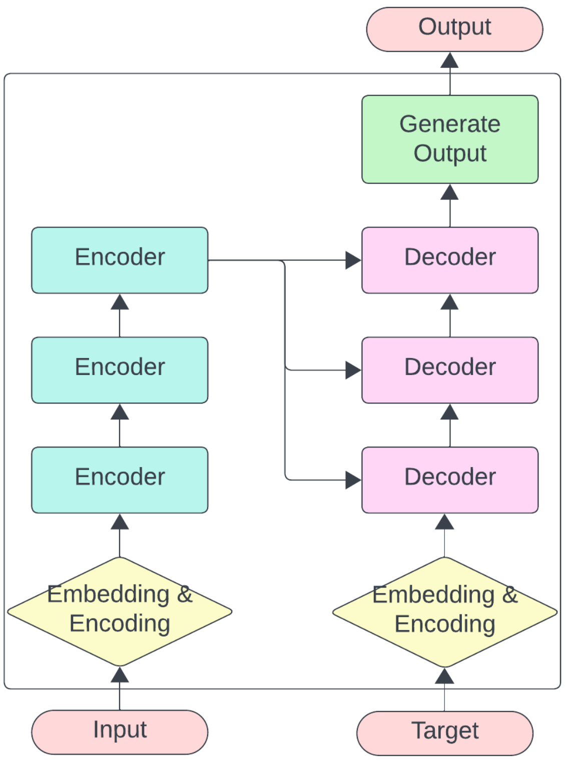


Figure 8: Diagram of possible transformer architecture. The encoder on the left and decoder on the right. [17]

4.7.5 Batches

When loading data into your model training there are two options. You can either load all the data in as one and feed it through the network or you can break all the data down into batches and feed the batches through one at a time. Using batches in this way can help with efficient memory management and can improve training speeds when running a models training on GPUs that support parallel processing. Batch size and batch training will be a very important parameter in this project as there is a huge amount of training data and a hardware bottleneck for memory usage.[6]

4.7.6 Root Mean Squared Normalization (RMSN)

RMSN is a layer normalization technique commonly used in deep learning. The main purpose is to stabilise the training of a model by preventing rapid exploding or diminishing gradients during network backpropogation.

1. First the for any given layers input, the root mean squared (RMS) value of the neuron activation is calculated by squaring each element, averaging the values across features for each batch and then taking the square root of the average.

$$\text{RMS} = \sqrt{\tfrac{1}{n} \sum_{i=1}^{n} x_i^2}$$

Figure 9: RMS Formula where x is the input sequence.

2. Each element is then divided by the RMS value, normalizing the scale of the inputs to each layer.

$$\hat{x}_i = \frac{x_i}{\text{RMS}}$$

Figure 10: Normalization where x_i is the i-th element of the input and $\hat{x}_i$ is the normalized value of x_i.

RMSN is particularly important in transformer training as it is crucial for ensuring the stable activation of the many neurons in the networks and provides a simple, effective efficient way to achieve this. [46]

4.7.7 Learning Rate and Learning Rate Decay

One of the most important hyper parameters in deep neural networks is the learning rate. This value determines how much the weights of the network are updated during backpropogation to minimize the loss of the network. It is important to pick the right learning rate for the model because if the learning rate is too high it risks making updates to the weights that are too large that will potentially bounce around the global minima instead of converging on it. Equally, a learning rate that is to small will fail to update the weights enough to make a significant impact on the training. It is important to explore a wide range of learning rates whilst training the model to maximise the changes that you converge on the global minima. A common approach to exploring the learning rate is to start the training on a higher learning rate and gradually minimise the rate as training progresses. More recently researchers have started using a learning rate warm up called simulated annealing which increases the learning rate from a low value up to the maximum value over a small amount of the training at the beginning and then starting the learning rate decay. The intuition behind the warm up is that models are initialised with random weights at the start of training therefore the initial weights are very likely far from the optimal values so a large learning rate from the start can cause instability in the models training. [29]

4.7.8 Optimizers

Optimizers play a critical role in machine learning model training. They are the algorithms that are used to update key values in the model such as weights and learning rate. It is the role of the optimizer to guide the training of the weights to they converge on the global minima as accurately and efficiently as possible. Some common optimizer choices are: [38]

- **Stochastic Gradient Decent (SGD)**: Updates the parameters θ by a fixed learning rate η based on the gradient of the loss. In high dimension space SGD is much more likely than other methods to get stuck in a local minima, bounce around optimal points or progress very slowly towards the global minima.

$$\theta = \theta - \eta \cdot \nabla_\theta L(\theta)$$

Figure 11: SGD formula.

- **Momentum**: Often called SGD+Momentum, it works by accelerating SGD by introducing a 'velocity' term v in the algorithm which is calculated from previous steps by multiplying it by a constant γ value. This technique is usually fast to converge on a point but can often overshoot or circle the global minima or still get stuck in local minima in a complicated gradient space.

$$v = \gamma v + \eta \cdot \nabla_\theta L(\theta)$$
$$\theta = \theta - v$$

Figure 12: SGD+Momentum formula.

- **AdaGrad**: Adapts the learning rate by scaling it by the square root of the sum of the historical squared values plus some constant ϵ. Let G_t, i be the sum of the squares of the gradients up to the time step t. This allows progress along 'steep' gradient directions to be damped and progress along 'flat' gradient directions to be accelerated adding a kind of gradient space smoothing to the optimization. Over time the learning rate decays to zero implementing learning rate decay.

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,i} + \epsilon}} \cdot \nabla_\theta L(\theta_i)$$

Figure 13: AdaGrad formula.

- **RMSProp**: Sometimes called 'Leaky AdaGrad'. It is a modified version of AdaGrad that introduces a 'decay rate' β to calculate G_t instead of simply taking the sum of the squared gradients. This performs better in more complicated gradient spaces but is not necessarily fast.

$$G_t = \beta G_{t-1} + (1-\beta) \cdot (\nabla_\theta L(\theta))^2$$

$$\theta = \theta - \frac{\eta}{\sqrt{G_t + \epsilon}} \cdot \nabla_\theta L(\theta)$$

Figure 14: RMSProp formula.

- **Adam**: Combines elements of both Momentum and RMSProp. m_t is the first moment (mean) of the gradients, β_1 is the decay rate similar to v in momentum. v_t is the second moment calculating the an exponentially decaying average of past squared gradients using β_2 is another decay rate. Adam also incorporates bias corrections mechanisms $\hat{m}_t$ and $\hat{v}_t$. Both m_t and v_t are both initialized as vectors of zero so tend to be biased towards zero during the early steps of training and with decay rates close to 1. Finally Adam updates the parameters θ using the bias corrected momentum and RMSProp moments with adaptive learning rates. Adam does well in both sparse and noisy gradient spaces and can converge on global minima efficiently and effectively but still not quite as fast as SGD+Momentum.

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) \cdot \nabla_\theta L(\theta)$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) \cdot (\nabla_\theta L(\theta))^2$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

$$\theta = \theta - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \cdot \hat{m}_t$$

Figure 15: Adam formula.

In summary Adam is a popular choice for most models for the benefits outlined above and its ability to adapt to different gradient spaces. SGD+Momentum can outperform Adam in some cases but may require more tuning of the learning rate and an optimized learning rate decay scheduler.

4.7.9 Early Stopping

This is a regularisation method used to prevent a machine learning model over-fitting. The basic concept is that throughout the training of the model certain metrics are monitors such as loss on the training and validation sets to determine how well the model is converging on an optimal point. Using these metrics we can determine if the model is over-fitting to the training data by measuring the training loss against the validation loss. We can define a stopping criterion as a simple integer like 3 which means if we notice that the loss on the validation set is no longer decreasing after an epoch three times in a row we can stop training early to avoid over-fitting and use model checkpoints (saving models after each x epochs) to load in the best model for use. The advantages of Early stopping include: [32]

- Prevents over-fitting by stopping the model learn the training data to much.
- Saves time and resources by reducing unnecessary training epochs.
- Acts as a kind of hyper parameter tuning finding the optimal number of training epochs needed.
- Improves overall performance of model by preventing over-fitting

4.7.10 Activation Functions

Activation functions are the non linear functions used in machine learning techniques such as neural networks that enable a network to learn much more complex relationships between inputs and outputs an enable the network to explore problems in higher dimensional spaces. They also play a crucial role in the backpropogation phase of network training as the gradient of these functions is used with respect the the gradient of the loss of the model to update the weights of the network. Some commonly used activation functions include: [18]

- **Sigmoid (Logistic Function)**: This maps the output value between 0 and 1. Good for classifications problems but can suffer from vanishing gradients.

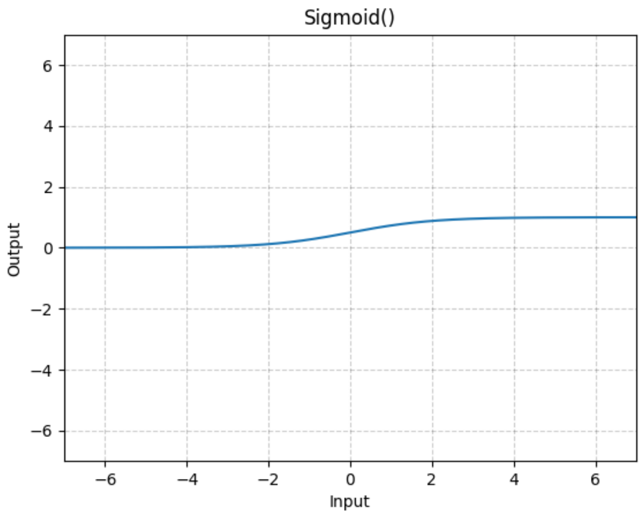


Figure 16: Sigmoid activation function plot

$$\text{Sigmoid}(x) = \sigma(x) = \frac{1}{1 + \exp(-x)}$$

Figure 17: Sigmoid activation function formula.

- **Sigmoid Linear Unit (SiLU)**: Also known as the Swish function, is a self gating activation function. Its takes the Sigmoid function σ and multiplies it with the the input x. SiLU has been shown to outperform ReLU in various deep learning models and excels in image recognition and natural language processing. This is the activation function used in the Mistral 7B model.

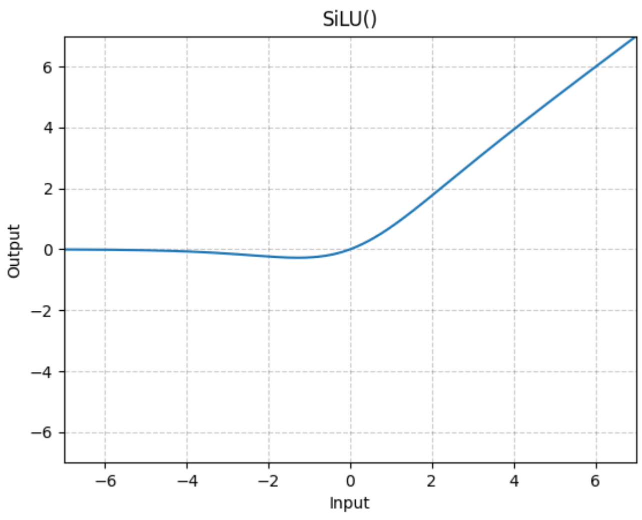


Figure 18: SiLU activation function plot.

$$f(x) = x \cdot \sigma(x)$$

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Figure 19: SiLU activation function formula.

- **Hyperbolic Tangent (tanh)**: Tanh acts similarly to Sigmoid but maps values between -1 and 1 and is centered around 0 which can aid in backpropogation in hidden layers. However is also suffer from vanishing gradients and can produce dead neurons where the value is 0 and the neuron does not learn.

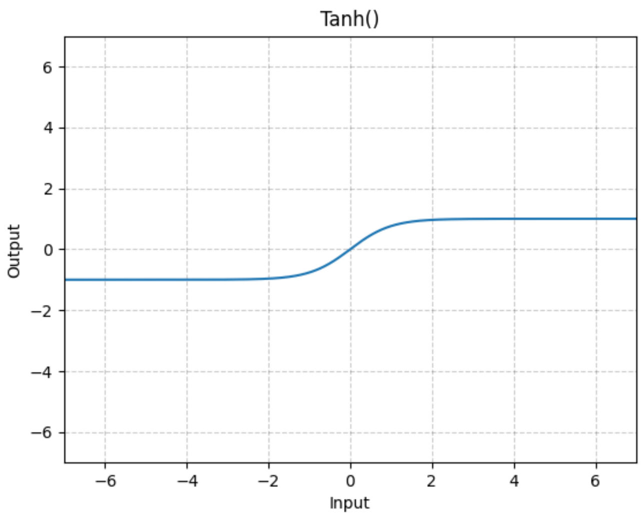


Figure 20: Tanh activation function plot.

$$\text{Tanh}(x) = \tanh(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}$$

Figure 21: Tanh activation function formula.

- **Rectified Linear Unit (ReLU)**: ReLU is the most widely used activation function due to its simplicity and effectiveness when tackling vanishing gradients. It simply activates a neuron if the input x is greater than 0.

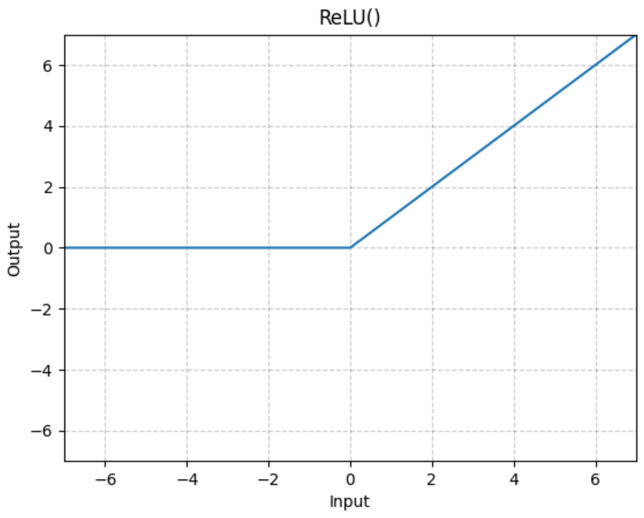


Figure 22: ReLU activation function plot.

$$\text{ReLU}(x) = (x)^+ = \max(0, x)$$

Figure 23: ReLU activation function formula.

- **Leaky ReLU**: A variant of the ReLU function designed to tackle the ”dying ReLU” problem where neurons consistently output zero by allowing negatives numbers. This means that during backpropogation the network is more protected from getting stuck multiplying gradients by 0. Leaky ReLU is probably preferable to ReLU in most situations.

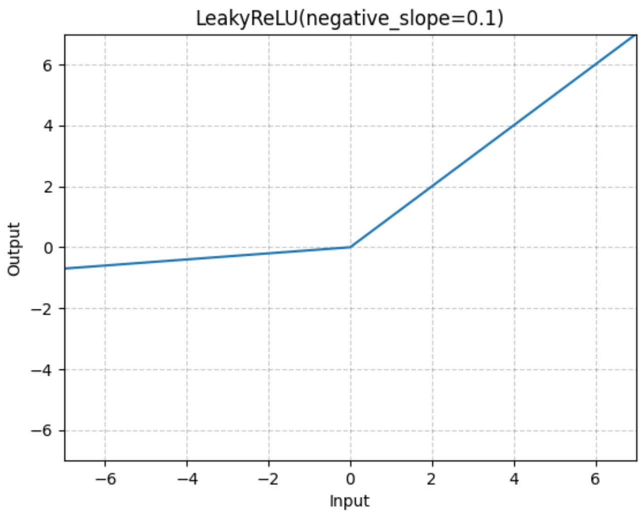


Figure 24: Leaky ReLU Activation Function Plot.

$$\text{LeakyReLU}(x) = \begin{cases} x, & \text{if } x \geq 0 \\ \text{negative\_slope} \times x, & \text{otherwise} \end{cases}$$

Figure 25: Leaky ReLU Activation function formula.

- **Softmax**: Mainly used in the output layer of multi-class classification problems to output a probability distribution over the different classes where the sum of all probability are equal to 1.

$$\text{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

Figure 26: Softmax activation function formula.

4.7.11 Attention

Attention is a mechanism designed for sequence generation. The idea is to allow the model to focus on different and the most important parts of the input sequence similar to how a human might pay attention to certain words in a sentence to gain insight into the context. While attention can be applied to RNN and LSTM models, it was initially introduced and performs best in transformers. Transformers use a method called self attention, that allows the model to process all parts of a sequence simultaneously. The 'self' part is an indication that it relates different parts of a sequence to the whole sequence itself to compute relations in the model. Both the encoder and decoder parts of the transformer can use self attentions.

1. For each token in a sequence, it is transformed into three vectors: a query matrix Q where each row is a query vector for a specific token, a key K matrix where every row is the key vector for a specific value and a value matrix where each row is the value vector for a specific token V. These are calculated through multiplication weights which are parameters of the model.

2. The attention scores are calculated by multiplying the queries by the keys transposed QK^T.

3. Then a scaling factor is applied to the QK^T to help with gradient flow preventing gradient becoming too large, this is usually done by dividing by the square root of the dimensionality of the key vectors $QK^T/\sqrt{d_k}$.

4. Then a Softmax is applied to the scaled scores to normalise them into a probability score. This probability indicates how much focus to put on each token when computing the output.
5. The normalized scores are then multiplied with the value vectors V. This is the weighted sum of the values vectors for each token.

Attention can be expressed like so:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Figure 27: Attention formula.

Attention offers many advantages when being used. Self attention allows for increased parallelization during training which is beneficial for larger sequences and large models. It allows models to handle long range dependencies of the input sequence that were previously impossible. [41]

4.7.12 Rotary Position Embedding (RoPE)

RoPE was a technique introduced to work effectively with the attention mechanism in transformer models. The main idea of RoPE is to encode the positional information of tokens in a sequence by rotating the tokens position in high dimensional vector space using a rotation matrix where elements of the matrix are based on its position in the sequence. There are a number of benefits to using RoPE in transformer models:

- **Flexibility**: It can handle sequences of varying lengths without modification.
- **Token Dependency**: As the relative distance between tokens increases their dependency on each other decays supporting the idea that two token with a long relative distance relationship have less connection.
- **Integration with self attention**: By encoding positional data straight into the embeddings it ensures the self-attention mechanism can consider th relative positions of the input sequence tokens when calculating the attention weights.
- **Computationally cheap**: In comparison to other positional encoding approaches, RoPE is computationally efficient helping greatly with model scalability and performance.

RoPE has been proven to outperform previous positional encoding techniques and offers a powerful and efficient method for capturing relationships between input tokens in a sequence for long range dependencies. [37]

4.8 Generation techniques

4.8.1 Temperature

Temperature is a hyper-parameter in deep learning sequence generation tasks meant to introduce a randomness to the output. The Primary goal of temperature in this project is try to simulate creativity in the output by adding a random choice to which of the potential outputs gets chosen. It does this by applying a positive scalar to logits of the output before it is converted in probability scores summing up to 1. Once the network has chosen the top k probabilities from the output of the model we choose a temperature value between 0 and 1 (no value greater than 1 is used in this project as testing revealed it introduced to much randomness).

- **temperature** $= 1$: No scaling is applied and the output acts as normal.
- **temperature** $= > 1$: Increases the randomness of the output by making less probable outcomes more likely to be chosen.
- **temperature** $= < 1$: decreases the random as you approach 0 making the prediction more deterministic with more probable outcomes more likely to be selected.

This can make temperature an important hyper-parameter when generating out for this project and offers the ability to play around with a more 'creative' output. [39]

4.8.2 Top-k

Top-k sampling is another hyper parameter used to help introduce creativity in sequence generation tasks. It is a simple function that filters out a list of the top k most probable tokens according to the models predictions for the prediction to choose from. A higher k mean more tokens to choose from. When paired with other hyper-parameters such as temperature we can add even more control over the 'creativity' of the models output. [44]

4.8.3 N-gram limitation

N-grams are a sequences of tokens of n length that are sub sequences of out input sequence. One hyper parameter we can play around with in sequence generation is one that limited repeating N-grams. We can essentially tell our model generation that we do not want to repeat any N-gram sequences in the output. If we set $n = 2$ during generation the model will make sure there are no two sub sequences in the overall generation that are the same. Having a lower N-gram limit can make the output very chaotic and unpredictable but finding the right number can encourage creativity and flow and help stop repetition in the output. [28]

5 Design and Implementation

This section will display both the initial design plans and actual implementation of each section.

5.1 Project Plan

5.1.1 Original Timeline

The original timeline for the project was outlined as below:

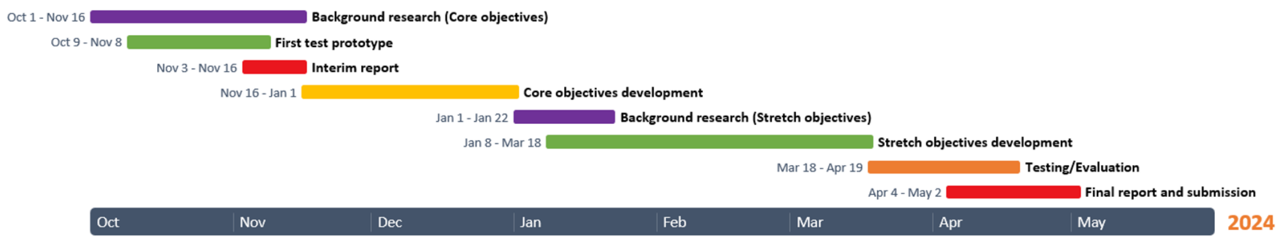


Figure 28: Original Gantt chart project timeline.

A lot of time was allocated for research and development on the stretch objectives. Mainly how to incorporate the multi-instrument functionality.

5.1.2 Actual Timeline

However during development the actual timeline looked more like this:

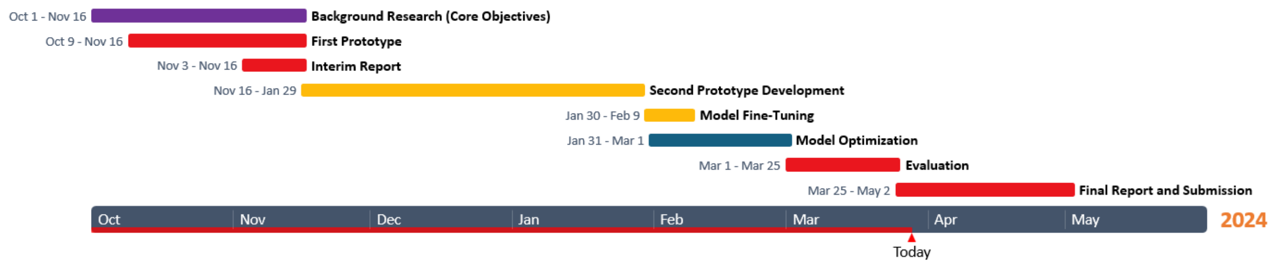


Figure 29: Actual Gantt chart project timeline.

There were a number of reasons for the changes in timeline especially from January onwards. Other responsibilities played there part such as other course works and personal matters. After the second prototype development there was a model that already satisfied the core objectives and most of the stretch objectives. From then on it was a matter of fine tuning and optimizing the model for the best output and sensible training times. A substantial amount of time was allocated to finding ways of speeding up model training as this was a significant bottleneck in the project. Testing and evaluation of the model went smoothly and with better results than expected and helped to outline ways in which the model could be improved in the future with more time and processing power.

5.2 Online example

The architecture and pipeline of this project was established using the help of a notebook tutorial on the MidiTok GitHub page. This tutorial offered a simple demonstration on how to use the MidiTok library with the transformers library and served as template foundation for the rest of development.
The tutorial can be found here:
`https://github.com/Natooz/MidiTok/blob/main/colab-notebooks/Example_HuggingFace_Mistral_Transformer.ipynb`

5.3 Model Functionality

The functionality of the model is fairly simple. It should take samples of music in the form of MIDI files processed and tokenized as input. From there it will generate a continuation of the input that is creative enough as a stand alone piece of music and is good enough that it can't be identified from human made music. The model must be able to output multi-track music to be converted to MIDI files. These files can then be used by anyone and imported into software such as Ableton to be played around with and added to as desired.

5.4 Data Acquisition

Data is one of most important aspects of this project. Luckily there are a good amount of options when it comes to which data to use. It was decided to go with the Lakh MIDI dataset as it is the largest currently available dataset which gave many more options for how the data is used. The Lakh dataset consists of 176, 581 unique MIDI files of which 45, 129 were matched to the Million Song Dataset which is a community led dataset of confirmed popular music tracks. This sub set of 45, 129 files was chosen as the training data for this project for a few reasons:

1. It gives a large amount of good quality data, verified by a trustworthy community led organisation.
2. Data is confirmed to be popular music tracks which is ideal for training a model to output enjoyable music.
3. Data is mostly multi-instrumental as most popular tracks are musical ensembles.

From this dataset specific sub sets can be chosen based on attributes such as total bar counts in the song and track length or use the whole dataset to train the model. [11]

5.5 Pre-processing and Tokenization

5.5.1 Extracting Tracks Design

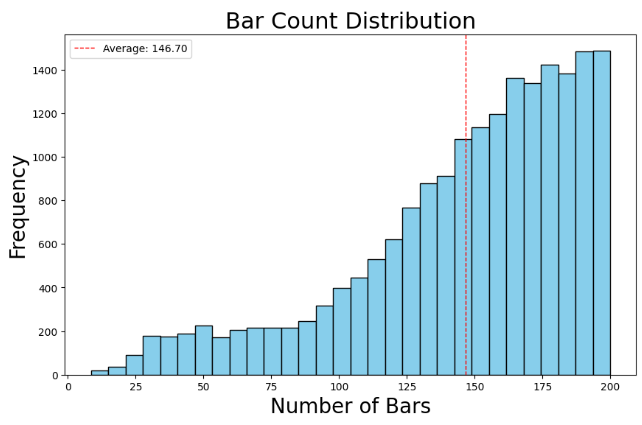
The model is supposed to output music that is similar to popular human made music, because of this the very first step to take will be to extract only the songs that were in a 4/4 time signature. This means that from the 45, 129 MIDI files in the matched set will be further reduced into a slightly smaller sub set of only songs that have a consistent 4/4 time signature. The reason for doing this is:

1. 4/4 time signature is the most commonly used time signature in popular music.
2. Having a uniform time signature throughout the training data means the model should not struggle with beats and bars and allows for greater chance that all generated music is not out of time.
3. The vast majority of the data in the matched set (35, 978 files) are in 4/4 time signature meaning there will be a significant bias in the training towards this time signature anyway. Without using data augmentation or bias mitigation techniques which would be very difficult with MIDI data the bias towards 4/4 would be left unchecked and have a negative impact on training.

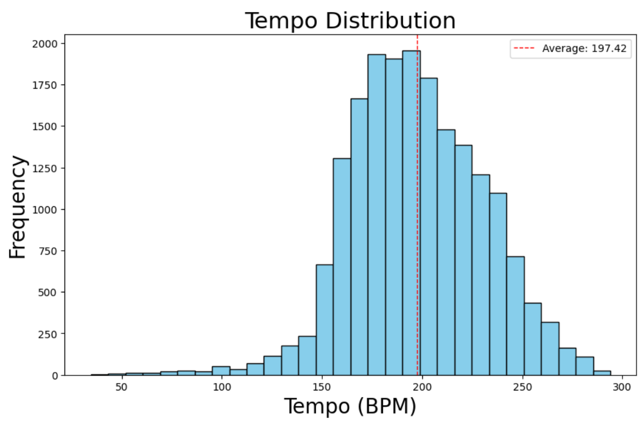
5.5.2 Extracting Tracks Implementation

All 4/4 time signature tracks were extracted from the matched set which left the total number of tracks at 35, 978. Further sub datasets were extracted as well due to training time constraints and initial model training and calibration by only including tracks of a certain number of bars.

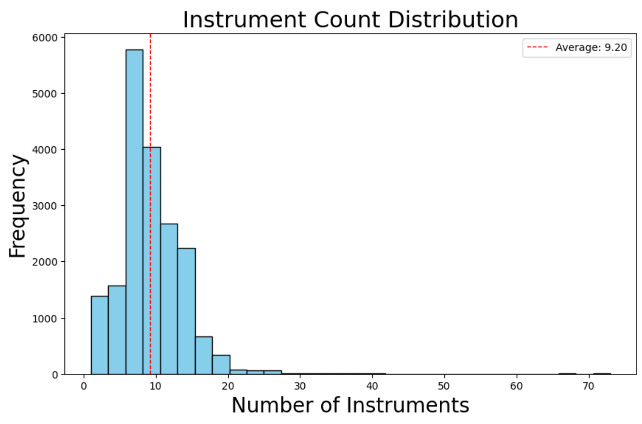
- A smaller dataset of 8, 290 tracks with a maximum bar count of 150. This was used on earlier versions of the model.
- A larger dataset of 18, 939 tracks with a maximum bar count of 200. This was used on the final model as large enough dataset to capture variety but not to large to make training times unfeasible. Shown in figure 30.



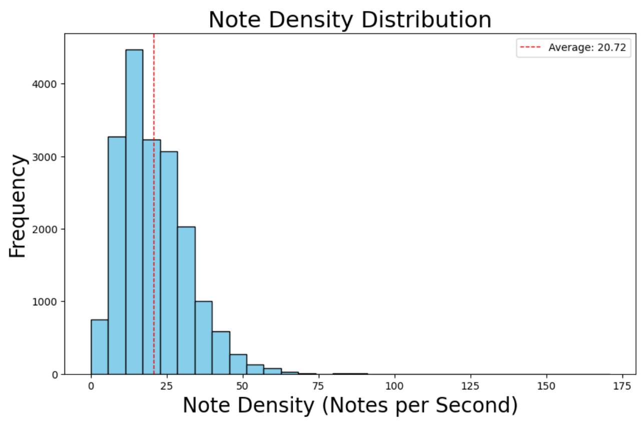
(a) Bar count distribution.



(b) Tempo distribution.



(c) Instrument count.



(d) Note density.

Figure 30: Larger dataset visualisation.

5.5.3 Tokenization Design

Tokenization is important to the project for two main reasons: The first is that the model itself needs a numerical representation of the midi events in order to train on the data. The second is that the final layer output of the model will be the same size as the size of the list of vocabulary tokens which means a smaller vocabulary should train faster as there are less parameters in the final layer.

The MidiTok Python library will be used for tokenization as it has all the most common MIDI tokenization methods available to use. There are two important aspects to consider when choosing which method to use: Firstly the chosen tokenization method must include the ability to tokenize multi-track pieces an retain respective instrument channel data. And secondly it must be in the correct format to to be fed into the chosen GPT model (Mistral 7B).

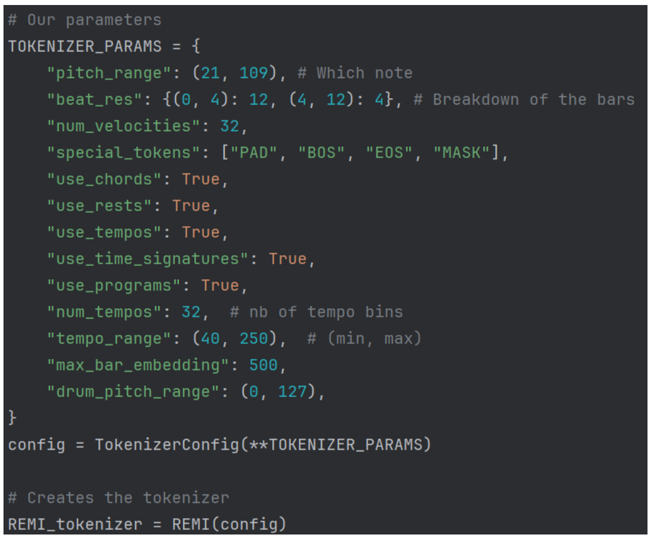
There are two promising potential multi-track methods available:

1. **REMIPlus**: Encodes each event separately into a single continuous sequence including time signature, tempo, bar, position, instrument(program), pitch, duration and velocity.
2. **OctupleMIDI**: Encodes data into 8-tuples including time signature, tempo, bar, position, instrument(program), pitch, duration and velocity.

REMIPlus would be the most appropriate method to use with my chosen GPT model. Mistral 7b expects a single sequence of tokens that a sliding window can slide across to train the model. This is simply a format that OctupleMIDI does not do. RemiPlus will also work much better for other pre-processing steps such as BPE and one-hot encoding which will be employed in this pipeline.

5.5.4 Tokenization Implementation

The MidiTok Library handles REMI and REMIPlus tokenization as the same tokenizer. REMIPlus tokenization was used on the data with a total token vocabulary size of 992 before BPE.

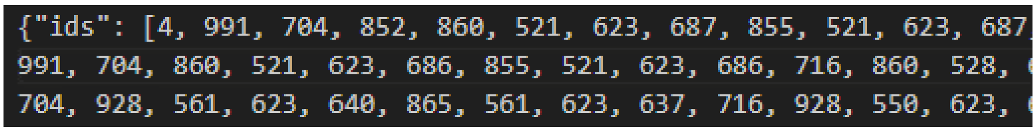


```
# Our parameters
TOKENIZER_PARAMS = {
    "pitch_range": (21, 109), # Which note
    "beat_res": {(0, 4): 12, (4, 12): 4}, # Breakdown of the bars
    "num_velocities": 32,
    "special_tokens": ["PAD", "BOS", "EOS", "MASK"],
    "use_chords": True,
    "use_rests": True,
    "use_tempos": True,
    "use_time_signatures": True,
    "use_programs": True,
    "num_tempos": 32,  # nb of tempo bins
    "tempo_range": (40, 250),  # (min, max)
    "max_bar_embedding": 500,
    "drum_pitch_range": (0, 127),
}
config = TokenizerConfig(**TOKENIZER_PARAMS)

# Creates the tokenizer
REMI_tokenizer = REMI(config)
```

Figure 31: Tokenizer parameters.

The tokenizer is then used to tokenize each MIDI file in the dataset being used. Turning the MIDI events into a continuous sequence of numerical token IDs associated with each MIDI event.



```
{"ids": [4, 991, 704, 852, 860, 521, 623, 687, 855, 521, 623, 687
991, 704, 860, 521, 623, 686, 855, 521, 623, 686, 716, 860, 528,
704, 928, 561, 623, 640, 865, 561, 623, 637, 716, 928, 550, 623,
```

Figure 32: Example snippet of tokenized MIDI track.

5.5.5 BPE and One-hot Design

BPE is a common approach used in NLP tasks. Seen as this project is very similar to language sequence generation BPE will be used here as well. This will effectively help to shorten training sequences down to smaller sub sequences to help with memory management and model training times whilst still capturing all the important relationships between tokens in the sequence. A maximum value will need to be considered on the total size of the vocabulary after BPE has been done as the final output layer of the model will be the same size as the BPE vocabulary. Increasing the BPE vocabulary size can have a huge impact on the amount of trainable parameters in the model and therefore model training times.

Considering there may be a large vocabulary size it may be worth doing one-hot encoding on the inputs as well to avoid potential ranking or bias problems occurring with higher numerical token IDs. However, implementation of one-hot encoding will be down to available system memory and computation time. If the project has spare memory and the model can train in a sensible time then one-hot encoding should be employed otherwise it would be sensible to leave it out so long as the model still produces acceptable output.

5.5.6 BPE Implementation

BPE was implemented on the tokenized data which lead to a good reduction in the size of training data files. Initially setting the max BPE token vocabulary to $25,000$ was too large and caused issues with model training times. Finally a value of 5000 was settled on which allowed the model to train much faster and did not negatively impact performance.

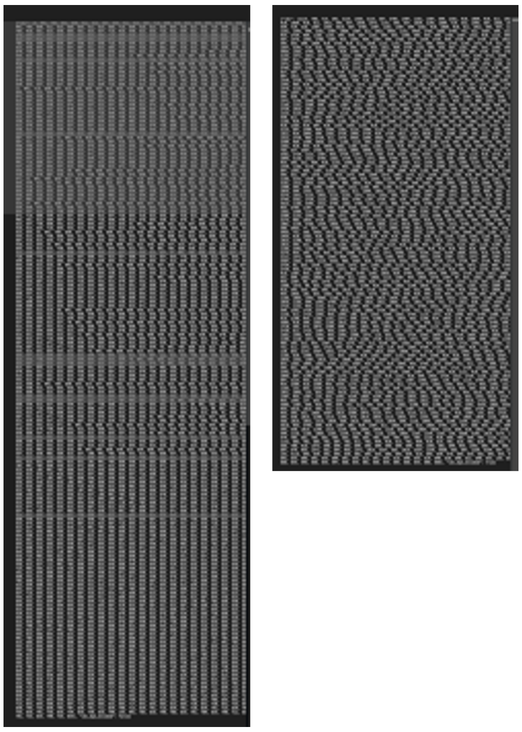


Figure 33: Comparison of non-BPE(left) file size and BPE(right) file size.

One-hot encoding was not used in the project at all due to memory limitation. Overall there was no discernible negative impact of not using one-hot encoding and the model was able to perform well without it.

5.5.7 Data Split Design

The data will need to be split into training, validation and test data to access the accuracy of the model and quality of the generated output:

- **Training set**: This will be used as the input training data for the model.
- **Validation set**: This will be used to access how well the model does on unseen data.
- **Test set**: This will be used to generate example tracks for evaluation.

As there is a quite a large amount of data it would be good to have a smaller ratio on the validation set so that it maximises the training set and still has a decent amount of validation files to test with. The data will be split 90/10 so the training data will consist of ninety percent of the data and the validation set will be ten percent of the data. The test data will consist on a set number of tracks taken from the train data. Probably between 10 to 25 tracks.

5.5.8 Data Split Implementation

The data was split by shuffling the file paths to make sure the files were in random order. The split itself was done on a ratio of 80/10/10.

- **Training set**: 80% of the data was used for training.
- **Validation set**: 10% of the data was used as the validation set.
- **Test set**: 10% of the data was used a test set.

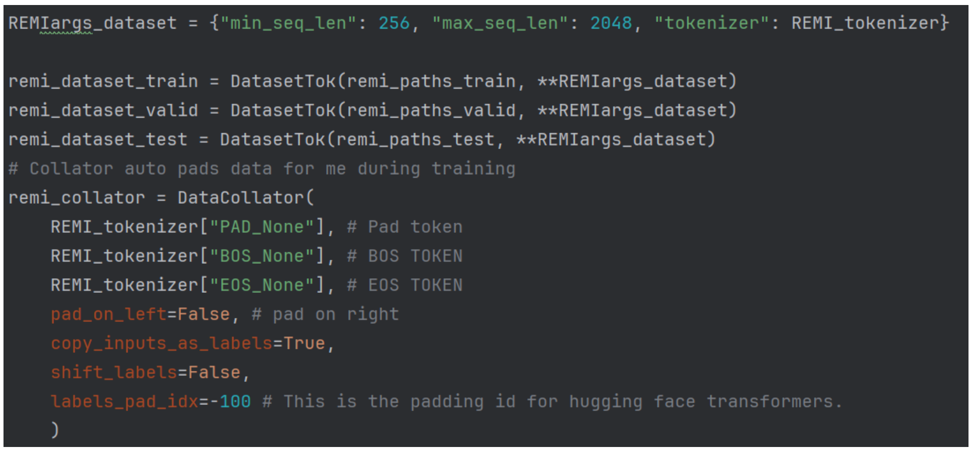
This was a slight change from the plan that enabled the test set to consist of a larger number of files for more diverse testing data whilst not losing out much in the way of training data.

5.5.9 Input padding Design

All input sequences from the training data will need to be made to a uniform length to input into the model. The best approach would be to split inputs into sequences with minimum and maximum token lengths and fill in the values with a padding token (usually 0) to make sure inputs are a uniform length. Tokens will be padded to the right of the sequence so if a track contains a low amount of tokens the padding will fill in with the padding tokens until the end of the maximum token length.

5.5.10 Input Padding Implementation

Sequences were split into sub sequences with a minimum value of 256 and maximum value of 2048 tokens. The sub sequences would act as the individual input sequences for the model to train, validate and test on. If tracks were cut off right at the end and there was a very small sub sequences the tokens would be padded with 0's until the sequence hit the minimum token value. The model is aware of the pad token value so the pad token does not effect that models training.



```
REMIargs_dataset = {"min_seq_len": 256, "max_seq_len": 2048, "tokenizer": REMI_tokenizer}

remi_dataset_train = DatasetTok(remi_paths_train, **REMIargs_dataset)
remi_dataset_valid = DatasetTok(remi_paths_valid, **REMIargs_dataset)
remi_dataset_test = DatasetTok(remi_paths_test, **REMIargs_dataset)
# Collator auto pads data for me during training
remi_collator = DataCollator(
    REMI_tokenizer["PAD_None"], # Pad token
    REMI_tokenizer["BOS_None"], # BOS TOKEN
    REMI_tokenizer["EOS_None"], # EOS TOKEN
    pad_on_left=False, # pad on right
    copy_inputs_as_labels=True,
    shift_labels=False,
    labels_pad_idx=-100 # This is the padding id for hugging face transformers.
    )
```

Figure 34: Code snippet for padding set up and data sub sequencing.

5.6 Hugging Face Library

The Hugging Face transformers library offers an extensive and easy to use deep learning package with lots of ready to go GPT models, trainers and generators to help with model training. This gives access to some of the most recent and successful GPT models developed by reputable companies and integrates very well with MidiTok and PyTorch libraries. These libraries will enable the most of the development pipeline.

5.7 Model Architecture

5.7.1 Mistral 7B

The GPT Mistral 7B model will be used in this pipeline. It can offer sliding window attention that is perfect for REMIPlus tokenization input. This is a decoder only transformer model that features a self attention mechanism and multi layer perceptron in each decoder. A decoder only model for this project can be used because with sequence generation there is no need to use encoders to project input into a higher dimensional vector space that other tasks such as classification or translation might need. The Mistral model itself contains a number of layers with the following roles:

- **Embedding Layer**: Maps the tokenizer vocabulary to a specified vector size, in this case our hidden layer size.
- **Decoder layers**: A specified number of decoders stacked together. Each decoder contains the below components.
- **Attention layers**: Performs the self-attention in the model, splitting the input into query, key and value matrices to learn complex relations between input tokens in the sequence. Finished off by a rotary embedding layer that focuses on encoding relative positions of tokens in the sequence which is important for understanding the order in which the token appears in the sequence.
- **MLP (Multi-layer Perceptron) layer**: Inputs the hidden size from the results of the attention layer and projects it into an intermediate size and then back down to the hidden size and performs a SiLU activation on the output.
- **RMS normalization**: Root Mean Squared normalization is applied to the input before and after the attention layer and again before the final output layer. This applied to help stabilise training and offer superior performance to the model.
- **Final layer**: Maps the hidden state back into the vocabulary size. Each neuron in the final output will represent a probability of that vocabulary token being the next prediction in the sequence.

Mistral by default uses the SiLU activation function which has been shown to outperform other activation functions such as ReLU.

5.7.2 Sliding Window size Design

The input of the model will be a specific size sliding windows that slides across the training data. The size of the window is important as a larger window represents larger sections of the track meaning we can capture important relationships between tokens across larger sections of the track itself. Ideally the sliding window will be as large as possible to capture enough data but there will be a trade off in memory usage. It will be important to explore the possible windows sizes based on the hardware available for this project. Starting from a windows size of 256 seems sensible and increasing from there during prototyping to discover the optimal size.

5.7.3 Sliding Window Implementation

During prototyping a value of 512 was eventually settled on as the sliding window size. As the inputs for training are at maximum 2048 tokens this means the sliding window accounts for around one quarter of the input at a time which is a good size to capture the relationships across that sub section of the track. Increasing the window size beyond this would often lead to memory errors as the system did not contain enough GPU memory.

5.7.4 Hidden and Intermediate Layer Sizes Design

Again, these values will be subject to trial and error based on hard limitations but intuitive decisions can be made early on during prototyping to find sensible values. Specifically in the Mistral 7B model these refer to the size of the hidden layer on input and output of the model (hidden size) and the size of the hidden layers during the multi-layer perceptron stage of the decoder that projects the hidden input size into a larger sized layer to allow the model to learn more complex behaviour.

5.7.5 Hidden and Intermediate Layer Sizes Implementation

The hidden layer size was set to 512 to match the sliding window size, and the intermediate layer size was set to 1024 so that when the hidden layer was projected to the intermediate layer it would have double the neurons to help learn more complex relationships. This configuration of neurons offered the best balance of performance and training times.

5.7.6 Number of Decoders Design

A sensible starting value for the number of decoder layers would be somewhere between 6 and 10. The default value for the model is 32, however this is under the assumption that the model will be used with vocabulary sizes in the tens of thousands like in NLP tasks with the English language. This project that deals with MIDI inputs as its 'language' will have a considerably smaller vocabulary and therefore should not need as complex a decoder stack as 32 layers. Adding more layers can drastically enlarge the amount of trainable parameters the model has so keeping the value as small as possible without affecting the models output will be crucial.

5.7.7 Number of Decoders Implementation

The model used employed 8 decoder layers in its stack which was the middle value of the initial design thought. 8 decoder layers proved to be a sensible decision as it kept the number of training parameters to a minimal value to help with training times and still captured the complexity in the tracks training data.

5.7.8 Attention Heads and Key-Value Heads Design

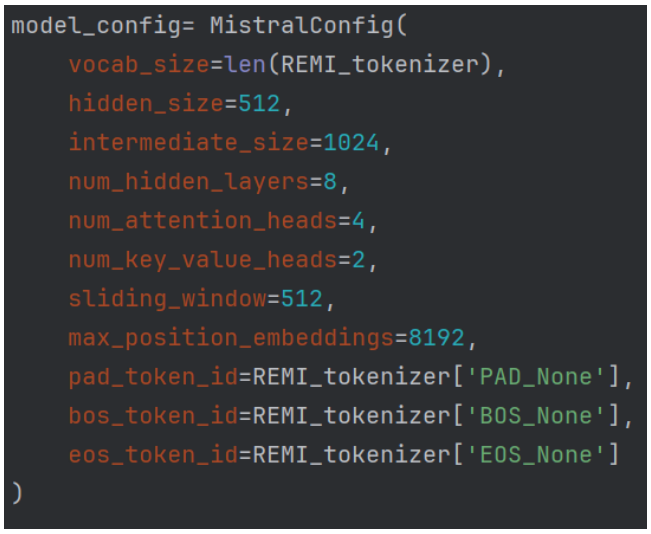
These hyper-parameters directly influence how effective the attention mechanism are. The number of attention heads allows the simultaneous focus on different parts of the input sequence so having a higher number can be beneficial to the model but have drastic effects on computational complexity, memory usage and training times. Key-Value heads are related to the Grouped Query Attention (GQA) in the model which groups attention heads into key-value pairs. With the Mistral 7B model it can either employ GQA, Multi-Head attention (MHA) or Multi-Query Attention (MQA). Each has there own benefits and disadvantages:

- **MHA**: Is the most flexible and capable of capturing a wider range of dependencies at the cost of efficiency.
- **MQA**: The most simple of the three, focusing on a single query per attention head reducing complexity at the cost of performance.
- **GQA**: A mix between the two by grouping into key-value pairs can offer a balance between performance and efficiency.

GQA seems like the most sensible approach in this situation where there can be a balance between performance and efficiency. To achieve this in the Mistral 7B model the key-value heads parameter simple needs to be set to less than the number of attention heads but more than 1.

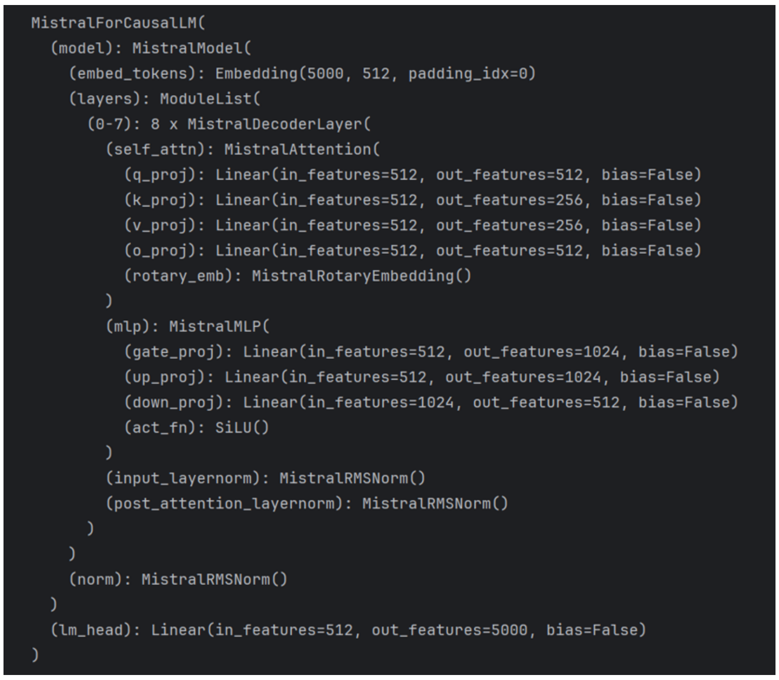
5.7.9 Attention Heads and Key-Value Heads Implementation

The number of attention heads was set to 4 and the key-value head was set to 2 for this model. This allowed the model to use GQA as a good balance between performance and efficiency. This is half the values that were used in the online example that helped with this project to help save memory and training time but still still enabled to attention mechanism to do a good job at focusing on important parts of the sequence.



```
model_config= MistralConfig(
    vocab_size=len(REMI_tokenizer),
    hidden_size=512,
    intermediate_size=1024,
    num_hidden_layers=8,
    num_attention_heads=4,
    num_key_value_heads=2,
    sliding_window=512,
    max_position_embeddings=8192,
    pad_token_id=REMI_tokenizer['PAD_None'],
    bos_token_id=REMI_tokenizer['BOS_None'],
    eos_token_id=REMI_tokenizer['EOS_None']
)
```

Figure 35: Code snippet for Mistral 7B hyper-parameters.



```
MistralForCausalLM(
  (model): MistralModel(
    (embed_tokens): Embedding(5000, 512, padding_idx=0)
    (layers): ModuleList(
      (0-7): 8 x MistralDecoderLayer(
        (self_attn): MistralAttention(
          (q_proj): Linear(in_features=512, out_features=512, bias=False)
          (k_proj): Linear(in_features=512, out_features=256, bias=False)
          (v_proj): Linear(in_features=512, out_features=256, bias=False)
          (o_proj): Linear(in_features=512, out_features=512, bias=False)
          (rotary_emb): MistralRotaryEmbedding()
        )
        (mlp): MistralMLP(
          (gate_proj): Linear(in_features=512, out_features=1024, bias=False)
          (up_proj): Linear(in_features=512, out_features=1024, bias=False)
          (down_proj): Linear(in_features=1024, out_features=512, bias=False)
          (act_fn): SiLU()
        )
        (input_layernorm): MistralRMSNorm()
        (post_attention_layernorm): MistralRMSNorm()
      )
    )
    (norm): MistralRMSNorm()
  )
  (lm_head): Linear(in_features=512, out_features=5000, bias=False)
)
```

Figure 36: Overview of Mistral 7B model used.

5.8 Training

There are various hyper-parameters and processes crucial for an effective training cycle for this project. The Hugging Face trainer library offers a great set of configuration classes and documentation to help streamline the process and take advantage of the different deep learning techniques available.

5.8.1 Checkpoints Design

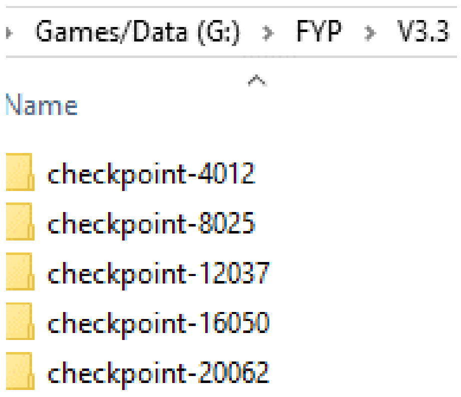
Working with complex models often means long training times. In order to make sure that time and power are not wasted during training models can be saved locally at specified points in the training cycle. The saved files can contain all the learned model parameters from training so far and be loaded in when needed. This is beneficial for a few reasons:

1. If for any reason training is interrupted by power cuts or connection errors there is no need to start training from scratch again. Simple load in the latest saved model checkpoint and carry on.
2. If you want to split the training over different time days or fine-tune the model on a larger set of training data you can pause the training after a specified number of epochs or steps and continue on with the larger dataset.
3. If a model earlier in the training cycled performed better or your model started over-fitting or under-fitting during training we can go back to an earlier point that performed better.

As this project is expected to have very large run times the model will be saved every epoch. Files will be saved locally and backed up to cloud storage and an external storage device on a regular basis.

5.8.2 Checkpoints Implementation

During training, the model was saved locally after every successful training epoch as well as the training logs which held important data from training such as learning rate changes and loss rates. This was especially useful as the latest epoch of training turned out not to be the best model in terms of loss on the testing set. So I could load in one of the previous iterations of the model that performed better on unseen data. It also meant that if for any reason I had to stop the training during one of the training epochs I did not lose much progress, especially in the earlier versions of the model that had training times of around four and a half hours per epoch. Model checkpoints were also saves to an external device.


Games/Data (G:) > FYP > V3.3

Name

checkpoint-4012
checkpoint-8025
checkpoint-12037
checkpoint-16050
checkpoint-20062

Figure 37: Saved checkpoints in the local folder.

5.8.3 Early Stopping Design

Early stopping will be used as an attempt to cut training times as much as possible. Throughout the training the loss on the validation set will be monitored for a specific number of epochs and if no improvement in the loss is recording for that amount of epochs training will stop and the saved model with the best validation loss will be used.

5.8.4 Early Stopping Implementation

Early stopping was introduced during training with a patience of 2 epochs. This means that if the model does not improve on the evaluation loss in two epochs the model will stop training. This was a sensible number as the number of epochs used in training turned out to be relatively small (around 12-15 epochs) as the amount of training data was quite high so the model managed to converge on its optimal quicker. Using this method I was able to stop the model as it started over-fitting.

5.8.5 Batches Design

Using batches is important for memory usage. Larger batch sizes will use up more memory but enable the model to train on more batches in parallel speeding up training times. As there will be a limitation on the available memory due to working locally memory will need to be prioritized over speed in this case. The expectation is a low batch size between size 2 and 10 but some experimentation might be needed. Gradient accumulation steps can also be utilized to help with the memory usage if needed. Gradient accumulation allows us to split the batch size into smaller batches that are processed and gradients are calculated but the weights are not updated until we reach the batch limit which is the product of the batch size and gradient accumulation steps.

5.8.6 Batches Implementation

After experimenting with different configurations to optimize for memory constraints, the following setup was adopted for training and evaluation:

- **Batch Size** Set at 8 for both training and evaluation phases. This means each batch processes 8 input sequences at a time.
- **Gradient accumulation steps**: Configured to 16. This approach allows the simulation of larger effective batch sizes by accumulating gradients over multiple batches before updating the model's weights. Specifically, weights are updated only after processing $8 * 16$ (128) input sequences.

This allowed the local environment to process larger batches as if it had much more memory without the need for much larger memory availability.

5.8.7 Learning rate Design

It's important to explore a variety of learning rates throughout training in this tasks as there is potentially a very large and complex loss gradient landscape. Using the Hugging Face Trainer class certain hyper-parameters to do with learning rate and learning rate decay can be set to help explore that learning rate values. Starting with a fairly high value at the start of training and slowly decaying as the training runs seems sensible, as does including a warm up portion at the start of training.

5.8.8 Learning Rate Implementation

Two different training cycles were conducted using a slightly different schedulers in each during the training of the model. For the initial training from scratch:

- **Learning rate**: set to $1e-3$ (0.001). This fairly high initial learning rate encourages fast adjustments towards the global minima. It is effective when combined with learning rate decay, which gradually reduces the learning rate as training progresses, allowing for finer adjustments in later stages.
- **Learning rate decay**: A cosine decay schedule is specified, meaning that the learning rate will decrease following a cosine curve over the course of training.
- **Warm up**: Set to 0.3 indicating that for the first 30 percent of the training, there will be a linear increase in the learning rate up to the initial value before the cosine decay takes effect. This warm-up phase is designed to stabilize the training early on.

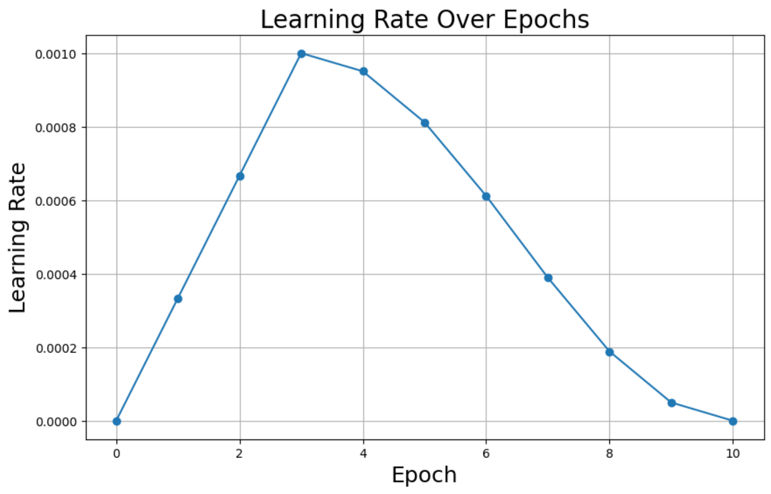


Figure 38: Learning rate of the first training cycle.

During the second cycle the aim was to minimize the evaluation loss as much as possible so a slightly different was approach was used:

- **Learning rate**: Set back up to $1e-3$ to mimic a restart in the cosine curve. This is intended to help the model jump out of any local minima it may have found itself in during the initial cycle.
- **Learning rate decay**: The learning rate is again set to decay in a cosine fashion, continuing the strategy of reducing the learning rate in a cosine curve.
- **Warm up**: No warm-up period was used, as the model was beyond its initial stages of training. The rationale is that the model parameters are already significantly optimized, and the primary focus is on refining and adjusting these parameters rather than the stabilization that was needed at the start.

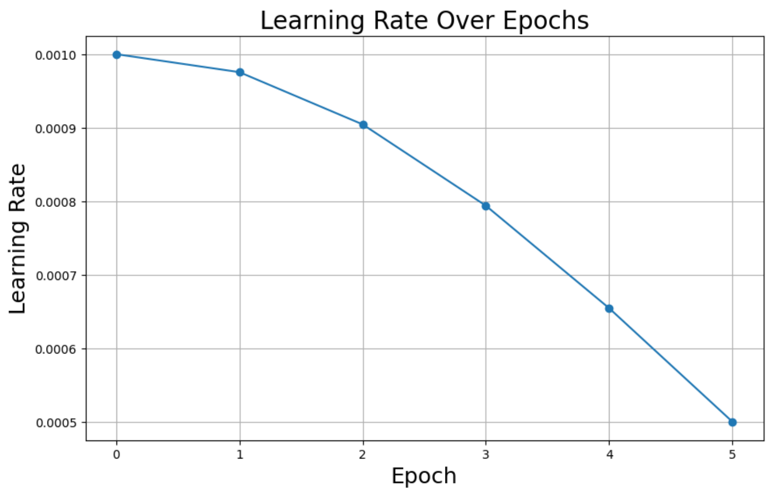


Figure 39: Learning rate of the second training cycle.

5.8.9 Optimizer

There are only two real contenders for the correct optimizer to use for this task:

- **SGD + Momentum**: Can converge faster but may require a more extensive learning rate schedule and has the potential to overshoot global minima.
- **Adam**: An all rounder, Adam performs better in complex gradient spaces and can potentially converge on the global minima more accurately.

Adam will be the choice of optimizer for this task as it offers benefits that align well with this task and model and the loss gradient space is expected to be fairly complex. SGD + Momentum while potentially faster is more likely to struggle in complex gradient space and fail converge on the global minima.

5.8.10 First Train Cycle

Both cycles were trained on a subset of the 4/4 time signature data that totalled $18,939$ tracks with a data split of 90/10/10. This meant there was 15027 training tracks, 1878 validation tracks and 1878 testing tracks.

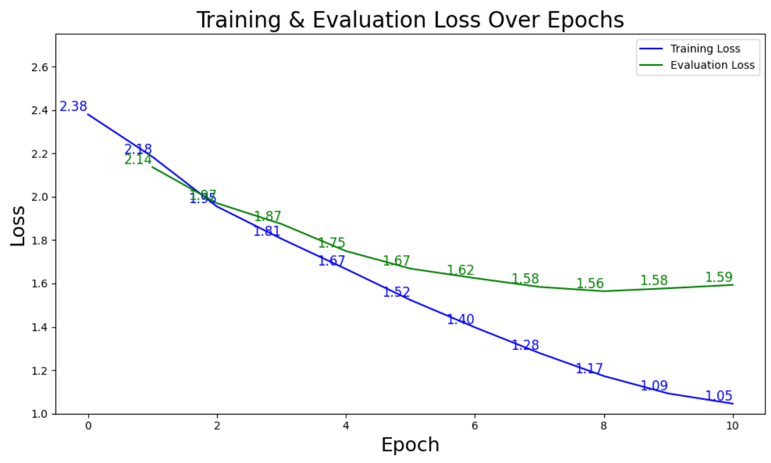


Figure 40: Training and validation loss in first training cycle.

We can see from figure 40 that although the model was initialized from scratch the loss on both the training and evaluation data started quite low. There are a few potential reasons for this:

- **Effective Initialization**: The Mistral model uses a truncated normal distribution as its default method of weight initialisation(mean of 0, standard deviation of 0.02). This gives the model a nice balanced initialization to begin with potentially leading to lower initial loss then if the initialisation was random.

- **Data Quality**: Well prepared data could help the model to output reasonable prediction very early on.
- **Luck**: Possibly it was just very lucky that the initial weights of the model landed in a decent spot in the loss gradient space.

We can also see from figure 40 That the model was starting to over-fit quite a lot as the training loss was dropping but the evaluation loss started to increase.

5.8.11 Second Training Cycle

While being conscious of over-fitting the main concern for this task was minimizing the evaluation loss because as long as the model can output good quality tracks from unseen data it's performing its function well. The learning rate was reset to 0.001 as shown in the Learning Rate Implementation section in the hope that the high rate would help the model exit any local minima it may have found itself in.

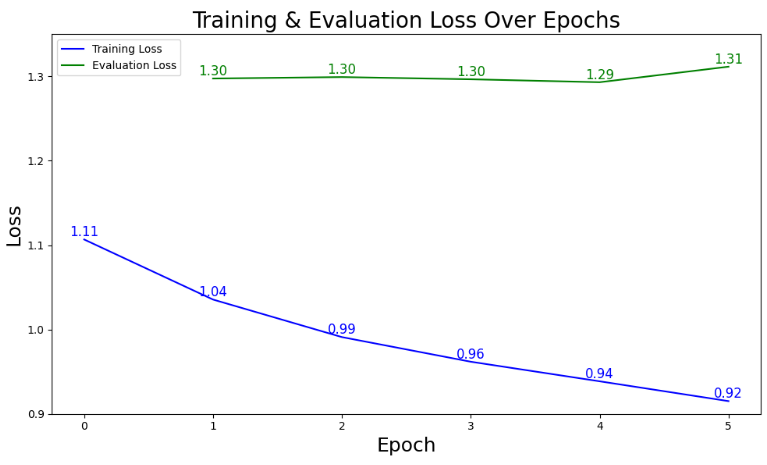


Figure 41: Training and validation loss in second training cycle.

As figure 41 shows the evaluation loss did eventually drop to 1.29 which is the lowest value achieved on the project and thanks for model checkpoints was the model that was loaded in a used for the final model. There were a number of reasons why the training was ended here:

1. **Evaluation loss stagnation**: The reset in learning rate worked well for the initial epoch but the evaluation loss was no longer improving at a sensible rate.
2. **Over-fitting**: The model was just continuing to over fit to the training data with visible signs that unseen data was not performing as well.
3. **Time constraints**: Training the model was taking significant time and energy to run. Each epoch of this model was roughly 2 hours or greater to train.
4. **Good results**: The model was already outputting tracks that satisfied the core and stretch objectives.

In summary, training the model went well with the evaluation loss decreasing throughout training but with a real risk of over-fitting. However, the training was finalized and ended after 15 epochs for a number of reasons but the final model employed gave good results for the task.

5.9 Generation

A variety of different generation hyper-parameters will be explored in the tasks and will be evaluated in the testing phase. The different parameters of interest are:

- **Max and min token length**: This can force the generation output to generate tracks of a certain length.
- **Temperature**: This controls the chance of other tokens that aren't the most probable being chosen.
- **Top-k**: The number of tokens presented for the model to chosen from, taken from the k most probable tokens.

- **Repeat N-gram size**: This give the option to force the model to generate different sub sequences of N size.

With these hyper-parameters we can explore different levels of 'creativity' in the models output. With the maximum and minimum token length determining how long the track is we want to make sure generated music is not too repetitive or too chaotic. This will be further explained in section 6.

5.10 Issues

5.10.1 Hardware, Memory and Training times

By far the most limiting factor in this project was access to powerful hardware that could parallel process quickly and have a substantial amount of memory to store data whilst training. Initially the model being used was taking roughly four and half hours to train each epoch on the local GPU used (NVIDEA 2070, 8GB VRAM). This was for obvious reasons not feasible as each prototype developed would take the better part of a week to train and cost a lot of electricity as the the project was being developed on a personal computer. This presented a real problem that meant the model would have to be limited in terms of complexity and would need to minimise the amount of trainable parameters, the sequence length of inputs, batch sizes and the sliding window size to save not only training time but also memory allocation.

Options were explored to solve this issue such as:

- **Paid services**: Google Colab premium or other service such as RunPod offer a paid service that gives access to a certain amount of compute power on some of the best available machine learning GPU s available. However this was not financially viable.
- **University of Sussex lab facilities**: The lab machines in the University of Sussex's Chichester building were tried as they offered a more powerful GPU then the one used on the personal computer with faster processors and 16GB VRAM. This unfortunately still took around three hours per epoch on the same model and meant waiting around in the labs for a substantial amount of time which was not feasible due to other commitments and responsibilities.
- **University of Sussex remote facilities**: Sussex does have specialised remote machines that can be accessed specifically for machine learning tasks that have much more powerful components. These are usually reserved for PhD students and this project was not granted access to them.

The only option left to try was to see if the project pipeline could be optimised in anyway to cut training times down as much as possible so it could run sensibly on the personal machine. This resulting in optimisation in the following areas:

- **One-hot encoding**: Stop using One-hot encoding during pre-processing. This saved memory without negatively affecting the model.
- **Batch gradient accumulation**: Take advantage of the gradient accumulation step to effectively simulate the use of larger batches
- **BPE vocabulary size**: Reduce the maximum size of the BPE vocabulary from $25,000$ to 5000 meaning the embedding layer and output layer on the model were significantly smaller.

Changing the BPE vocabulary size was by far the most effective optimisation as it cut the number of trainable parameters in the model down from $44,483,072$ to $24,003,072$ without negatively effecting the models output. After these revisions the model training was now taking approximately one hour per epoch, enabling overnight training cycles and a much more practical use of the personal machine.

5.10.2 Library Issues

This project uses various Python libraries to achieve its objectives. A common issue that was encountered were libraries not working as intended because of bugs or not working well with other libraries. This was mitigated by ensuring correct versions of libraries were installed that were known to work and by manually going into library source code and debugging parts of the code that were causing issue. This was done in the Hugging Face Trainer class that had a bug that made it impossible to store a model checkpoint in a folder that already contained one.

5.10.3 Repetitive Tracks

During generation it was observed that tracks would start to repeat the same one or two bars over and over again and limited the creativity and authenticity of the output. This was an issue because it was one of the obvious factors that made the track seem AI made rather than human made which was important for the projects objectives. This was able to be largely sorted by the use of different generation techniques that forced creativity and punished repetitive sequences such as temperature, top-k and no repeated n-grams of a certain size but still happens in some of the generated tracks.

5.10.4 Inconsistent Output

Not all created tracks were usable. Some common issues with output included:

- **Empty tracks**: Tracks with two or three minutes of silence with or without the occasional random note being played.
- **Too repetitive**: Tracks that failed to be creative and got stuck in loops or did not represent section changes.
- **Mistakes**: Tracks that featured a number of musical mistakes such as out of key notes.

These issues are still present in the project but were lessened when experimenting with the generation settings.

6 Testing/Evaluation

As this was a project centered around the generation of music, there was little quantitative evaluation that could be achieved as like other art forms, music is a very subjective thing and can resonate very differently depending on the individual listening. The aim of this evaluation was to perform qualitative assessment to confirm that the model has the ability to reach its core and stretch objectives. The evaluation was done in four main sections:

1. **Introduction**: Some simple questions to gauge the participants level of experience with music.
2. **Turing test**: To gauge if the participants can tell the difference between AI generated or human generated tracks.
3. **Blind model rating with Turing test**: To ask participants to rate AI tracks without being told if it is AI or not and to ask if they thought it was AI.
4. **Model rating**: Ask the participant to rate AI tracks knowing that they are AI.

There were a total of 10 participants that took the evaluation survey.

6.1 Track preparation

All tracks used in this evaluation were either generated by the model developed in this project or were taken from the Lakh MIDI dataset as examples of human made music. To demonstrate the applicability of all these tracks being loaded into third party software, all tracks used were opened in Ableton Live and allocated the instruments that were specified in the MIDI tracks data. No further edits were made to the tracks. All evaluation tracks were treated the same to avoid a bias in the evaluation data. Evaluation was done in person.

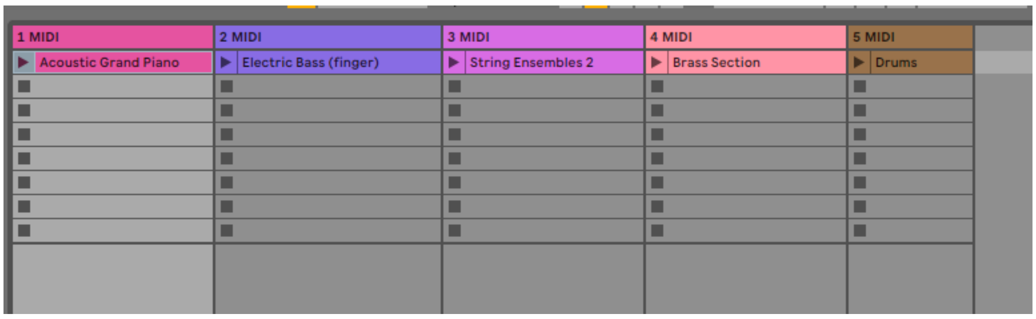


Figure 42: Evaluation track in Ableton with instrument label data visible.

6.2 Participant Musical Experience

The first part of the evaluation was to gauge whether or not participants had any musical experience outside of simply consuming music. The goal was to see if people with more experience were more likely to tell if a track was AI generated or not. Participants were asked a simple yes or no question a to whether they any experience as a musician or musical producer and then how they would rate themselves from beginner, intermediate, advances and expert levels. However none of the participants had any experience beyond a beginner level of playing an instrument or producing music so only minor insights can be found here.

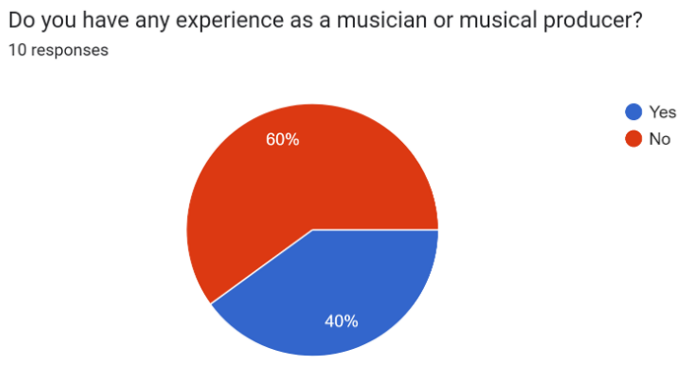


Figure 43: Response for participants with musical experience.

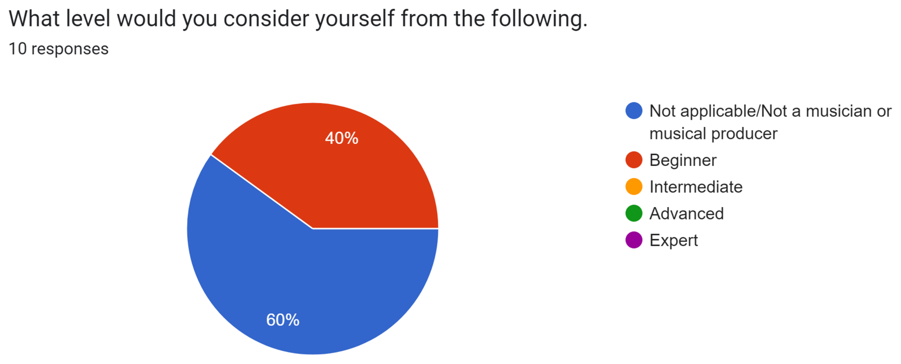


Figure 44: Response asking participants to rate their experience.

The obvious insight from figure 43 and 44 shows that only 40 percent of participants had previous experience with music beyond a consumer level and that all of them would only consider themselves a beginner.

6.3 AI vs Human Turing Test

The idea for this test is simple, the participant is played two separate tracks known as 'A' and 'B' and has to say which they think was an AI generated track. They could pick between five options: 'A', 'B', 'Both A and B', 'Neither A or B' and 'Don't know'. This was helpful in gauging if the participants could tell what was AI when presented with both AI and human made tracks.

In figure 46 we can see the total counts for each answer for the ten participants with the green bar (or green arrow) indicating the correct answer.

We can see that in no instance does more than half the participants pick the correct answer. The lowest scores came from the sets that contained either both AI or both human tracks, perhaps because the two were more similar in sound than when it was one or the other. We can see that set 3 contained the highest score for 'B' being AI generated with a score of 7, meaning participants were more convinced in this instance that a human made track could be AI generated. Participants in general picked the right track more often when they presented with one human and one AI track, but when presented with two AI tracks there were still many incorrect answers.

This is a good indication that the model is capable of generating tracks that listeners can believe is human made, and while there is some indication that the difference is easier to spot when one AI track and one human track were played together there is still sufficient response to conclude the model can output music that would pass for human made music. Figure 45 Does indicate that the more experience one has with music the more likely they are to know the difference between the model and human output, with those answering 'yes' having an average of 30% correct answers and those who answers 'no' slightly lower on 26.6% but more data would be needed from participants with higher levels of experience.

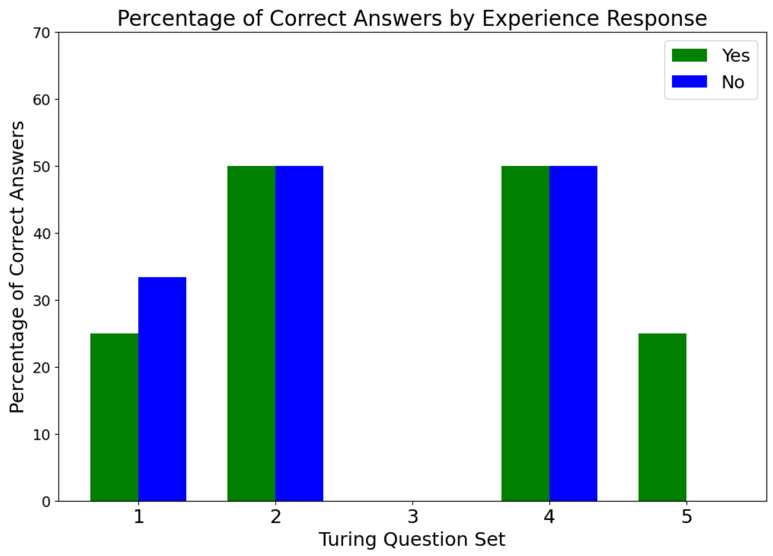


Figure 45: Percentage of correct answers for each set based on whether participant selected 'Yes' or 'No' to having experience beyond consumer level.

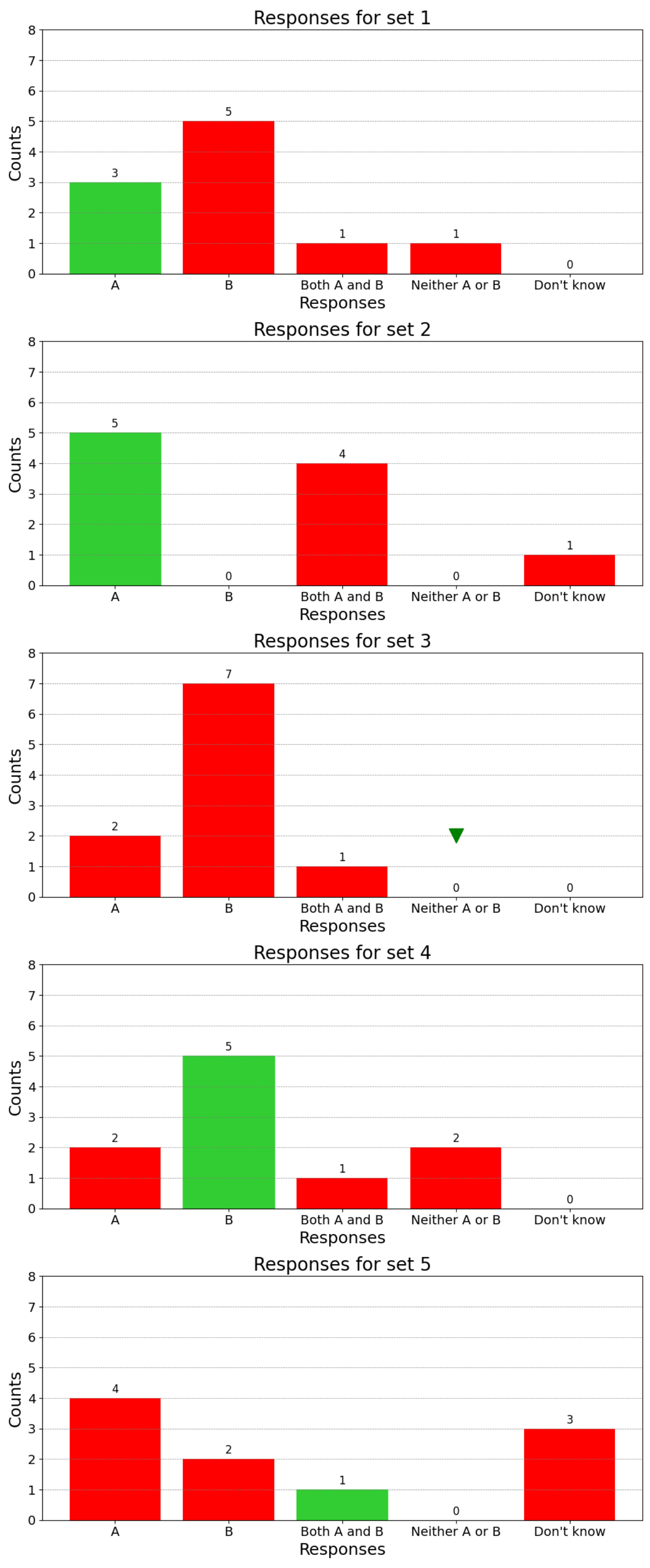


Figure 46: Turing test responses. Green indicates the correct answer.

6.4 Music Rating with Turing Test

This evaluation was a chance to get some feedback on how the different hyper-parameters in the generation part of the model affects how the music is perceived. There were 12 tracks with different generation hyper-parameters with two tracks displaying the same values for a balanced approach. Participants were asked to rate the tracks between 1 and 5 based on whether they enjoyed the piece or whether they thought there might be a real world application for the music such as background music for advertisement or a game. They were also asked if they could indicate if they thought the track was AI generated or not as another extension to the Turing test (participants did not know all the tracks were AI generated).

The twelve tracks would each leverage a different set of values for the temperature, top-k and no repeat n-gram size values in the generation of the track to simulate varying levels of 'freedom' or 'creativity' in the output. See section 5.9 for a more detailed explanation of the different parameters. The following list outlines the different parameters in each track in the following order
(**Track number**: (temperature value, top-k value, no repeat n-gram value) short explanation):

- **Track 1**: (0.7, 12, 4) Temperature and top-k set to a high value with low n-gram limitation to encourage high creativity and low repetition.
- **Track 2**: (0.7, 12, 10) Temperature and top-k set to a high value with high n-gram limitation to encourage high creativity and high repetition.
- **Track 3**: (0.3, 8, 4) Temperature and top-k set to a low value with low n-gram limitation to encourage low creativity and low repetition.
- **Track 4**: (0.3, 8, 8) Temperature and top-k set to a low value with low n-gram limitation to encourage low creativity and high repetition.
- **Track 5**: (0.5, 6, 6) Temperature and top-k set to a average value with average n-gram limitation to encourage average creativity and average repetition.
- **Track 6**: (0.5, 6, 6) Temperature and top-k set to a average value with average n-gram limitation to encourage average creativity and average repetition.
- **Track 7**: (0.9, 30, 12) Temperature and top-k set to a very high value with a very high n-gram limitation to encourage very high creativity and very high repetition.
- **Track 8**: (0.9, 30, 2) Temperature and top-k set to a very high value with a very low n-gram limitation to encourage very high creativity and very low repetition.
- **Track 9**: (0.1, 5, 2) Temperature and top-k set to a very low value with a very low n-gram limitation to encourage very low creativity and very low repetition.
- **Track 10**: (0.1, 5, 10) Temperature and top-k set to a very low value with a very high n-gram limitation to encourage very low creativity and very high repetition.

Figure 47 shows that track 6 with the balanced approach for generation parameters was the most successful track scoring positively across the board and only 1 person believing it was AI, however track 5 that had the same parameters did not do as well. Other tracks such as track 10 did well on the scores but the majority of people did believe it was AI. The worst performing track was track 7 that encouraged very high creativity and very high repetition where every participant guessed it was AI and a majority gave it low scores most likely due to it sounding too random. In general a balanced generation seems to be the right approach to generate music that people enjoyed or recognise to have a real world application and pass a Turing test.

This test displays the models ability to reach core and stretch objectives such as well structured tracks that are hard to distinguish between model generated and human made tracks when given the right values for hyper-parameters.

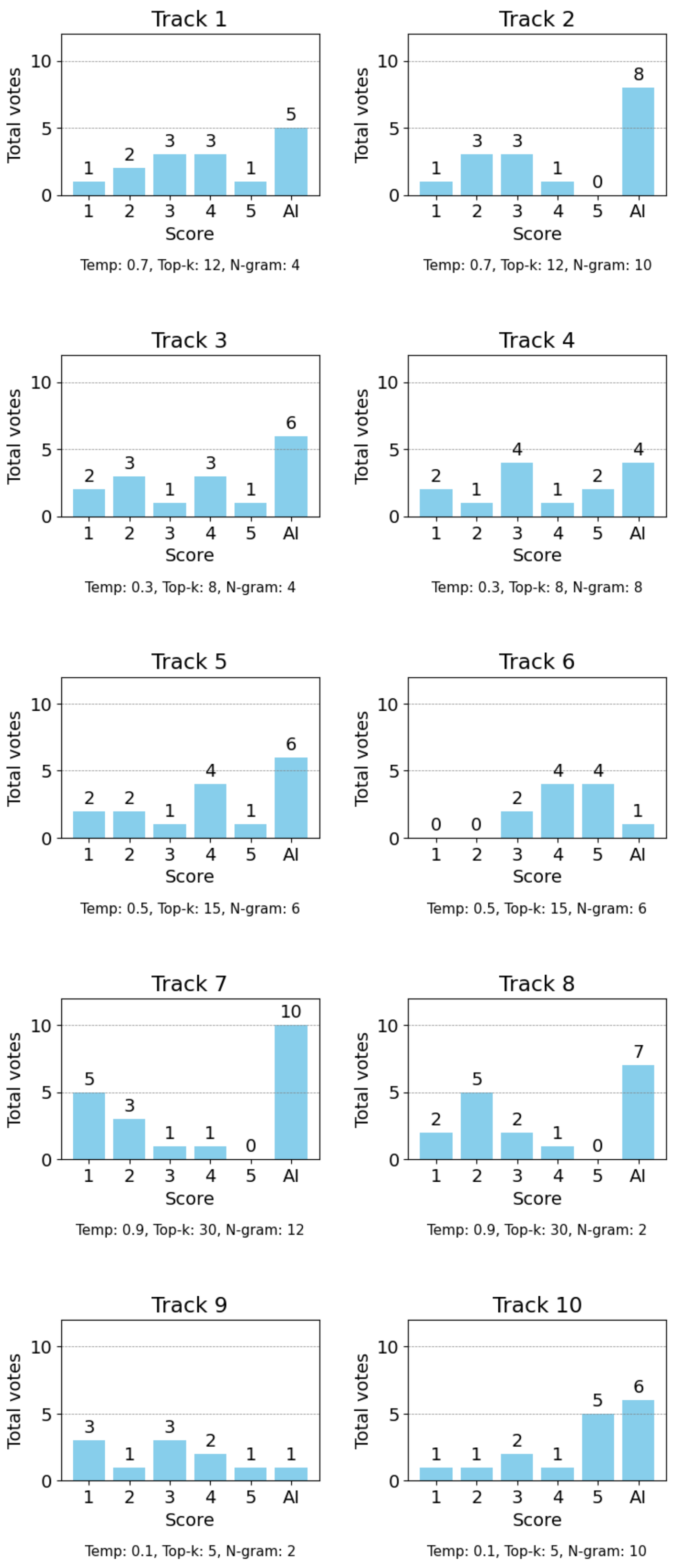


Figure 47: Music rating with Turing test responses.

6.5 Final Model Rating Test

For the final test the participants were told that the tracks they were about to hear were all generated by this project's model. They were presented with 3 tracks all generated with the same hyper-parameters as track 6 from section 6.4 and asked to rate them again based on the same criteria as before.

Figure 48 shows that in general each track was scored favourably with an average score above 3 with track 3 scoring very well meaning participants could see the track having a real world application such as background music for advertise meant or theme tunes further demonstrating the models ability to produce music that satisfies the core and stretch objectives.

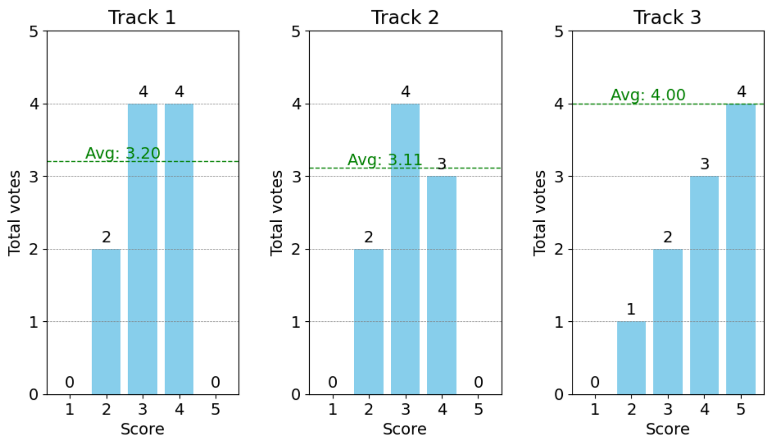


Figure 48: Final music rating test responses with averages.

6.6 Evaluation Conclusion

Overall the evaluation showed that the model was capable of generated tracks that satisfied core and stretch objectives. There was a slight indication that the more experienced a participant is as a musician or musical producer the more likely they are to tell between AI and human generated tracks, but more participants with a higher experience level would be needed to test further. The model was able to convince participants in many cases that its generated music could be human made with both Turing tests providing sufficient evidence of this. Generation hyper-parameters played a huge role in how participants rated the tracks and how obvious it was that the track was AI made, but displayed a good indication of what hyper-parameters should be used for generation. The model was able to generate tracks that on average participants felt had a real world application and found enjoyable to listen to.
Throughout the evaluation participants were asked to comment on why they thought certain tracks were AI generated with the most common responses being:

- Tracks seemed too repetitive.
- Some tracks felt off-beat and did not flow well.
- Some tracks felt too chaotic.
- Tracks did not feel natural.

It is important to note that these responses also included the first test where participants would give these answers about a track that was human made as well.

7 Conclusion

7.1 What Went Well

Overall the project timeline went well with the projects objectives being realised in plenty of time and with enough time to conduct a decent testing and evaluation stage to display the models capabilities in a qualitative and somewhat quantitative way.

Background research was time consuming but greatly developed the knowledge base needed for not only this task but for machine learning and deep learning projects as a whole. The project lined up well with other modules undertaken and there was a cross over of knowledge and experience which could be applied to many both this project and others.

Data acquisition and pre-processing was straight forward and easy thanks to the extensive background research and availability of datasets and python libraries.

The model deployed was effective despite limitations introduced by hardware and model complexity, and after an extensive optimisation search training times were brought down to a manageable level.

An extensive and effective testing and evaluation section was carried out that displayed the models capabilities as well as thoroughly testing the effects of generation hyper-parameters and how they can aid in a more enjoyable and convincing output. This evaluation stage also helped to outline where the project had met it's core and stretch objectives.

7.2 What Did Not Go Well

The biggest issue with the project was the limitations on how complex the model could be based on hardware available. Ideally, the significant amount of time spent on exploring different options for hardware and optimising the model to fit the hardware's available capabilities and minimise training time could have been spent exploring different configurations of the transformer itself and how a different number of decoders, hidden layers, window size or attention heads could impact the quality of generations made.

Another problem with the model is the inconsistent output explained in section 5.10.4 that while mitigated to a certain extent with generation hyper-parameters are still present to a significant enough degree to warrant mention. This is most likely a product of the limitations imposed upon the model and project as a while by the lack of high performing hardware. However, considering how well the model performed with said limitation it can be said confidently that using the same techniques employed in this project with access to better equipment and therefore a model much more complex could mitigate these errors in a much more substantial way.

7.3 What Would Be Done Differently Next Time

Next time better source control and file management would be beneficial. While prototypes and project folders were effective and kept everything where they needed to be and backups were done to prevent loss of work, after a few prototypes were developed folders could became large and cluttered and difficult to navigate especially when working with the large amount of data for this project.

More background research earlier on in the project about what hardware would be available would have saved time later on. So next time considerably more time would go into researching needed hardware specifications and how to obtain them.

7.4 Did The Project Meet The Objectives

For the core and stretch objectives outlined in section 2.1, the model does meet both the core and stretch objectives.

- Extensive research was done on existing methods used to create generative music.
- Generated tracks were well structured and can be imported into third party software to be editing and used as desired.
- Generated tracks have both single and multi-track capabilities depending on the MIDI prompt.
- Generated tracks are difficult to distinguish between them and human made tracks as demonstrated in testing and evaluation.
- Generated tracks display a sensible track length and hold local and global structure as demonstrated in testing and evaluation.
- Extensive testing and evaluation was carried out display the ability of the model and gain insight into both how the generated music is perceived and how generation can be improved going forward.

8 Further Work

Going forward, there are a number of ways in which this project can be improved upon.

The first would be to vastly improve the complexity of the model with access to greater hardware. With a more complex model greater insight into the relationships between the tokens in the training data can be captured to eliminate most of the problems this project encountered with the output, eliminating inconsistent generations.

Another way would be to expand the training data to include tracks of different time signatures for more variety in the output.

With sufficient labelled data such as genre or artist a similar model could be developed using all the same techniques that can input text prompts that specify what genre or feel the output should be rather than prompting the model on a midi token input like this project. This could be more appropriate for users wanting a specific type of music for use in third party software.

Further and more extensive testing and evaluation could be done. It would be beneficial to include many more participants and be able to secure participants that have musical experience beyond just beginner for greater insight into this project and AI generated music and a whole.

9 Appendices

References

[1] Autoregressive long-context music generation with perceiver ar. https://magenta.tensorflow.org/perceiver-ar.

[2] Bitmidi.com. https://bitmidi.com/.

[3] https://datagy.io/one-hot-encoding/. https://datagy.io/one-hot-encoding/.

[4] https://miditok.readthedocs.io/. https://miditok.readthedocs.io/en/v3.0.1/bpe.html/.

[5] huggingface.co. https:/huggingface.co/.

[6] medium batches. https://medium.com/@elimu.michael9/understanding-epochs-and-batches-23120a04b3cb/.

[7] miditok 2.1.7 documentation. https://miditok.readthedocs.io/en/latest/tokenizations.html.

[8] mistral.ai. https:https://mistral.ai/.

[9] Classical Archives. classical archives: midi. https://www.classicalarchives.com/midi.html.

[10] The MIDI Association. Official midi specifications. https://www.midi.org/specifications.

[11] Thierry Bertin-Mahieux, Daniel PW Ellis, Brian Whitman, and Paul Lamere. The million song dataset. *Lahk*, 2011.

[12] Nicolas Boulanger-Lewandowski, Yoshua Bengio, and Pascal Vincent. Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. *arXiv preprint arXiv:1206.6392*, 2012.

[13] Rewon Child. Sparse attention. https://github.com/openai/sparse_attention#readme, Aug 2023.

[14] Rewon Child and Scott Gray. Generative modeling with sparse transformers. https://openai.com/research/sparse-transformer, journal=openai.com, Apr 2019.

[15] Michael Scott Cuthbert and Christopher Ariza. music21: A toolkit for computer-aided musicology and symbolic music data. *Music21*, 2010.

[16] Prantosh Das. Wavenet: A generative model for raw audio synthesis. https://medium.com/@prantosh.das97/wavenet-a-generating-model-for-raw-audio-synthesis-610242071c12, Oct 2021.

[17] Ketan Doshi. Transformers explained visually (part 1): Overview of functionality. https://towardsdatascience.com/transformers-explained-visually-part-1-overview-of-functionality-95a6dd4604 Jun 2021.

[18] Shiv Ram Dubey, Satish Kumar Singh, and Bidyut Baran Chaudhuri. Activation functions in deep learning: A comprehensive survey and benchmark. *Neurocomputing*, 503:92–108, 2022.

[19] Lucas N Ferreira, Levi HS Lelis, and Jim Whitehead. Computer-generated music for tabletop role-playing games. 2020.

[20] Nathan Fradet, Jean-Pierre Briot, Fabien Chhel, Amal El Fallah Seghrouchni, and Nicolas Gutowski. MidiTok: A python package for MIDI file tokenization. In *Extended Abstracts for the Late-Breaking Demo Session of the 22nd International Society for Music Information Retrieval Conference*, 2021.

[21] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[22] Curtis Hawthorne, Andrew Jaegle, Cătălina Cangea, Sebastian Borgeaud, Charlie Nash, Mateusz Malinowski, Sander Dieleman, Oriol Vinyals, Matthew Botvinick, Ian Simon, et al. General-purpose, long-context autoregressive modeling with perceiver ar. In *The Thirty-ninth International Conference on Machine Learning*, 2022.

[23] Curtis Hawthorne, Andriy Stasyuk, Adam Roberts, Ian Simon, Cheng-Zhi Anna Huang, Sander Dieleman, Erich Elsen, Jesse Engel, and Douglas Eck. Enabling factorized piano music modeling and generation with the MAESTRO dataset. In *International Conference on Learning Representations*, 2019.

[24] Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Noam Shazeer, Curtis Hawthorne, Andrew M. Dai, Matthew D. Hoffman, and Douglas Eck. Music transformer: Generating music with long-term structure. *CoRR*, abs/1809.04281, December 2018.

[25] Yu-Siang Huang and Yi-Hsuan Yang. Pop music transformer: Beat-based modeling and generation of expressive pop piano compositions. In *Proceedings of the 28th ACM International Conference on Multimedia*, MM '20, page 1180–1188, New York, NY, USA, 2020. Association for Computing Machinery.

[26] Shulei Ji, Xinyu Yang, and Jing Luo. A survey on deep learning for symbolic music generation: Representations, algorithms, evaluations, and challenges. *ACM Comput. Surv.*, 56(1), aug 2023.

[27] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b, 2023.

[28] Yuchao Li and Dimitri Bertsekas. Most likely sequence generation for n-grams, transformers, hmms, and markov chains, by using rollout algorithms, 2024.

[29] Kensuke Nakamura, Bilel Derbel, Kyoung-Jae Won, and Byung-Woo Hong. Learning-rate annealing methods for deep neural networks. *Electronics*, 10(16):2029, 2021.

[30] Sageev Oore, Ian Simon, Sander Dieleman, Douglas Eck, and Karen Simonyan. This time with feeling: Learning expressive musical performance. *Neural Computing and Applications*, 32:955–967, 2018.

[31] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks, 2013.

[32] Lutz Prechelt. Early stopping-but when? In *Neural Networks: Tricks of the trade*, pages 55–69. Springer, 2002.

[33] Colin Raffel and Daniel PW Ellis. Intuitive analysis, creation and manipulation of midi data with pretty_midi. In *15th international society for music information retrieval conference late breaking and demo papers*, pages 84–93, 2014.

[34] Yi Ren, Jinzheng He, Xu Tan, Tao Qin, Zhou Zhao, and Tie-Yan Liu. Popmag: Pop music accompaniment generation. In *Proceedings of the 28th ACM International Conference on Multimedia*, page 1198–1206. Association for Computing Machinery, 2020.

[35] Jamell Samuels. One-hot encoding and two-hot encoding: An introduction, 01 2024.

[36] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In Katrin Erk and Noah A. Smith, editors, *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany, August 2016. Association for Computational Linguistics.

[37] Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding, 2023.

[38] Shiliang Sun, Zehui Cao, Han Zhu, and Jing Zhao. A survey of optimization methods from a machine learning perspective. *IEEE transactions on cybernetics*, 50(8):3668–3681, 2019.

[39] OV Usatenko and GM Pritula. Information temperature as a parameter of random sequence complexity. *arXiv preprint arXiv:2307.12841*, 2023.

[40] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[41] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[42] Dimitri von Rütte, Luca Biggio, Yannic Kilcher, and Thomas Hofmann. Figaro: Generating symbolic music with fine-grained artistic control, 2022.

[43] Gavin Wright. What is midi (musical instrument digital interface)? – techtarget definition. https://www.techtarget.com/whatis/definition/MIDI-Musical-Instrument-Digital-Interface#:~:text=What%20are%20MIDI%20files%3F, Jan 2023.

[44] Jiacheng Xu, Caiming Xiong, Silvio Savarese, and Yingbo Zhou. Best-k search algorithm for neural text generation. *arXiv preprint arXiv:2211.11924*, 2022.

[45] Mingliang Zeng, Xu Tan, Rui Wang, Zeqian Ju, Tao Qin, and Tie-Yan Liu. MusicBERT: Symbolic music understanding with large-scale pre-training. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 791–800, Online, August 2021. Association for Computational Linguistics.

[46] Biao Zhang and Rico Sennrich. Root mean square layer normalization, 2019.

9.1 Weekly Log

The weekly log for the project can be found here:
https://github.com/DanielFoulkes-Halbard/AIMusicGenProject/blob/main/Notes/WeeklyLog

9.2 Project code

The code for the final project can be found here:
https://github.com/DanielFoulkes-Halbard/AIMusicGenProject/blob/main/Code/FYP_246736.ipynb

9.3 Original Project Proposal

The initial project proposal can be found here: https://github.com/DanielFoulkes-Halbard/AIMusicGenProject/blob/main/Notes/Final_Year_Project_Proposal%20(1).pdf

9.4 Evaluation Data

The raw data from the evaluation can be found here:
https://github.com/DanielFoulkes-Halbard/AIMusicGenProject/blob/main/Evaluation%20Responses.xlsx

9.5 User Testing Compliance Form

The user testing compliance form signed by myself and my supervisor can be found here:
https://github.com/DanielFoulkes-Halbard/AIMusicGenProject/blob/main/Notes/User%20Testing%20Compliance%
20Form.docx