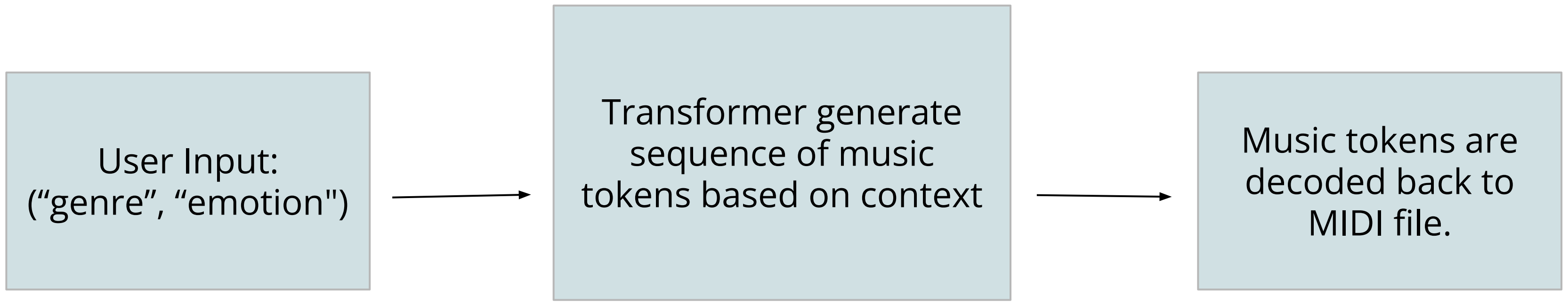
TEAM63 - 溜馬
總冠軍

組員: 112550042 葉哲睿
112550074 彭博脩
112550164 蔣忻勳
112550171 賴承浩

Introduction

- **Goal**

  The goal is to build a transformer AI that can automatically compose and generate music in accordance with user inputs. Users can choose a genre that goes with an emotion, such as ‘classical and happy,’ or ‘rock and sad’. The AI then outputs a piece of music according to the input.


User Input: (“genre”, “emotion")
Transformer generate sequence of music tokens based on context
Music tokens are decoded back to MIDI file.

Introduction

- ➢ **Why important? Why work on this?**
  When searching for music we like, we often find that the tracks we discover don't match what we have in mind. Therefore, we'd like to develop our own music-generating AI to create the exact music we want to listen to.
- ➢ Additionally, in the field of AI generated music, LSTM and transformer have been two of the most talked about methods. Therefore, we want to compare their ability to generate music, and we also want to analyze transformer's ability to generate music that's more fitting to a theme, and more structured throughout the track.

Related Work

1. AI Music Composer (Notation) (Spring 2022 Group 19)

https://github.com/jonehu901008/AI-comoser

This group used LSTM to train the AI to compose music and generate music notations. We use transformer and compare with LSTM instead. We also use a larger dataset and want to add the feature of specifying a genre and a emotion to generate music that fits them.

Related Work

2. Recognizing the genre of music (Spring 2024 Group 10)

https://github.com/vch2128/AIFinal-Music_Genre_Recognizer

This group's purpose is to train an AI with LSTM, and to classify genre of music from audio input. Our main goal is to generate music by users input.

Furthermore, this group used Librosa to extract features, whereas we transform music notes from MIDI file to "event tokens", and train the transformer to predict/generate the sequence of token events.

Dataset - XMIDI Dataset

Source & Data Size

We used XMIDI Dataset, the largest known symbolic music dataset with precise emotion and genre labels, comprising 108,023 MIDI files. The average duration of the music pieces is around 176 seconds, resulting in a total dataset length of around 5,278 hours.
https://github.com/xmusic-project/XMIDI_Dataset

Format

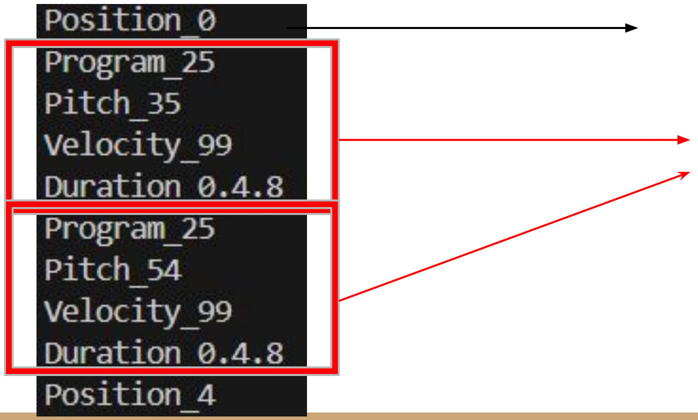
The dataset’s format is like “XMIDI_funny_country_JVZIL3M5.midi”, with the file name containing its genre and emotion.

Dataset

Preprocessing

To process the music in the MIDI file, we use REMI's Tokenizer from miditok https://miditok.readthedocs.io/en/latest/tokenizations.html

This transforms music notes into sequence tokens, for example:



```
Position_0
Program_25
Pitch_35
Velocity_99
Duration_0.4.8
Program_25
Pitch_54
Velocity_99
Duration_0.4.8
Position_4
```

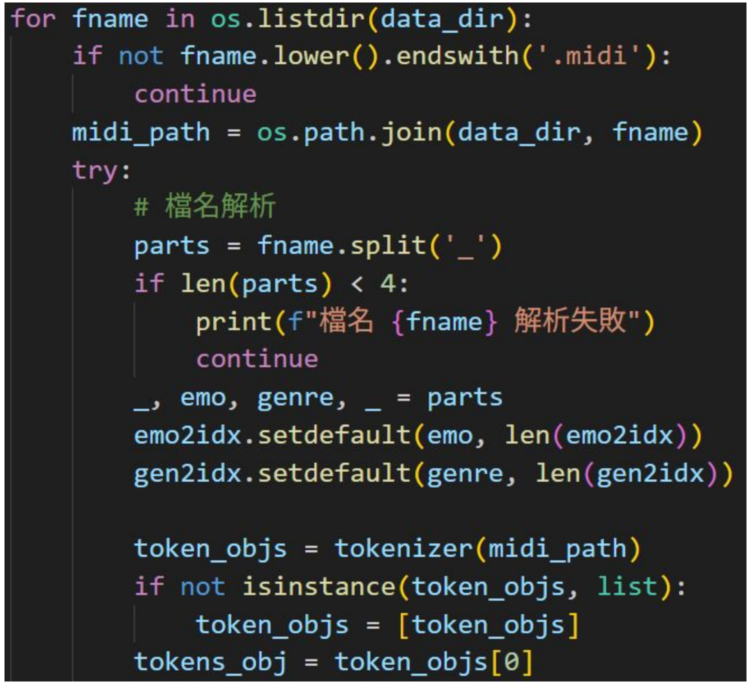
position of the bar

A music note! With its:
1. type of instrument,
2. pitch,
3. velocity (the force with which it is played),
4. duration

Dataset

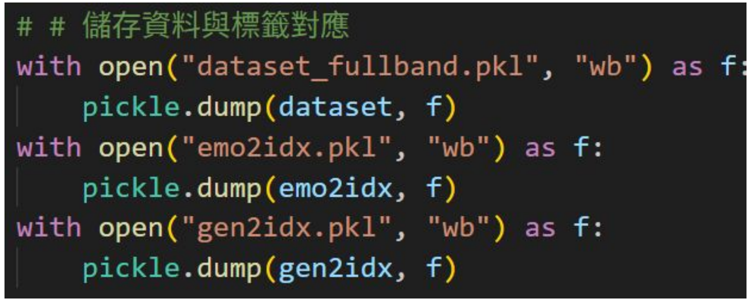
Each music file’s:
1. emotion label,
2. genre label,
3. sequence of tokens,

are then merged into the list “dataset”, which, along with the collected list of emotions and genres, is saved to .pkl using pickle.



```python
for fname in os.listdir(data_dir):
    if not fname.lower().endswith('.midi'):
        continue
    midi_path = os.path.join(data_dir, fname)
    try:
        # 檔名解析
        parts = fname.split('_')
        if len(parts) < 4:
            print(f"檔名 {fname} 解析失敗")
            continue
        _, emo, genre, _ = parts
        emo2idx.setdefault(emo, len(emo2idx))
        gen2idx.setdefault(genre, len(gen2idx))

        token_objs = tokenizer(midi_path)
        if not isinstance(token_objs, list):
            token_objs = [token_objs]
        tokens_obj = token_objs[0]
```

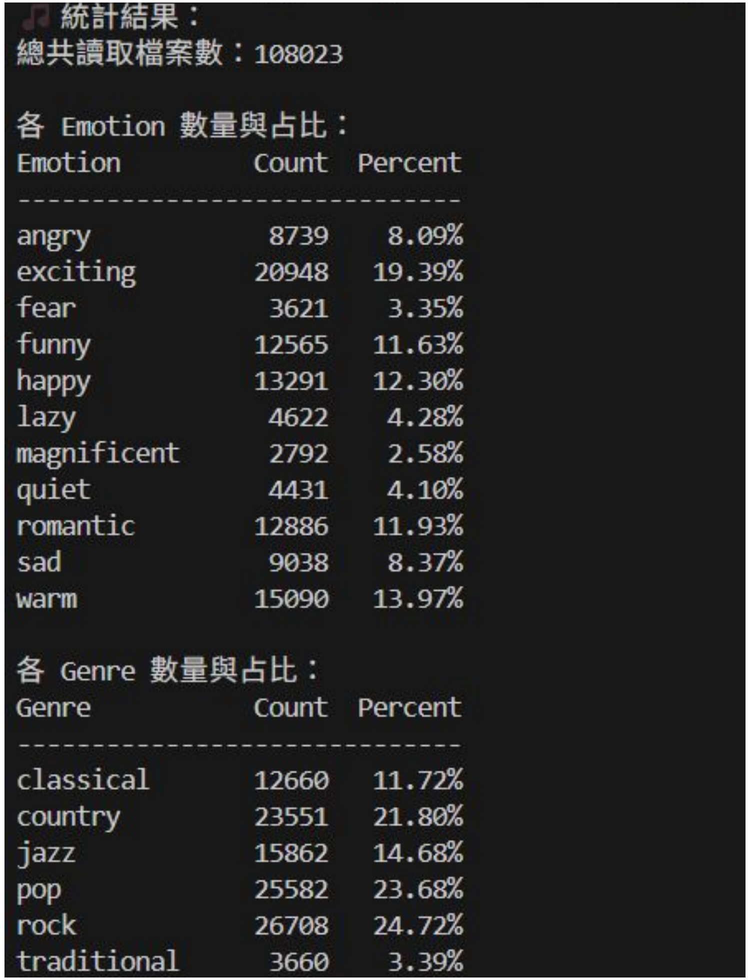


```python
# # 儲存資料與標籤對應
with open("dataset_fullband.pkl", "wb") as f:
    pickle.dump(dataset, f)
with open("emo2idx.pkl", "wb") as f:
    pickle.dump(emo2idx, f)
with open("gen2idx.pkl", "wb") as f:
    pickle.dump(gen2idx, f)
```

Dataset

The proportion of each genres and emotions

The data size of each categories may influence the training result !



```
♫ 統計結果：
總共讀取檔案數：108023

各 Emotion 數量與占比：
Emotion          Count   Percent
------------------------------
angry             8739     8.09%
exciting         20948    19.39%
fear              3621     3.35%
funny            12565    11.63%
happy            13291    12.30%
lazy              4622     4.28%
magnificent       2792     2.58%
quiet             4431     4.10%
romantic         12886    11.93%
sad               9038     8.37%
warm             15090    13.97%

各 Genre 數量與占比：
Genre            Count   Percent
------------------------------
classical        12660    11.72%
country          23551    21.80%
jazz             15862    14.68%
pop              25582    23.68%
rock             26708    24.72%
traditional       3660     3.39%
```

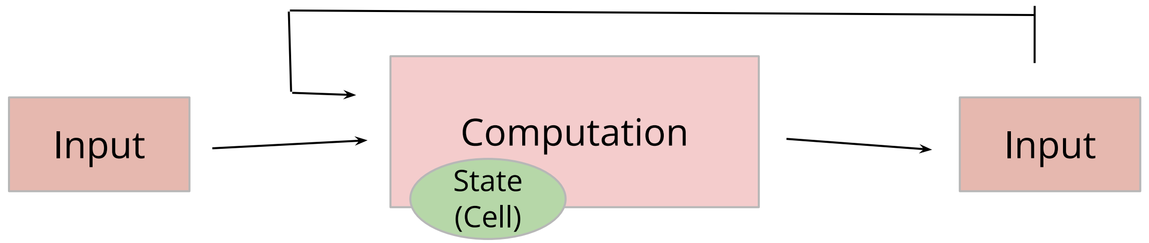
Baseline - Long-Short-Term Memory

What is Long-Short-Term Memory(LSTM)?

RNN with "Memory Cell," enabling the network to learn when to remember, and when to forget.

The cell has three "gates":

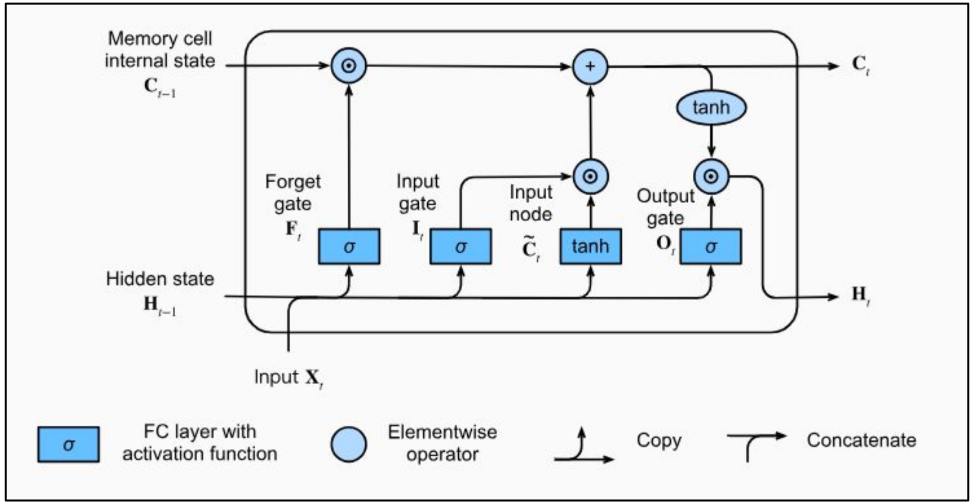
1. Forget Gate -> decide which information to discard
2. Input Gate -> decide which new information to add
3. Output Gate -> decides what to output from the cell



Baseline - Long-Short-Term Memory

In each recursive iteration, the LSTM executes gating operations at every timestep(Xt), simultaneously updating and preserving its

1. cell state(C)
 -> the long-term memory
2. hidden state(H)
 -> the short-term memory


Memory cell internal state $\mathbf{C}_{t-1}$
$\mathbf{C}_t$
tanh
Forget gate $\mathbf{F}_t$
Input gate $\mathbf{I}_t$
Input node $\tilde{\mathbf{C}}_t$
Output gate $\mathbf{O}_t$
σ
tanh
Hidden state $\mathbf{H}_{t-1}$
$\mathbf{H}_t$
Input $\mathbf{X}_t$
FC layer with activation function
Elementwise operator
Copy
Concatenate

Dive into Deep Learning:
https://d2l.ai/chapter_recurrent-modern/lstm.html

Baseline - Long-Short-Term Memory

How is it used to generate music?

The model is trained with dataset of MIDI files, during training:
1. it processes music tokens from the files, it predicts the next music token, and see if it is the same as the next.
2. Calculate loss: Use cross-entropy loss between the model's predicted distribution and the true next token.
3. Adjust weight matrix.

How to compare it with the main approach?

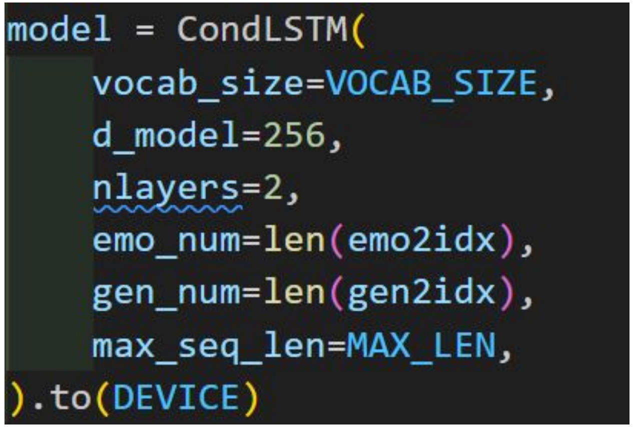
- ➢ **Cross-Entropy Loss**
- ➢ **Token Accuracy**

We would describe more in Evaluation metric !

Baseline - Long-Short-Term Memory

Parameters

Both d_model & max_seq_len are the same as transformer.



```
model = CondLSTM(
    vocab_size=VOCAB_SIZE,
    d_model=256,
    nlayers=2,
    emo_num=len(emo2idx),
    gen_num=len(gen2idx),
    max_seq_len=MAX_LEN,
).to(DEVICE)
```

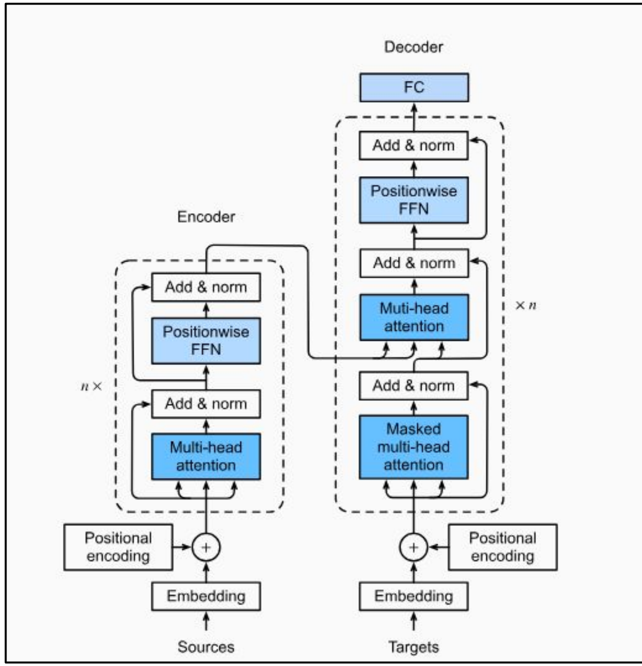
- hidden_size=256
- num_layer=2

  num_layer is different from transformer. Because LSTM layers process in sequential and transformer process in parallel. So LSTM layer is usually set from 1~ 3.

Main Approach - Transformer

What is Transformer?

1. **Embedding** - Different from RNN, Transformer Reads the whole sequence at once. To enable the model to understand the order of each token in a sequence, 'Positional Encoding' needs to be additionally added.
2. **Multi-Head self-attention** - The input is divided into several segments to identify different points of emphasis within those segments. Furthermore, the Transformer can access the entire sequence's data at any given timestep Xt.

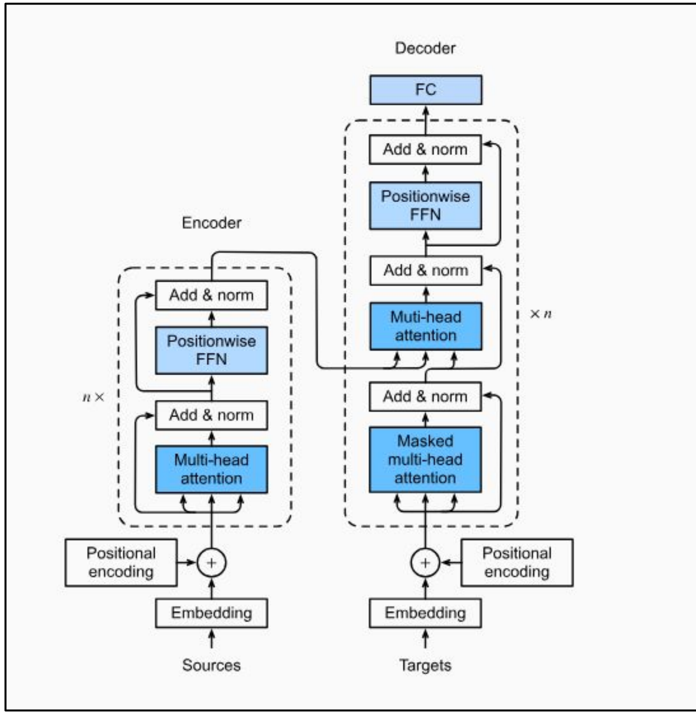


Main Approach - Transformer

What is Transformer?

3. **Position-wise Feed-Forward Network（FFN）** Independently for each position, two fully connected layers are applied to enhance non-linear expressiveness.

4. **Softmax（FC）**- Passes through another fully connected layer to project its dimensionality to the vocabulary size, followed by a Softmax operation to obtain the probability distribution for the next token.



Main Approach - Transformer

Why is it suitable for generating music?

1. **Multi-head attention**: It simultaneously captures relationships among melody, rhythm, harmony, and other musical dimensions.
2. **Parallelism and long-term dependency**: It can take into account the structure of an entire piece—chord progressions, repeated motifs, rhythmic patterns—at once.
   → "Thematic coherence"
3. **Flexible conditioning**: Emotion, genre, or event-based cues can be easily injected as embeddings.

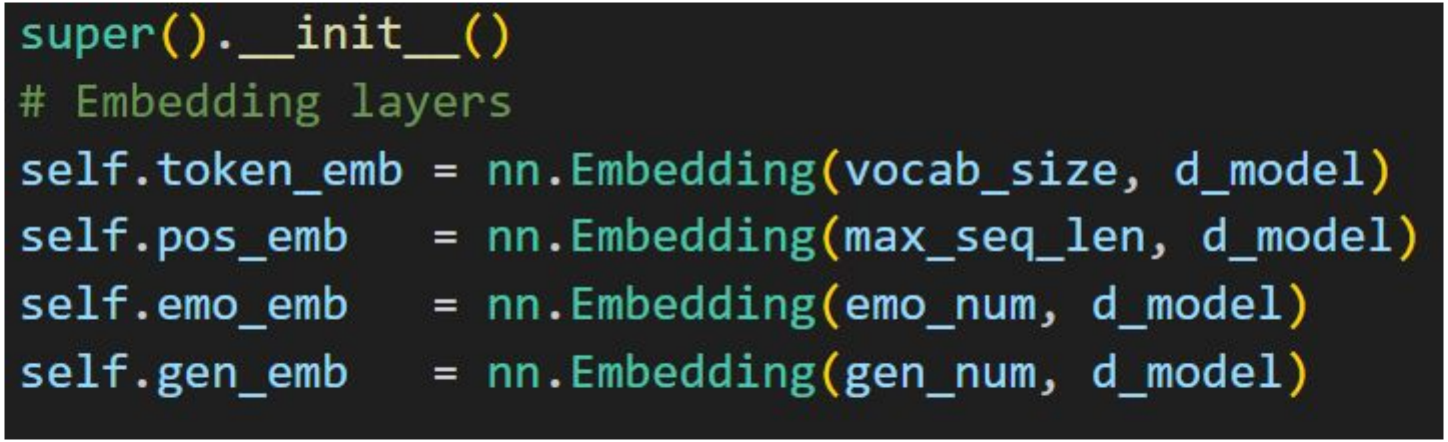
On the other hand, LSTM does not capture information from long-term sequence well enough (due to the lack of self-attention mechanism) to learn the more complicated structure of the entire piece.

Main Approach - Transformer

Implementation - CondTransformer Model Architecture

1. Embedding Layer:

- Convert “token IDs, positional indices, emotion, and genre” into vectors,
- Add them at each position to form the Transformer input.



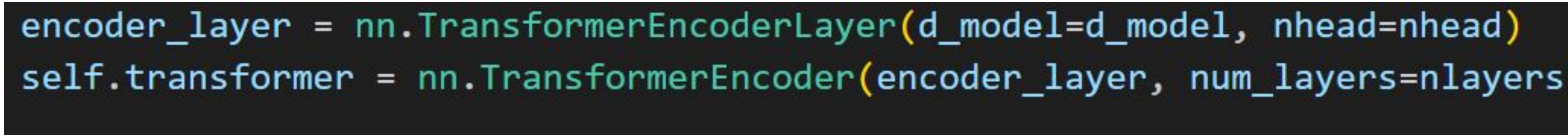
```
super().__init__()
# Embedding layers
self.token_emb = nn.Embedding(vocab_size, d_model)
self.pos_emb   = nn.Embedding(max_seq_len, d_model)
self.emo_emb   = nn.Embedding(emo_num, d_model)
self.gen_emb   = nn.Embedding(gen_num, d_model)
```

Main Approach - Transformer

Implementation - CondTransformer Model Architecture

2. Transformer Layers:

- Use a multi-layer TransformerEncoder with a causal mask to implement autoregressive masked self-attention.
- Sample [t_0, t_1, t_2, t_3]，inputs [t_0, t_1, t_2]。
- Stacking multiple layers enables it to capture long-range dependencies at different levels ex motif repetition, chord progressions



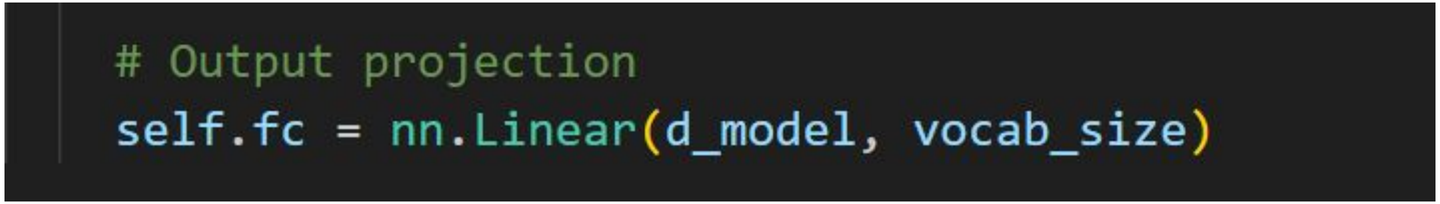
```
encoder_layer = nn.TransformerEncoderLayer(d_model=d_model, nhead=nhead)
self.transformer = nn.TransformerEncoder(encoder_layer, num_layers=nlayers
```

Main Approach - Transformer

Implementation - CondTransformer Model Architecture

3. Output Layer:

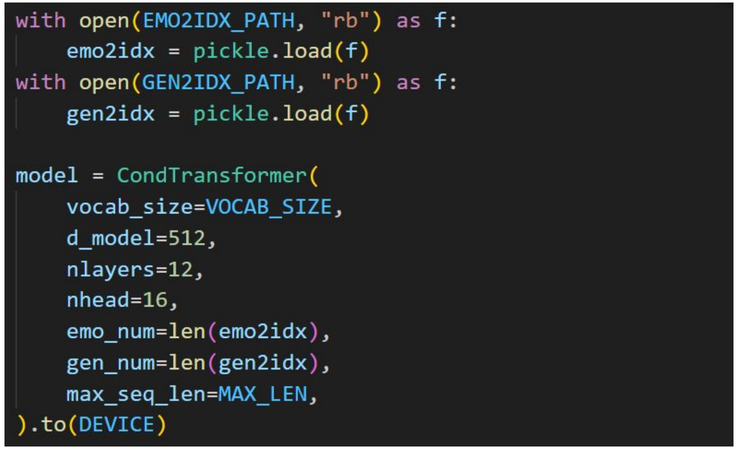
- A single linear projection maps the hidden vectors to dimensions, producing the logits distribution for the next token.
- Applying a softmax then yields the actual probability distribution



```
# Output projection
self.fc = nn.Linear(d_model, vocab_size)
```

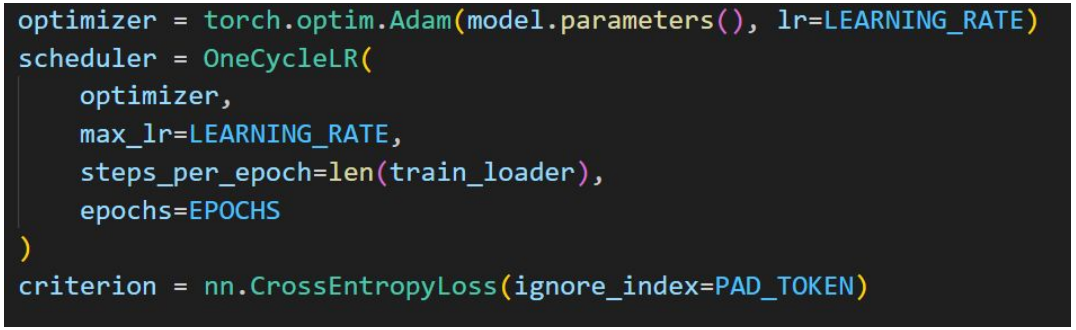
Main Approach - Transformer
Implementation – Training Process

1. Load the preprocessed data to obtain the number of emotion and genre categories.
2. Initialize the CondTransformer.
3. Define the CrossEntropyLoss and the Adam optimizer.



```python
with open(EMO2IDX_PATH, "rb") as f:
    emo2idx = pickle.load(f)
with open(GEN2IDX_PATH, "rb") as f:
    gen2idx = pickle.load(f)

model = CondTransformer(
    vocab_size=VOCAB_SIZE,
    d_model=512,
    nlayers=12,
    nhead=16,
    emo_num=len(emo2idx),
    gen_num=len(gen2idx),
    max_seq_len=MAX_LEN,
).to(DEVICE)
```



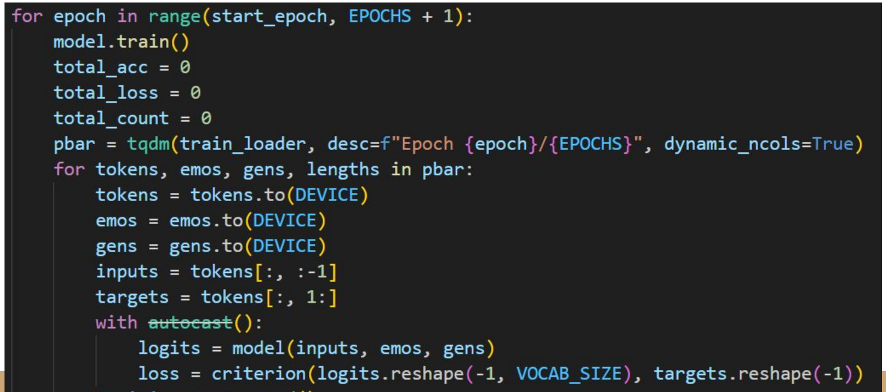
```python
optimizer = torch.optim.Adam(model.parameters(), lr=LEARNING_RATE)
scheduler = OneCycleLR(
    optimizer,
    max_lr=LEARNING_RATE,
    steps_per_epoch=len(train_loader),
    epochs=EPOCHS
)
criterion = nn.CrossEntropyLoss(ignore_index=PAD_TOKEN)
```

Main Approach - Transformer

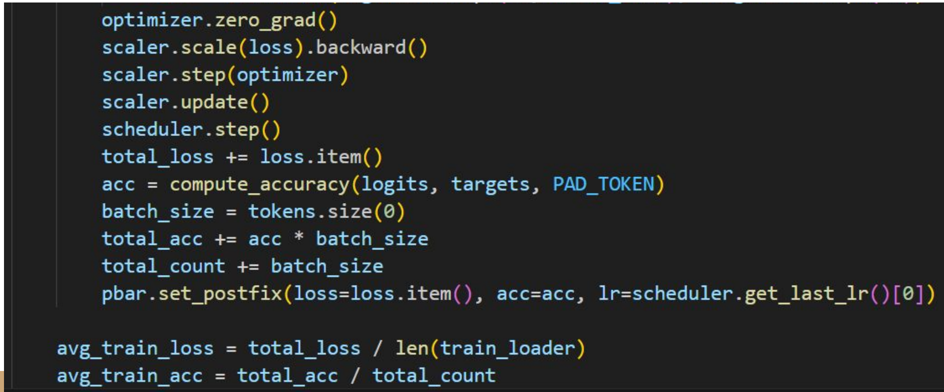
Implementation – Training Process

Training Loop (Epoch + Batch)

1. Iterate over multiple epochs, shifting each batch of tokens to create the input tokens. Use teacher forcing to generate inputs and targets.
2. Compute the loss, perform backpropagation, and update the model parameters. Periodically save checkpoints and keep track of the best



```
for epoch in range(start_epoch, EPOCHS + 1):
    model.train()
    total_acc = 0
    total_loss = 0
    total_count = 0
    pbar = tqdm(train_loader, desc=f"Epoch {epoch}/{EPOCHS}", dynamic_ncols=True)
    for tokens, emos, gens, lengths in pbar:
        tokens = tokens.to(DEVICE)
        emos = emos.to(DEVICE)
        gens = gens.to(DEVICE)
        inputs = tokens[:, :-1]
        targets = tokens[:, 1:]
        with ~~autocast~~():
            logits = model(inputs, emos, gens)
            loss = criterion(logits.reshape(-1, VOCAB_SIZE), targets.reshape(-1))
```



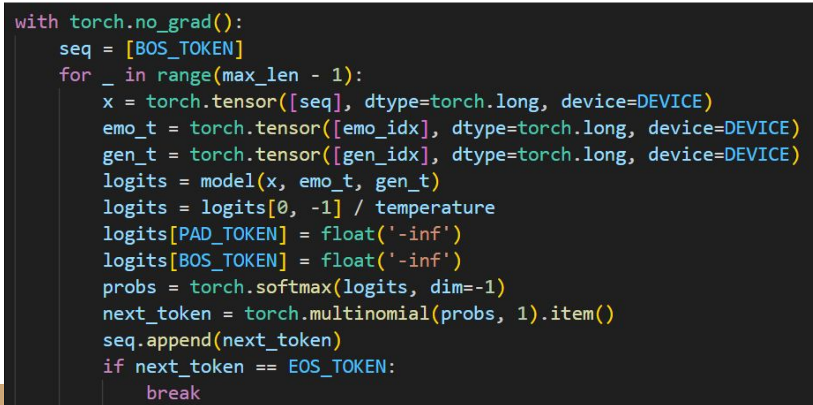
```
        optimizer.zero_grad()
        scaler.scale(loss).backward()
        scaler.step(optimizer)
        scaler.update()
        scheduler.step()
        total_loss += loss.item()
        acc = compute_accuracy(logits, targets, PAD_TOKEN)
        batch_size = tokens.size(0)
        total_acc += acc * batch_size
        total_count += batch_size
        pbar.set_postfix(loss=loss.item(), acc=acc, lr=scheduler.get_last_lr()[0])

    avg_train_loss = total_loss / len(train_loader)
    avg_train_acc = total_acc / total_count
```

Main Approach - Transformer

Implementation – Generation

1. Feed the current (or initial) sequence of generated tokens into the model to obtain the logits for the next position.
2. Use Top-K sampling to select a token, append it to the end of the sequence, and repeat until the maximum length is reached.
3. Decode the final token sequence back into a MIDI file.



```
with torch.no_grad():
    seq = [BOS_TOKEN]
    for _ in range(max_len - 1):
        x = torch.tensor([seq], dtype=torch.long, device=DEVICE)
        emo_t = torch.tensor([emo_idx], dtype=torch.long, device=DEVICE)
        gen_t = torch.tensor([gen_idx], dtype=torch.long, device=DEVICE)
        logits = model(x, emo_t, gen_t)
        logits = logits[0, -1] / temperature
        logits[PAD_TOKEN] = float('-inf')
        logits[BOS_TOKEN] = float('-inf')
        probs = torch.softmax(logits, dim=-1)
        next_token = torch.multinomial(probs, 1).item()
        seq.append(next_token)
        if next_token == EOS_TOKEN:
            break
```

Evaluation Metric - Model

Cross-Entropy Loss

- We use cross-entropy loss to train and evaluate our model.
- It shows how confident and accurate our model is at predicting the next token in a sequence.
- Lower loss means the model makes better predictions.

Token Accuracy

- We measure how many tokens the model predicts correctly, ignoring padding tokens.
- Higher accuracy means our model is more precise in generating or classifying sequences.

Evaluation Metric - Model

model	val loss	val acc
Baseline(LSTM)	1.49	0.55
Transformer	0.66	0.81

We set the same 20% of dataset as validation dataset while another 80% is used to train.

Since lower validation loss and higher accuracy means better performance under the training objective in generative AI, our main approach model did perform better than the baseline.

(All the result accuracy is rounded off to the 2nd decimal place)

Results & Analysis\ Type of Experiment

Experiment 1: Hyperparameter Sensitivity Analysis

- **Learning Rate**: Tested 1e-3, 5e-4, 1e-4, and 5e-5.
- **Batch Size**: Tested 32, 64, and 128.
- **Model Depth** (nlayers): Tested 6, 8, and 12.

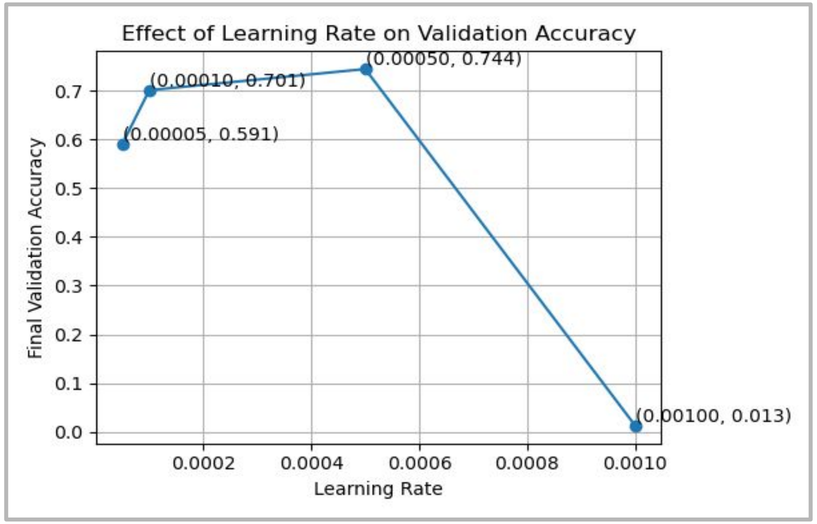
Default:

- Learning Rate: 1e-4.
- Batch Size: Tested 64.
- Model Depth (nlayers): 8.

Results & Analysis\ Discussion and Analysis

Learning Rate:

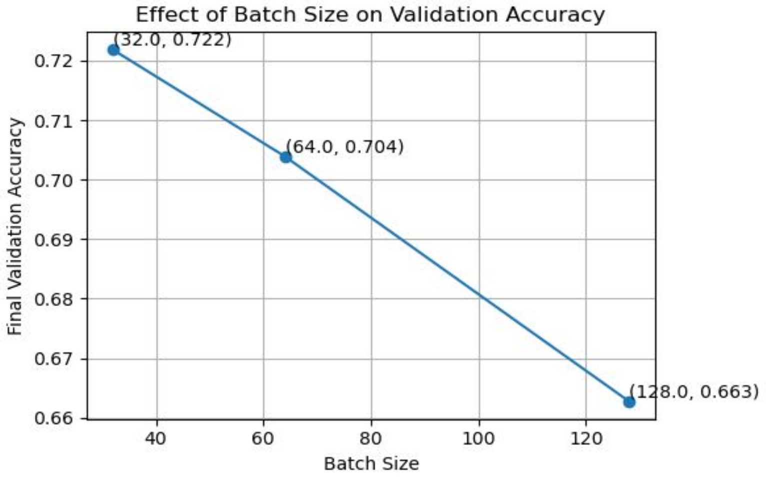
- peak: 5e-4 learning rate.
- observation: drop slightly at 1e-3 and 5e-5.
- conclusion: moderate learning rates (5e-4 ~ 1e-4) are optimal .



Results & Analysis\ Discussion and Analysis

Batch Size:

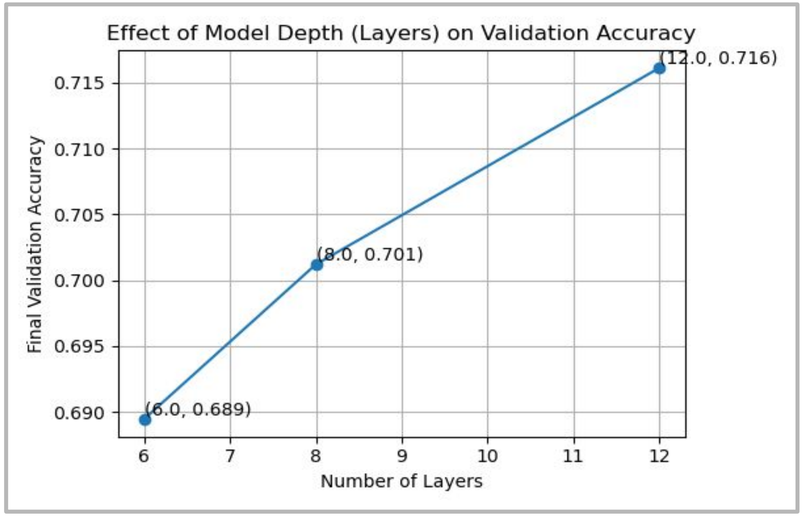
- peak: 32 batch size
- conclusion:
  a. smaller batches generalize better in the early training stage (10 epochs).
  b. the advantage of small batch sizes not guaranteed in long-term training.



Results & Analysis\ Discussion and Analysis

Model Depth:

- peak: 12 layers.
- observation: more layers higher accuracy.
- conclusion: deeper models up to 12 layers help performance.



Results & Analysis\ Type of Experiment

Experiment 2: Model Robustness to Noise

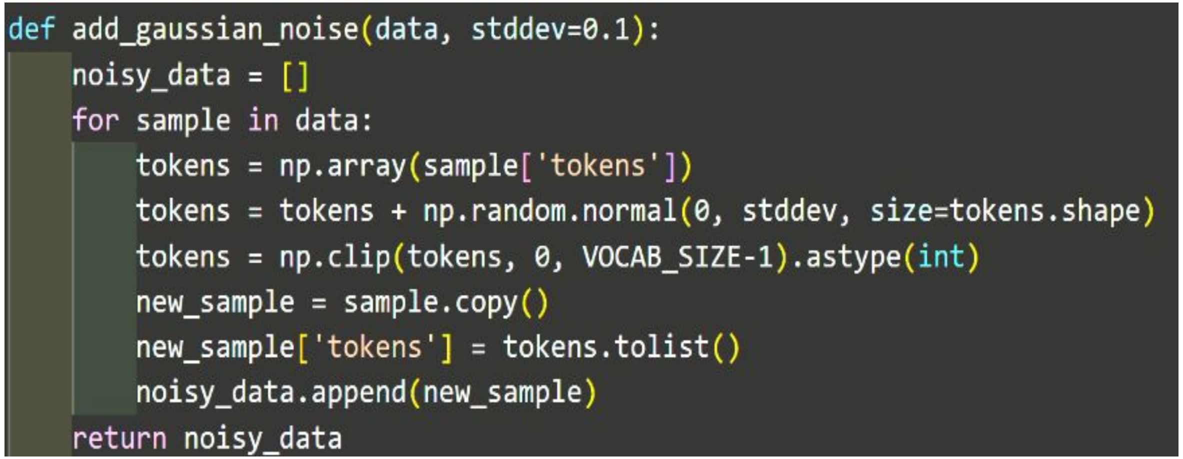
We tested three model types (Base, AttentionDropout, LayerNorm) with gaussian noise.

Key Terms

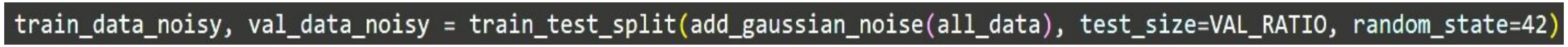
- **AttentionDropout:** Model with dropout(0.3) in the attention layers.
- **LayerNorm:** Model with layer normalization in each layer.

Results & Analysis\ Type of Experiment

Add Gaussian Noise to dataset



```python
def add_gaussian_noise(data, stddev=0.1):
    noisy_data = []
    for sample in data:
        tokens = np.array(sample['tokens'])
        tokens = tokens + np.random.normal(0, stddev, size=tokens.shape)
        tokens = np.clip(tokens, 0, VOCAB_SIZE-1).astype(int)
        new_sample = sample.copy()
        new_sample['tokens'] = tokens.tolist()
        noisy_data.append(new_sample)
    return noisy_data
```



```python
train_data_noisy, val_data_noisy = train_test_split(add_gaussian_noise(all_data), test_size=VAL_RATIO, random_state=42)
```

Results & Analysis\ Discussion and Analysis

- **LayerNorm** and have lower loss under noise, as expected.
- **AttentionDropout** get more loss when noise was added.
  Reason: number of epochs is not enough (only 10) or dropout(0.3) value not fit.

LayerNorm is more robust to noise than the others in early epochs.

	val acc	val loss
AttentionDropout	0.358	1.60
Base	0.367	1.56
LayerNorm	0.369	1.54

Music-wise Analysis(Subjective)

Although there’s no “right” or “wrong” in music, we still want to analyze how well the results are to human. After all, the music are made to be listened to by human!

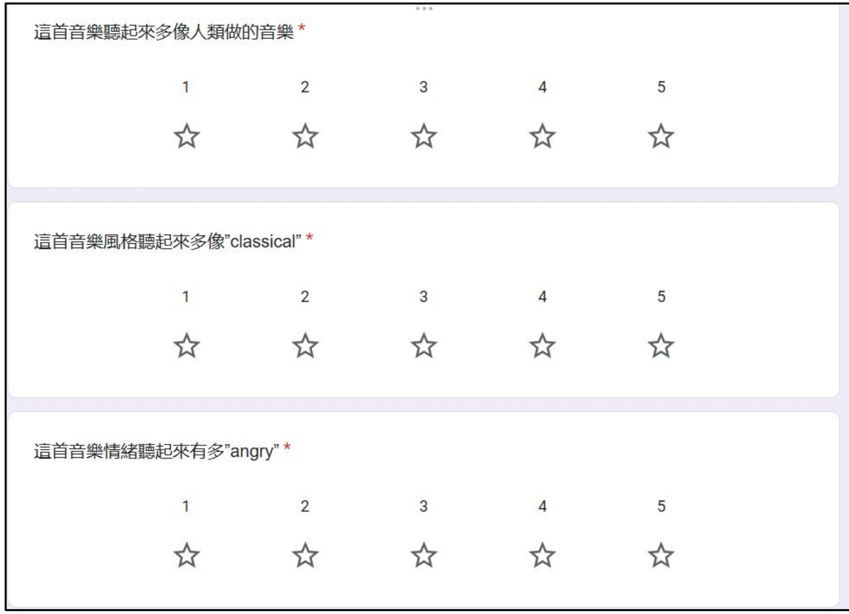
1. Survey to see people’s opinion on the accuracy of the matching of the Genre/Emotion.
2. We listen to the music, and very subjectively analyze what structure can we hear from the music generated by transformer.

1. Survey

We prepared five outputs music of different genre/emotion inputs, and surveyed our classmates' opinions after listening.

For each track, we asked 3 questions:
1. How much does the piece sounds like it's made by human?
2. How much does the piece resemble the genre "{genre}"?
3. How much does the piece resemble the emotion "{emotion}"?

And rate them from score 1 to 5.



這首音樂聽起來多像人類做的音樂 *

1	2	3	4	5
☆	☆	☆	☆	☆

這首音樂風格聽起來多像"classical" *

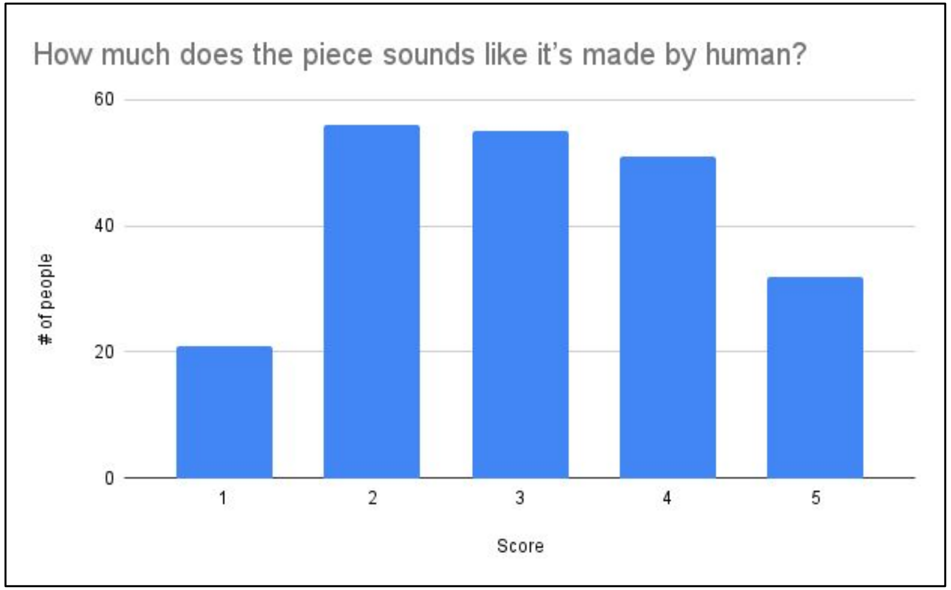
1	2	3	4	5
☆	☆	☆	☆	☆

這首音樂情緒聽起來有多"angry" *

1	2	3	4	5
☆	☆	☆	☆	☆

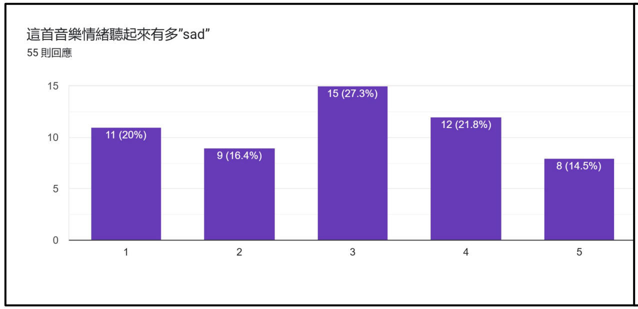
1. Survey

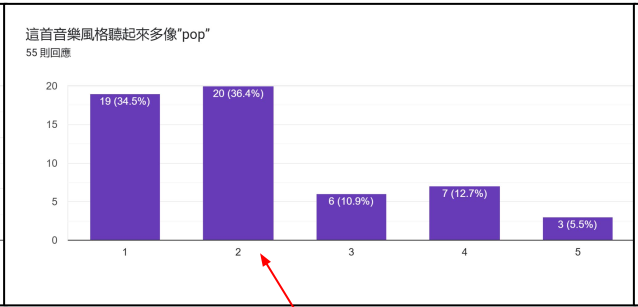
If we merge results of all five music pieces, we cannot draw clear conclusions. For example, on the right is “how much the piece sounds like it is made by human”, and as we can see, it is distributed very evenly.

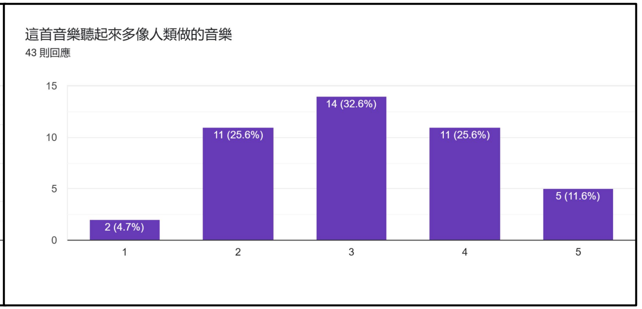


1. Survey

However, once we separately look at the results, we will see that some genre/emotion inputs perform better than others. For example, the input “sad/pop” performs rather poorly, as seen from the graphs below. Specifically, most people don’t think it resembles “pop.”





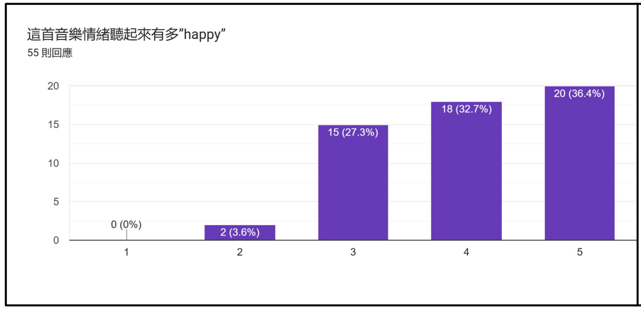


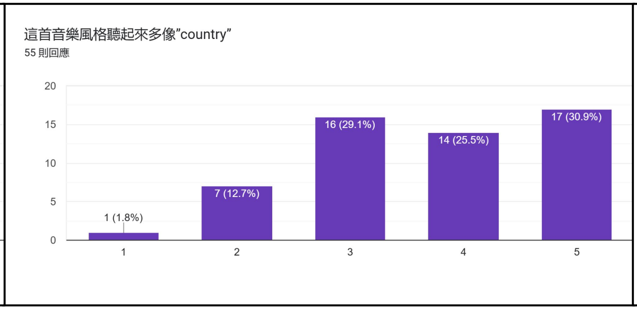
bad!

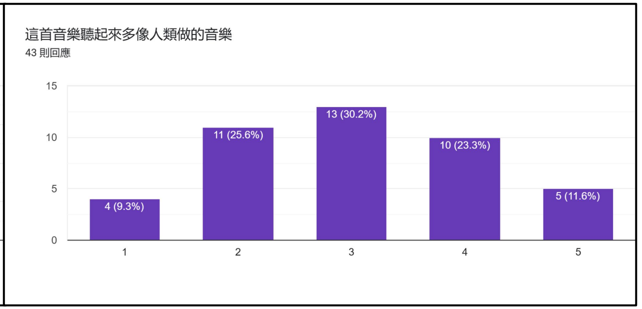
1. Survey

On the other hand, input, “happy/country" performs very well. Many people think the song gives happy feelings and does resemble a country song.

**Other results can be seen via the link to the Google Form in our Github page.







1. Survey - Analysis

We think this situation stems from 3 reasons:

1. The model is not complicated enough/ not well trained enough to really extract the “characteristics” of each genre/emotion.
2. Some categories have many variations of styles contained in it. For example, “pop” music doesn’t really have a very clear definitions. A lot of music can be considered “pop”, so the AI blended the multiple styles of pop music together, causing ambiguity.
3. Different categories differ in sizes of dataset, causing the AI to not learn the ones that have small size well.

2. Subjective Listening

We are no experts in music, and therefore we can only give very shallow analysis to the music. From what we’ve heard, we can notice the following:

1. There are real music structures in the tracks. The music, most of the time, is following a “key" using chords, notes from a certain key, and even ending on the central notes, as most real-life music pieces are.
2. “Themes” can be heard in the music, meaning some music phrases would be repeated to establish the sense of familiarity to the listeners.
   **this can be clearly heard in the sample track ”sad_pop.mp3”, as a theme melody is repeated throughout.

2. Subjective Listening

3. Sections organizations can be haerd. For example, in “angry_classical.mp3" , in the beginning it is quiet and builds up emotions. In the middle part, it becomes loud and complicated. And finally, it goes quiet again and ends pretty reasonably. $\longleftarrow$ "起承轉合"

We think these are results from the transformer’s “self-attention” mechanism. It provides the model to look at the entire music piece at any given time, so it picked up these characteristics during training and the whole track is more consistent from start to end.

Results & Analysis\ Limitation of work

1. Limited Dataset:

- prevent deeper models from achieving their full potential.

2. GPU Memory Constraints:

- setting MAX_LEN or batch size too high cause exceeding memory limit or reaching full computational load (even with a high-end GPU like the 4090).

3. Lack of Real-world Application Testing:

- have not been tested in a real-world or production-like environment with enough scales to measure generalization performance.

Results & Analysis\ Apply to practical use

Potential real-world uses:

- **AI-assisted composition:**
  For Musicians and producers, quickly generate new melodies or accompaniment tracks.
- **Personalized background music:**
  For Apps and games, dynamically generate background music.
- **Creative inspiration:**
  For composers, help overcome writer's block as a creative assistant.

However, **whether AI should be used to generate music is another debatable issue**, as seen from the recent controversy of ChatGPT generating images.

Others

- Github link：https://github.com/Hsiu727/AI_final_project_team_63
- Reference

  Dive into Deep Learning: LSTM
  https://d2l.ai/chapter_recurrent-modern/lstm.html
  Dive into Deep Learning: Transformer
  https://d2l.ai/chapter_attention-mechanisms-and-transformers/transformer.html
  XMusic - XMIDI Dataset
  https://github.com/xmusic-project/XMIDI_Dataset
  IBM Technology - What is LSTM (Long Short Term Memory)?
  https://www.youtube.com/watch?v=b61DPVFX03I&ab_channel=IBMTechnology

Others

- **Contribution of each member**

葉哲睿: Model Experiment, Analysis , model training, dataset processing, GUI

彭博脩: LSTM model, dataset, Report: background information, Github files organization

蔣忻勳: transformer model, dataset preprocessing, utility functions, Github README, Music Survey, Music Analysis

賴承浩: LSTM model, Report: Transfomer explanation, find dataset , Music Survey, GUI