Assignment 2

By Juhak Lee, Rachel Wang, Slater Mutunga, Vincent Tu

1. Symbolic, Unconditioned Generation

1.1 EDA, Data Processing

1.1 Context:

- Dataset: We used the MAESTRO 2018 dataset, which contains high-quality MIDI recordings aligned with real piano performances.

- Purpose: This dataset is suitable for tasks like symbolic music modeling, such as chord progression generation and melody composition.

- Collection Method: MIDI files were extracted from recorded performances and used as symbolic representations of music.

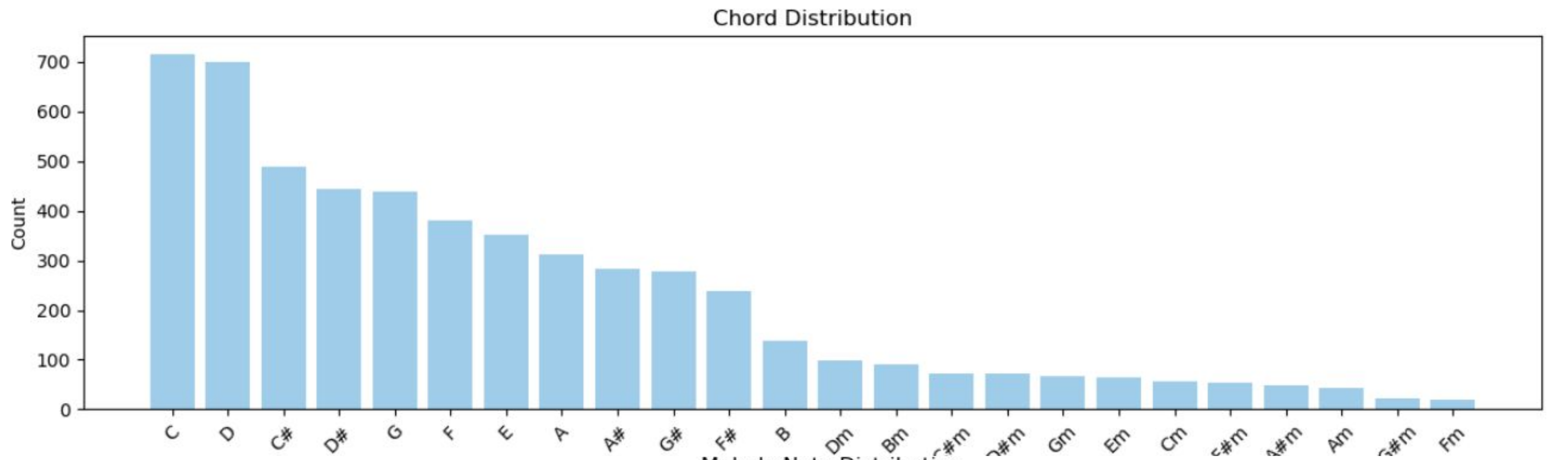
1.1 Discussion:

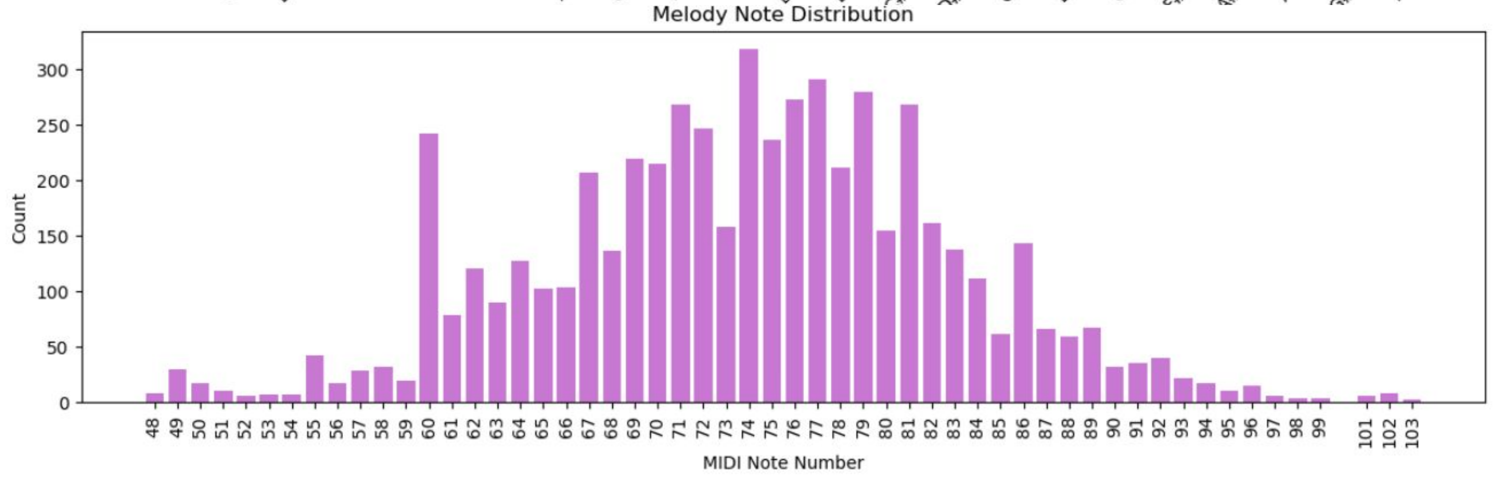
- We parsed each MIDI file and split it into 1-second intervals.

- For each time step:
  a. Chord Estimation: We estimated the chord using the set of pitch classes at that moment (both major and minor chords).
  b. Melody Extraction: We selected the highest-pitched note (above C3) as the melody note.

- We collected:
  a. A sequence of chords (chord progression)
  b. A sequence of melody notes (melody line)

1.1 Code:


Chord Distribution
Count
0
100
200
300
400
500
600
700
C
D
C#
D#
G
F
E
A
A#
G#
F#
B
Dm
Bm
Gm
Em
Cm
Am
Fm


Melody Note Distribution
Count
0
50
100
150
200
250
300
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
101
102
103
MIDI Note Number

1. Symbolic, Unconditioned Generation

1.2 Modeling

1.2 Context:

- Task split into two parts:
  a. Chord progression → sequence prediction (Markov Chain)
  b. Melody generation → sequence-to-one prediction (RNN)
- Inputs:
  a. Chord: previous 2 chords
  b. Melody: 8-note sequences

we separated chords and melodies, which made the generation more musically coherent and natural.

1.2 Discussion:

Chord Model (Markov Chain)

- Simple and less computationally expensive
- Captures local transitions well
- Limited global structure or long-term memory
- Easy to implement, fast generation

Melody Model (RNN - LSTM)

- Able to learn complex musical patterns
- Learns temporal dependencies across notes
- More computationally expensive
- Requires training and careful tuning

1.3 Code:

1. extract_data() + estimate_chord()

We first extract data from MIDI files using pretty_midi.
For each one-second window, we collect active pitches and use the estimate_chord() function to infer the chord.

- estimate_chord() uses pitch-class templates for major and minor chords, and does a partial match if no exact match is found.
- We also assume that the highest active pitch in each segment likely represents the melody note.

This step separates chords and melodies early on, which is crucial for our later modeling decisions.

1.3 Code:

2. ChordMarkovChain Class

To generate new chord progressions, we train a Markov Chain of order 2.

- This means it uses the last 2 chords to predict the next one.
- It’s a lightweight, data-efficient model that captures local transitions well.
- We chose this for simplicity and because chord progressions often have strong local dependencies.

1.3 Code:

3. MelodyDataset + MelodyRNN Class

We use a PyTorch Dataset to feed melody sequences into our LSTM-based RNN.

- Each training example is 8 consecutive notes predicting the 9th.
- The model embeds note indices, processes them through a 2-layer LSTM, and outputs logits over the note vocabulary.
- We use cross-entropy loss and Adam optimizer.

This architecture lets the model learn pitch movement and structure over time.

1.3 Code:

4. generate_melody()

Once trained, we generate a melody by seeding the RNN with a random melody slice and letting it autoregress forward for 32 steps.

- This corresponds to 4 notes per chord over 8 chords.

5. create_midi_file()

Finally, we merge the generated chord progression and melody into a new MIDI file.

- Each chord lasts 4 beats.
- Melody notes are added at 1-beat intervals.
- We use pretty_midi to write and save the file.

1. Symbolic, Unconditioned Generation

1.3 Evaluation

1.3 Context

🧪 **How should your task be evaluated?**

- Whether the generated 8-chord progression is musically plausible and coherent
- Whether each group of 4 melody notes aligns well with the harmonic context of the corresponding chord
- Whether the full 32-note melody line exhibits continuity, variety, and musicality

✅ **What are the properties of a "good" output?**

- **A consistent structure is enforced: each chord is paired with exactly 4 melody notes**, ensuring temporal and harmonic alignment

📉 **What is the relationship between the model objective and musical properties?**

- The RNN is trained to minimize cross-entropy loss (i.e., perplexity), encouraging it to predict likely next notes based on previous context
- While lower perplexity indicates better sequence modeling, it doesn't guarantee pleasant or harmonically valid music

1.3 Discussion

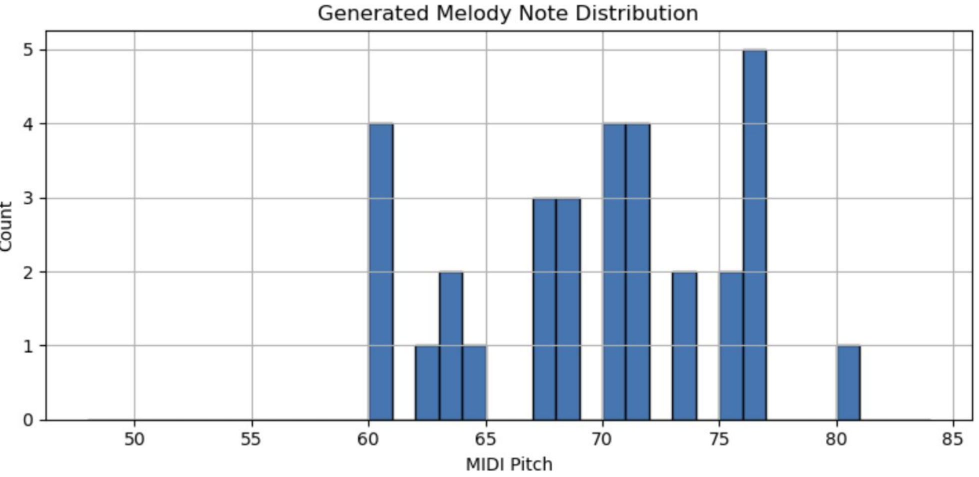
What are some baselines (trivial or otherwise) for your task?

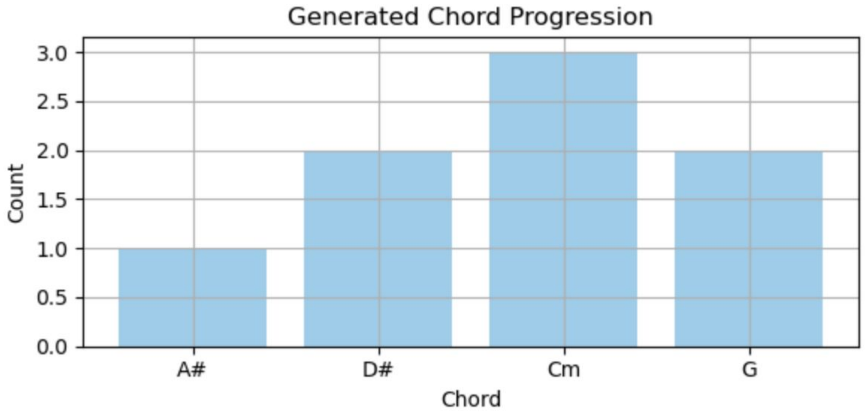
- A simple baseline is a model that does not separate chords and melodies, treating the entire sequence as a flat stream of notes.
- Such models are hard to interpret, since it's unclear which parts are harmonic (chords) and which are melodic.
- It is also impossible to extract or control chord progression explicitly from this type of model.

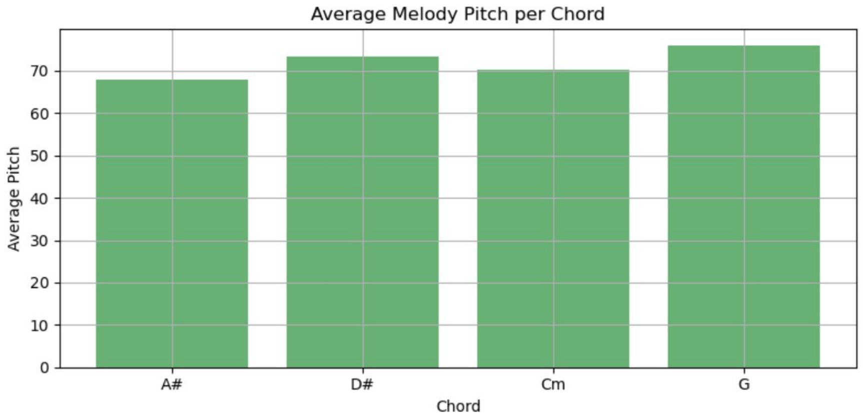
How do you demonstrate that your method is better than these trivial methods?

- Our method explicitly separates chord and melody generation, allowing us to model musical structure more transparently.
- By using a Markov chain for chord progression and an RNN for melody, we ensure musical consistency and interpretability.

1.3 Code


Generated Melody Note Distribution
Count
MIDI Pitch
0
1
2
3
4
5
50
55
60
65
70
75
80
85


Generated Chord Progression
Count
Chord
0.0
0.5
1.0
1.5
2.0
2.5
3.0
A#
D#
Cm
G


Average Melody Pitch per Chord
Average Pitch
Chord
0
10
20
30
40
50
60
70
A#
D#
Cm
G

1.4 Discussion of related work

How has this dataset (or similar datasets) been used before?
• The MAESTRO dataset has been widely used for symbolic music generation tasks, especially for training large language models or sequence models on piano performances.

How has prior work approached the same (or similar) tasks?
• Previous studies often treat music generation as a language modeling problem, training RNNs, LSTMs, or Transformers to predict the next token (note or chord) in a sequence.
• Some models generate end-to-end compositions, while others use conditional inputs (e.g., style, tempo, key).

How do your results match or differ from what has been reported in related work?
• Unlike many previous approaches, we **separate chord and melody streams**, modeling each differently: chords with a Markov chain and melody with an RNN.
• This separation helps maintain **clear harmonic structure** and allows us to enforce a **fixed ratio (1 chord : 4 notes)**, which is often missing in end-to-end generation.
• Our approach is simpler but interpretable, and the outputs produce musically coherent output without requiring complex architectures.

2. Symbolic, Conditioned Generation

2.1 EDA, Data Processing

2.1 Context: Where does your dataset come from and how's it collected?

- Collected by Eric Foxley from live folk-music sessions; refined by Jay Glanville, Seymour Shlien, and others.
- Contains ~1,200 tunes with melody + chord annotations (jigs, reels, hornpipes, etc.).
- Stored in ABC notation: plain text format with metadata and musical body.
- Freely available on SourceForge (GPL-3.0).

2.1 Context: What is it for?

- ABC is human-readable and scriptable.
- Chord + melody pairs allow multi-instrument symbolic learning.
- Structure of ABC makes it easy to parse, tokenize, and transpose.
- Widely used in symbolic generation and music information retrieval tasks.

2.1 Discussion: Preprocessing

1. Conversion & Cleanup
   - Original folk tune texts were converted to ABC format using NMD2ABC scripts.
   - Additional manual fixes corrected issues like missing beats, bracket mismatches, and malformed rests.
   - Jukedeck's cleaned fork of the dataset, which standardizes chord syntax (e.g., C/e), repeat brackets, and part labels.
2. Validation & Metadata
   - Ensured every file has X: (index), T: (title), M: (meter), L: (note length), and K: (key).
   - Explicitly handled repeat endings ([1, [2]) so abc2midi parses correctly.
3. MIDI Conversion & Splits
   - Converted ABC → ~1,034 MIDI via abc2midi into MIDI format.
   - Train/validation/test: first 800, next 100, final 100 MIDI files.

2.1 Discussion: Preprocessing Continued

1. Alignment & Filtering for ML
    - music21: parse() ABC → Score → chordify() → extract:
        - Melody: note.Note names
        - Harmony: Chord.root()
        - Bass: Chord.bass() or root octave.
    - Discard tunes with <16 events ("⚠️ … skipped: not enough notes").
    - Skip files if parse() >20 s ("⏱️ Skipping … took too long").
2. Integration Notes
    - The Nottingham corpus can be used standalone (~1,000 usable tunes) or combined with other ABC sources.
    - ABC→MusicXML/LilyPond conversion verifies headers match MIDI before modeling.
3. Data Augmentation
    - Transposed each sequence ±1 and ±2 semitones using music21.
    - Augmented melody, chords, and bass independently to preserve multi-track alignment.
    - Increased total data ~5×, improving generalization across keys.

2.1 Code: EDA

Number of tunes: 14

Note count - mean: 11126.6, std: 12818.9

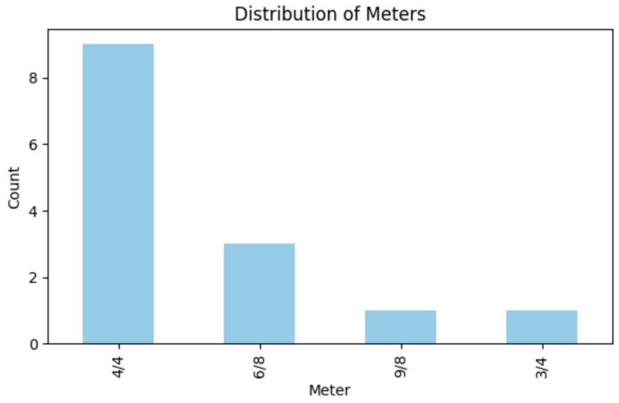
Min: 1161.0

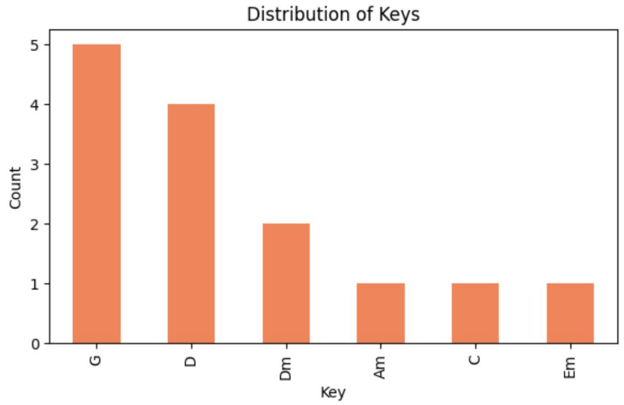
25th pct: 4568.0

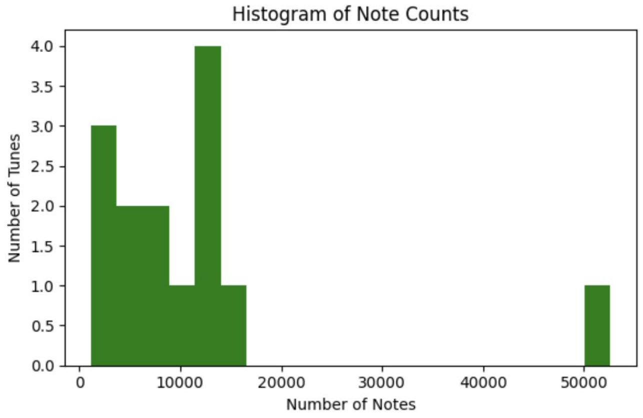
50th pct: 9027.0

75th pct: 12939.75

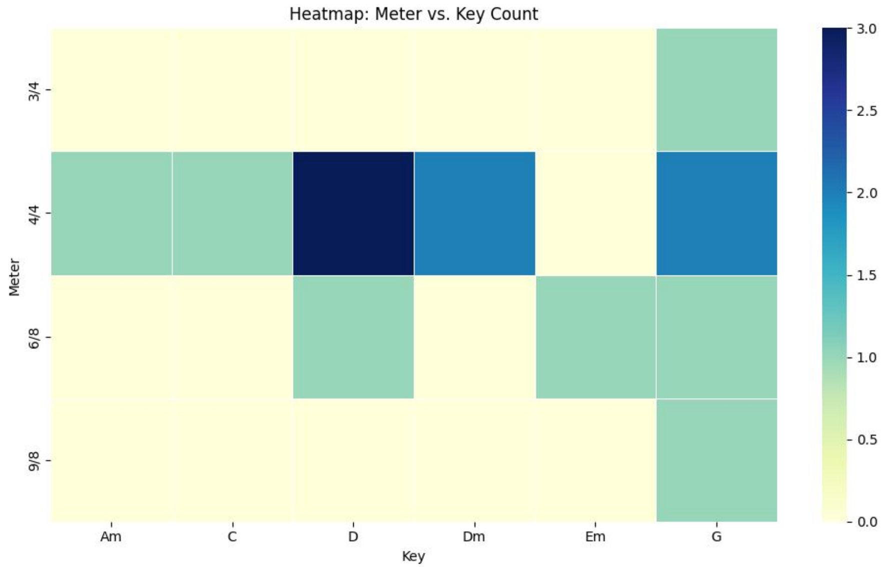
Max: 52614.0

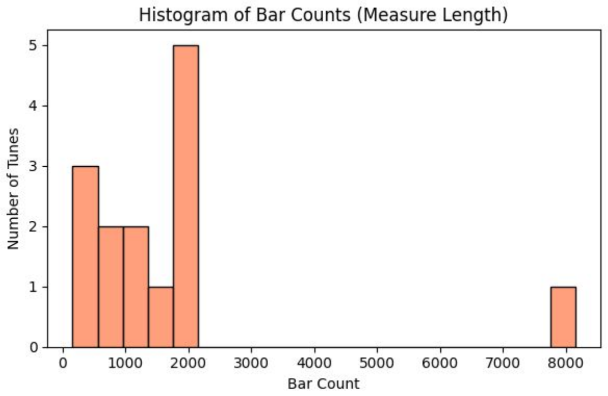


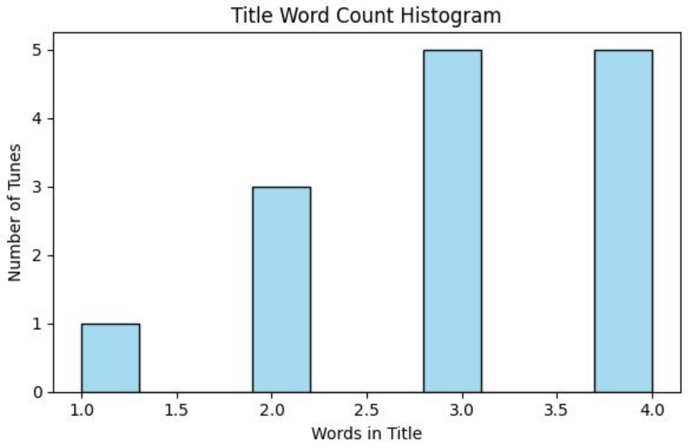




2.1 Code: EDA Continued







2. Symbolic, Conditioned Generation

2.2 Modeling

2.2 Context: Task Formulation

- Task: Predict next REMI token in a sequence combining melody, chord, and bass.
- Input:
  - Sequence of REMI tokens (e.g. Track_Melody, Note_C4, Track_Chords, Chord_G, …) up to length N.
- Output:
  - A continuation of symbolic tokens, conditioned on the previous segment (e.g., intro → climax → resolution).
- Optimization Objective:
  - Trained using CrossEntropyLoss over predicted tokens.
  - Model learns to predict y_t given x_0, ..., x_{t-1} (autoregressive token modeling).
- Model Type:
  - A decoder-only Transformer (MiniTransformer) with embedding + positional encoding.
  - Structured conditioning is handled via generation logic (not via additional input channels).

2.2 Discussion: Model Selection

- REMI-Token Transformer
  - Pros:
    - Single stream captures melody↔harmony↔bass.
    - Self-attention handles long dependencies.
  - Cons:
    - Large token vocab (hundreds of unique tokens) → expensive softmax.
    - $O(N^2)$ attention cost during training and generation.
- Feature-Based MLP (Pitch/Duration/Interval)
  - Pros:
    - Smaller numeric inputs → smaller models, less overfitting.
    - Easier to inject musical priors (pitch ranges, tempo).
  - Cons:
    - Lacks symbolic token structure.
    - Lower generative quality unless explicitly decoded to REMI again.

2.2 Code: Symbolic Generation

1. transpose_sequence() + Augmentation

- Transposes melody/chords/bass ±1 or ±2 semitones.
- Boosts data ~5× while preserving structure.

2. build_token_sequence() + generate_sequence()

- Converts melody, chords, and bass into REMI-style token sequences.
- Samples tokens autoregressively using a Transformer decoder..

3. structured generation logic

- Manually splits output into Intro (original), Climax (+5 semitones, thinned), Resolution (−4 semitones).

2.2 Code: Expressive Decoding

4. extract_duration_features() → duration_model
   - Trains an MLP to predict note duration from token features (track, pitch, position)
5. extract_velocity_features() → velocity_model
   - Predicts loudness (MIDI velocity) for each note
6. extract_tempo_features() → tempo_model
   - Predicts tempo (BPM) per bar using bar index as input
7. decode_to_midi()
   - Converts tokens to .mid, applying all predictions above, plus rhythmic swing and rubato

2. Symbolic, Conditioned Generation

2.3 Evaluation

2.3 Context

- How should your task be evaluated?
  - Whether the chord progression is plausible and musically coherent.
  - Whether the melody and chords are in alignment.
- What should be the properties of a “good”output?
  - 4 notes per chord
  - Controlled Dissonance, < 15% non-chord tones
- What is the relationship between the objectives being optimized by your model and musical properties? versus subjective properties?
  - Our model optimizes perplexity and accuracy, which control musical properties such as the chord progression rules.
  - Our model optimizes the subjective property of folk using REMI tokens.

2.3 Discussion

- What are some baselines(trivial or otherwise) for your task?
  - One baseline is to randomly select the notes and chords
  - Another trivial baseline is to alway predict the most common token.
- How do you demonstrate that your method is better than these trivial methods?
  - By implementing these other methods, and then comparing and contrasting the musical coherence, folk authenticity, and other metrics, against our model.

2.4 Discussion Of Related Work

- How has this dataset(or similar datasets) been used before?
    - The Nottingham dataset has been used for symbolic music generation, including for genre transformation, and automatic harmonization generation.
- How has prior work approached the same(or similar) tasks?
    - Most previous work generates short segments, while ours generates a lengthy sequence.
- How do your results match or differ from what has been reported in related work?
    - Ours separates into specific sections of intro/climax/resolution.
    - Combines REMI tokenization with expressive feature prediction.

Thank you!