Harmony Completion - Trie-based Autocompletion

Ahcene LOUBAR

LIP6 - SU UPMC

July 2025

Dataset Selection

I explored multiple MIR datasets from the GitHub repository:

`https://gist.github.com/alexanderlerch/e3516bffc08ea77b429c419`

Among them, three stood out:

- **MusicBench** – rich harmony + audio
- MAESTRO – MIDI-aligned piano dataset
- GuitarSet – pitch, beat, chords, hexaphonic audio

I selected **MusicBench** for its detailed harmonic content and clean structure.

MusicBench Sample Entry

Each song in **MusicBench** is stored as a JSON object.

Below is a simplified view of one entry from the test set:

Example Document

```
{
  "location": "data_aug2/-0SdAVK79lg_1.wav",
  "bpm": 112,
  "key": ["E", "major"],
  "chords": ["E"],
  "caption": "mellow guitar-driven music with coffee shop vibe",
  ...
}
```

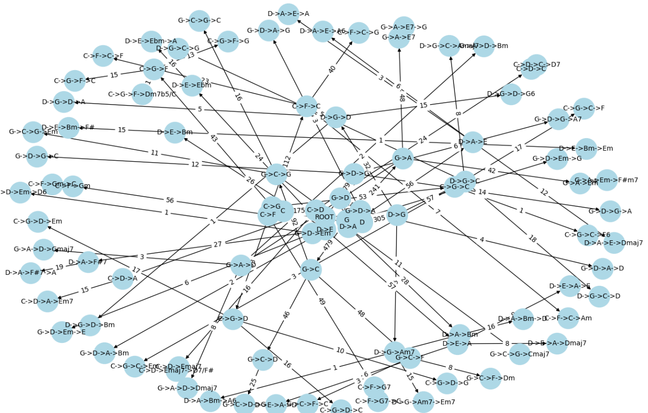
Note: In this work, we focus exclusively on the c̈hords¨ field to extract and learn harmonic structures via prefix trees (tries).

Harmony Trie: Concept

We represent all chord progressions as prefix trees:

- Nodes = chords
- Paths = observed chord sequences
- Frequencies stored as counts

Harmony Trie (limited depth and branching)



Pseudocode: Insert Chord Progression

Algorithm 1 Insert

```
procedure INSERT(sequence: List of Chords)
    node ← root
    for chord in sequence do
        if chord not in node.children then
            node.children[chord] ← new TrieNode
        end if
        node ← node.children[chord]
        node.count ← node.count + 1
    end for
    node.is_end ← true
end procedure
```

Pseudocode: Autocomplete Chord Sequence

Algorithm 2 Autocomplete

```
procedure AUTOCOMPLETE(prefix: List of Chords)
    node ← root
    for chord in prefix do
        if chord not in node.children then
            return $\emptyset$
        end if
        node ← node.children[chord]
    end for
    return COLLECTCOMPLETIONS(node, prefix)
end procedure
```

Pseudocode: CollectCompletions

Algorithm 3 CollectCompletions

```
procedure CollectCompletions(node, path)
    results $\leftarrow$ []
    if node.is_end then
        Append $(path, node.count)$ to results
    end if
    for (chord, child) in node.children do
        subresults $\leftarrow$ CollectCompletions(child, path + [chord])
        Append subresults to results
    end for
    return results
end procedure
```

Minimal Python Test

- Insert: `trie.insert(["E", "B", "A"])`
- Autocomplete: `trie.autocomplete(["E", "B"])`

Minimal Python Test

- Insert: `trie.insert(["E", "B", "A"])`
- Autocomplete: `trie.autocomplete(["E", "B"])`

Result

```
[["E", "B", "A"], 2]
```