Quantifying Genre Patterns in Music Through Huffman Compression

Anthony Casas, Teo Maayan, Emily Ren, Lulu Wang, Matin Kositchutima

Introduction

Music genres differ widely in structure, repetition, composition, and overall style, yet these differences can be difficult to measure in a consistent and quantitative way. In this project, we study these patterns using Huffman coding, a lossless compression method that assigns shorter bitstrings, or 'codes,' to more frequent symbols (Huffman 1952). By observing how music compresses, we analyze these symbolic representations of songs to compare trends within and across genres.

Our approach is based on the idea that Huffman compression behavior reflects the underlying musical structure, meaning that if a genre contains repeated patterns, such as pop and EDM, we would expect it to compress more efficiently than genres with greater variations, such as jazz. This leads to the central questions guiding our project:

1. How do compression ratios differ between genres?
2. How similar are the resulting Huffman codes across genres?
3. Do specific musical characteristics, such as repetition or vocal presence, affect compression?
4. How similar are songs within the same genre when using one song's Huffman code to compress another, do they yield similar codes? How does this relationship change across different genres?

The questions above have several real-world applications. First, compression-based comparisons may offer an empirical approach to examining genre similarities without subjective labels or traits. Second, the ratios of compression can inform storage estimates for large-scale music datasets. Finally, observing patterns in code structures may be used to develop automated genre classification methods.

In this project, we anticipate genres with highly repetitive patterns, such as pop and electronic, to produce better compression ratios, and instrumental songs to compress more efficiently than songs with lyrics, due to less variation. The rest of the paper will be organized as follows:

1. Methodology
2. Limitations
3. Implementation
4. Results and Analysis
5. Discussion and Applications
6. Conclusion
7. References

1 Methodology

In this project, we utilize the ADL Piano MIDI dataset of 11,086 piano pieces from different genres. This data originates from the Lakh MIDI "Piano Family" dataset and other publicly-available sources on the internet (Lucas 2019). The selection of MIDI files was intentional, as they're able to encode music through pitch, timing, and velocity. Because Huffman coding requires the selection of discrete symbols, we may compute chords easily to construct a semantically meaningful base unit used to generate Huffman codes, and thus to identify trends both across and within genres. Each musical piece can be represented as a finite sequence of random variable:

$$(X_1, X_2, \ldots, X_n), \quad X_k \in \mathcal{S},$$

where $\mathcal{S}$ denotes the alphabet of chord symbols. This allows us to compute the symbol frequencies

$$p_i = \frac{1}{n} \sum_{k=1}^{n} \mathbf{1}_{\{X_k = s_i\}}.$$

Because musical genres differ in their use of chords, rhythm, and repetition, the frequency distribution of chord symbols is expected to vary across genres. This motivates the use of Huffman encoding as a quantitative tool for surfacing structural similarities and differences across genres.

This representation enables the direct application of Huffman coding. The Huffman coding algorithm generates a prefix-free binary code $\mathcal{C} = \{c_1, \ldots, c_n\}$ which yields minimum expected codeword length

$$\mathbb{E}[L] = \sum_i p_i \, \ell_i$$

given the probability mass function $\{p_i\}$ of each symbol (musical notes in our case), where $p_i = \mathbb{P}(X = s_i)$ and ℓ_i denotes the length of the corresponding codeword. The prefix-free property of Huffman coding allows each codeword to be decoded without ambiguity, removing the need for delimiters. In addition to the mean codeword length, the variability of the encoding is quantified by

$$\mathrm{Var}(L) = \mathbb{E}[L^2] - (\mathbb{E}[L])^2 \tag{1}$$

$$= \sum_i p_i \, \ell_i^2 - \left(\sum_i p_i \, \ell_i \right)^2 \tag{2}$$

After running the algorithm for each piece, the compression ratio

$$\mathrm{CR} = \frac{\text{compressed length}}{\text{original length}}$$

will serve as a quantitative indicator of the piece's musical profile, which could be compared to draw meaningful conclusions.

2 Limitations

Our approach is subject to several limitations surrounding data representation, dataset composition, and musical scope. First, the use of MIDI data restricts the dimensions of music that we may analyze. Because they primarily encode pitch, timing, and velocity, this representation omits additional nuances. While MIDI data is well-suited for Huffman coding due to its discrete symbolic structure, this abstraction may oversimplify musical characteristics (Cataltepe 2005). A related limitation is that the dataset consists exclusively of piano MIDI files. As a result, genres or songs in which vocals play a central role cannot be directly analyzed. This constrains the generalizability of our findings, especially when comparing genres where lyrics or vocal variations serve as a defining feature.

The dataset also presents a notable imbalance across genres, with some being heavily represented and others containing relatively few pieces. The smallest genre in our dataset contains only 24 pieces, whereas the largest contains approximately 1,400. In particular, classical music dominates the dataset, accounting for roughly 35% of all files. This imbalance may introduce bias and limit the statistical soundness of cross-genre comparisons, especially for the underrepresented genres such as children's music and reggae.

Finally, the dataset contains minor metadata inaccuracies. For instance, "Twinkle, Twinkle, Little Star" is attributed to Mozart. Though a commonly mistaken attribution, and such errors do not directly affect compression results, they highlight potential issues with dataset labeling.

3 Implementation

Our code implementation is based on the coding method proposed by Huffman (1952). The code implementation is modeled after the pseudocode presented in "Huffman Coding" (2025). Some of the methods are also inspired by Cataltepe et al's (2005) work on music classification using another mathematical measure (Kolmogorov distance). The full implementation details are available in our GitHub repository:

https://github.com/teomaa/genre-huffman-coding

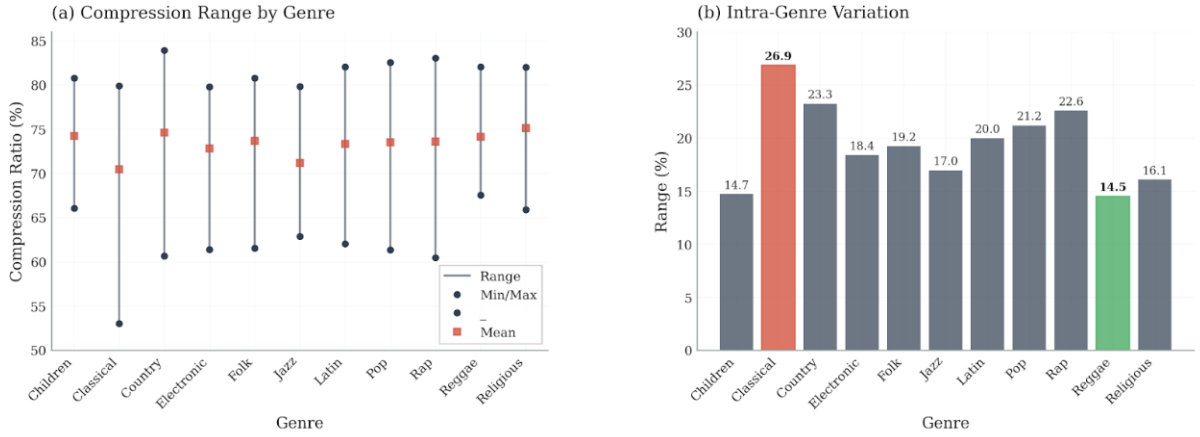
The following pseudocode presents a high-level overview of our Huffman compression pipeline applied to an individual song.

```
ALGORITHM ExtractIntervals(midi_file):
    notes <- []
    FOR each instrument IN midi_file.instruments:
        FOR each note IN instrument.notes:
            notes.APPEND((note.start, note.pitch))
    SORT notes BY start_time
    pitches <- [pitch FOR (_, pitch) IN notes]
    intervals <- DIFF(pitches)              # pitch[i+1] - pitch[i]
    RETURN intervals

ALGORITHM BuildHuffmanTree(intervals):
    freq <- COUNT(intervals)
    PQ <- empty min-heap
    FOR each (val, count) IN freq:
        PUSH(PQ, Node(value=val, freq=count))
    WHILE SIZE(PQ) > 1:
        a <- POP_MIN(PQ)
        b <- POP_MIN(PQ)
        PUSH(PQ, Node(value=None, freq=a.freq+b.freq, left=a, right=b))
    RETURN POP_MIN(PQ)                      # root

ALGORITHM GenerateCodes(node, prefix="", codes={}):
    IF node.value IS NOT None:
        codes[node.value] <- prefix
        RETURN
    IF node.left  =/= NULL: GenerateCodes(node.left,  prefix+"0", codes)
    IF node.right =/= NULL: GenerateCodes(node.right, prefix+"1", codes)
    RETURN codes

ALGORITHM EncodeIntervals(intervals, codes):
    bits <- ""
    FOR x IN intervals:
        bits <- bits  codes[x]
    RETURN bits

ALGORITHM DecodeBits(bits, root, expected_len):
    vals <- []
    node <- root
    IF root.value IS NOT None:              # single-symbol tree
        RETURN ARRAY_FILL(expected_len, root.value)
    FOR bit IN bits:
        node <- node.left  IF bit="0" ELSE node.right
        IF node.value IS NOT None:
```

```
        vals.APPEND(node.value)
        node <- root
RETURN vals
```

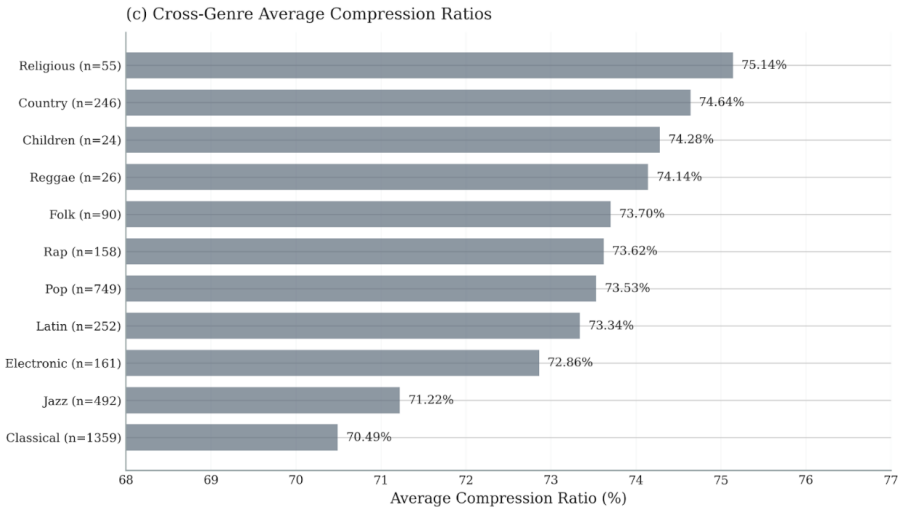
4 Results and Analysis

Figures 1 and 2



We first examine the intra-genre variation in compression ratios, shown in Figures 1 and 2. Among all genres, classical music exhibits the widest internal range, with a difference of 26.88%. Mozart's "Twinkle Twinkle Little Star" achieves the highest compression ratio (79.91%), while Tchaikovsky's "Waltz of the Flowers" yields the lowest (53.03%). This large spread with our musical intuition that pieces built on highly predictable harmonic patterns compress more efficiently, whereas works with greater harmonic complexity result in lower compression rations. In contrast, Reggae displays the narrowest internal range, with a difference of 14.52%. However, this result must be interpreted cautiously due to the genres small sample size of 26 pieces. Most other genres produce an internal variation between 20% and 23%.

Figure 3



Next, we analyze cross-genre variation using the average compression ratios across 13 genres, shown in Figure 3. Overall, the range of mean compression ratios is relatively small, spanning only 4.65% (from 70.49% to 75.14%). Religious music (75.14%), country (74.64%), and children's music (74.28%) show the highest average compression ratios, indicating that more of their chords could be compressed, resulting from greater repetition.

This is consistent with the common structural features of these genres, such as repeated hymns, verse-chorus structures, and simple, recurring melodies. Conversely, classical (70.49%) and jazz (71.22%) show the lowest mean compression ratios, reflecting higher variability associated with complex compositional forms and improvisation. Despite these differences, this narrow spread in ratios suggest that genres share similar levels of compressibility when represented using piano-only MIDI pieces.

Beyond average comparisons, song-level cross-genre compression experiments reveal cases where genre labels diverge from musical structure. By compressing each song using Huffman codes derived from other genres, we find instances where pieces align more closely with genres different from their assigned labels. Several of the strongest examples occur within the classical genre, with Yannis Marching Season, for instance, compressing 19.8% more efficiently with a blues-derived Huffman code than a classical one, ranking fifth overall in our dataset. This aligns with human listening perception: the piece uses shifting time signatures (9/8, 10/8, 4/4), triplet figures, dynamic shifts, and energetic rhythms that are more characteristic of blues shuffle patterns than conventional classical pieces.

Similar occurrences appear outside the classical genre, for example, the soundtracks genre also shows extreme cross-genre cases. The most extreme case being Sha Na Nas Yummy, Yummy, Yummy, which compresses 23.1% better as country than as soundtracks. This result is musically intuitive as the 1960s bubblegum pop cover is built on simple IIVV progressions and repetitive versechorus structures characteristic of mainstream country styles, rather than varied orchestral arrangements typical of film scores.

While these examples demonstrate individual instances of cross-genre compatibility, we next observe this pattern at a broader genre-wide level. Specifically, we observe the cross-compression rates, the ratio of songs within a genre that compress more efficiently under at least one other genres Huffman code. These rates are shown in Figure 4.

The results reveal a significant degree of cross-genre overlap. For example, 72% of the genres tested (13 out of 18) have more than half their songs compress more efficiently using codes from other genres. Though the average improvement is relatively small, approximately 1%, as shown in Figure 5, this pattern implies that genre boundaries arent entirely discrete, with many genres sharing statistical similarities that allow for shared encodings.

Interestingly, some genres stand out as outliers with notably high or low cross-compression rates. For example, jazz yields a particularly low cross-compression rate (31.5%), indicating that most jazz songs compress best under jazz-derived codes. This is consistent with jazz's distinct harmonic language and rhythmic variability, including extended chords and frequent improvisation, features less captured by statistical patterns of other genres. This observation aligns with jazzs overall compression rate, being the second-lowest of all genres, as seen in Figure 3. In contrast, several genres exhibit especially high compression rates, including latin (82.2%), pop (84.4%), unknown (86.5%), world (82.3%), and rock (82.3%). This is a likely result of the broad nature of the categories themselves, where placement seems to be subject to social intuition rather than objective stylistic grouping. For example, the pop genre simply refers to popular music, and rock is an internally heterogeneous genre with various subgenres. These figures seem to suggest that these genres feature a greater number of borrowed techniques from other genres, consistent with their broad labeling.

Figure 4

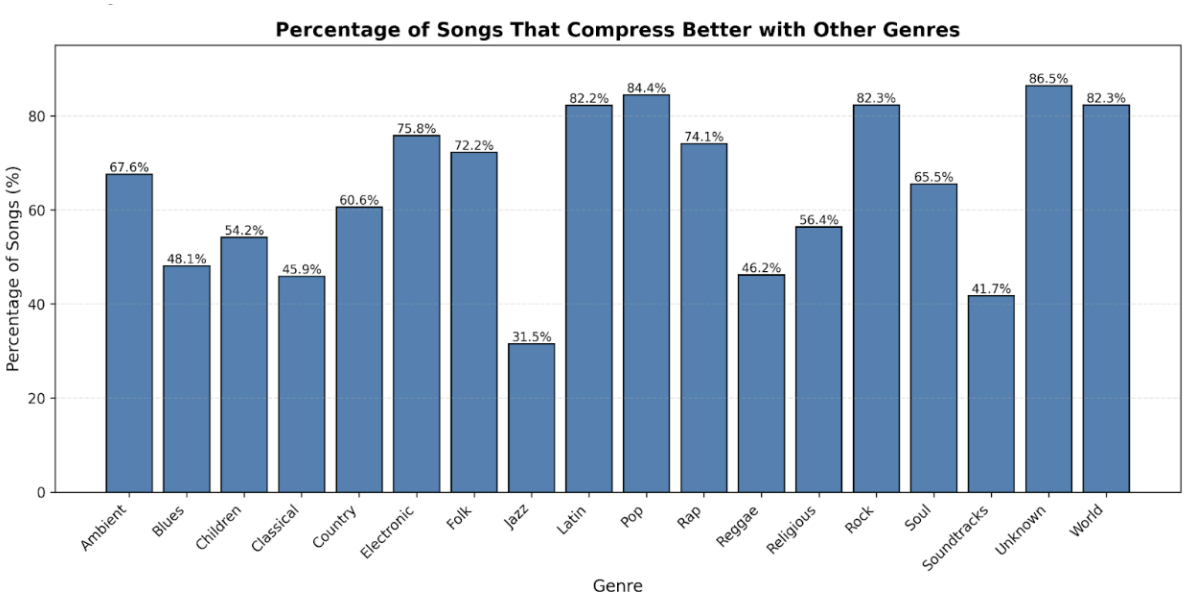


Figure 5

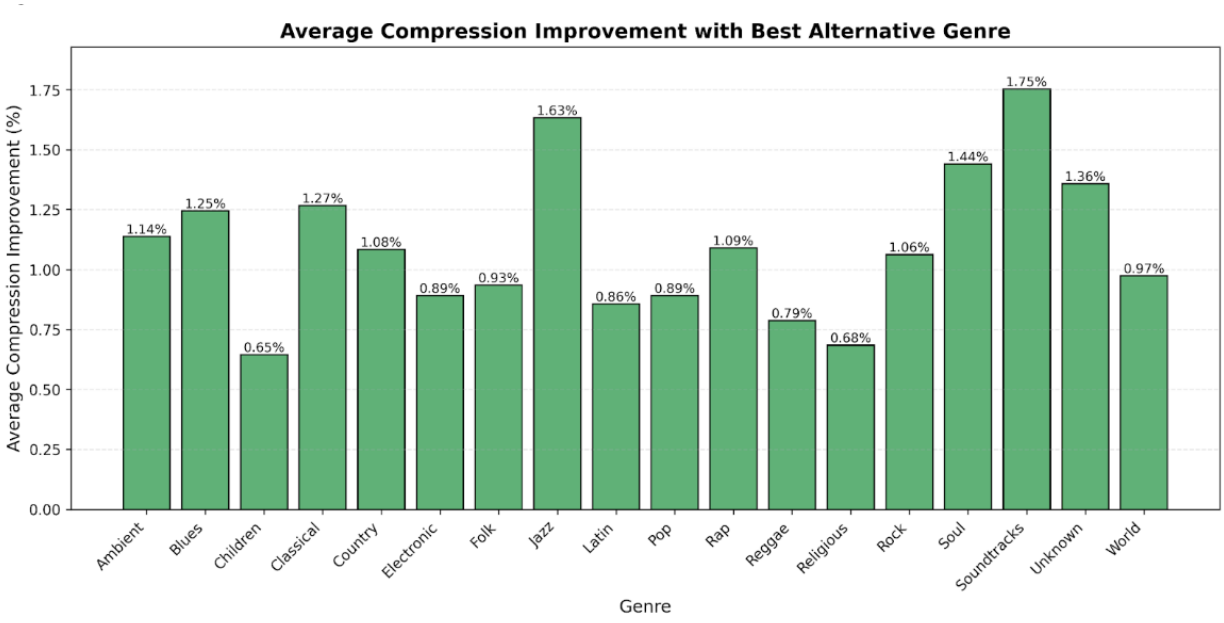
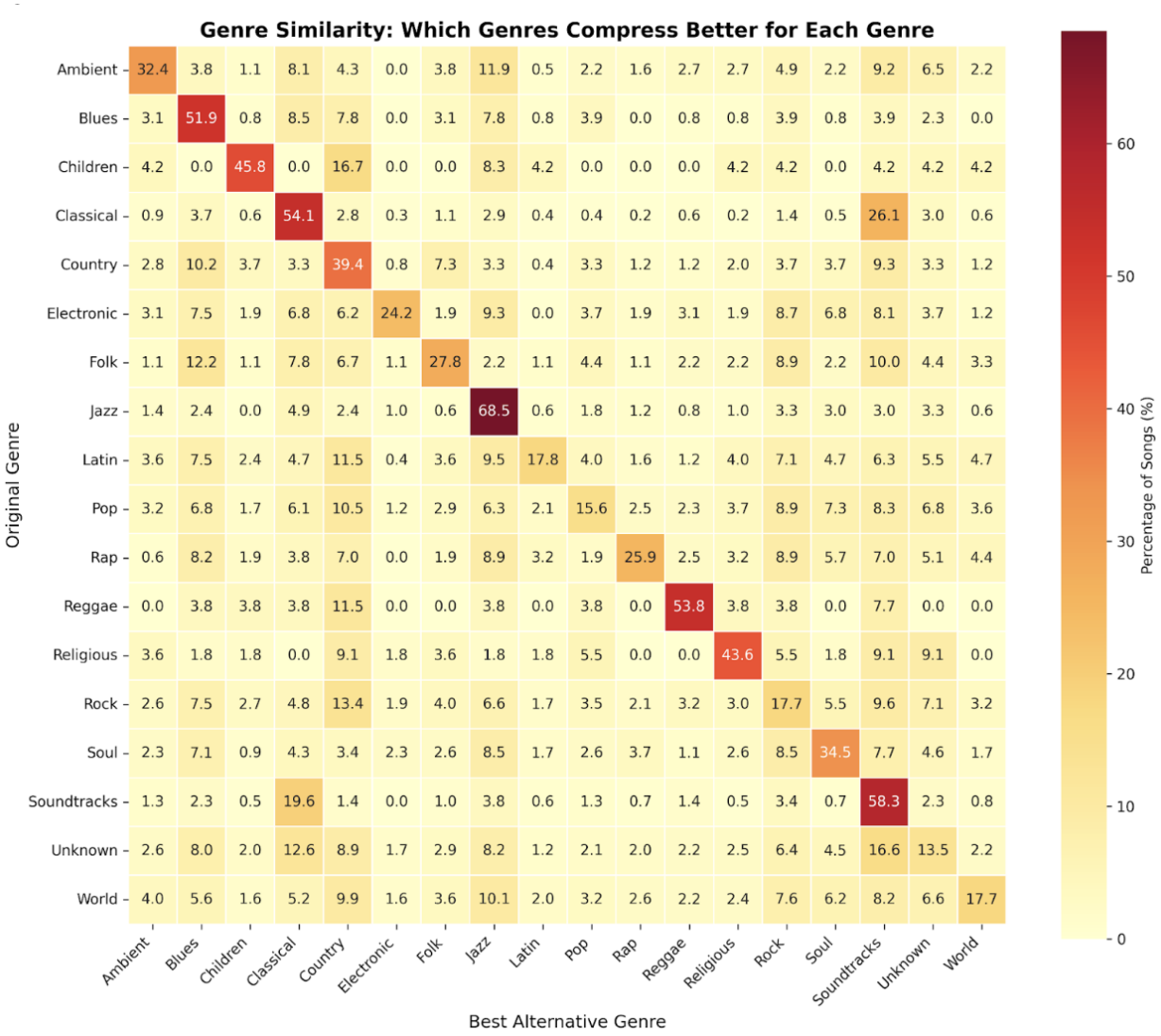


Figure 6 presents a heatmap illustrating cross-compression relationships between genres. The data reveals that these relationships are asymmetric, where a Huffman code derived from genre A may compress genre B more efficiently than genre B compresses genre A. Additionally, multiple genres may compress a given song more efficiently than its labeled genre. The entries in Figure 4 correspond to the inverse of the diagonal entries in Figure 6.

We use this heatmap to observe pairings between different genres. For example, pop shows a particularly weak self-alignment, with most songs compressing better under country, rock, or soundtracks, suggesting that pops harmonic attributes overlap strongly with these genres. From this, we interpret that many pop

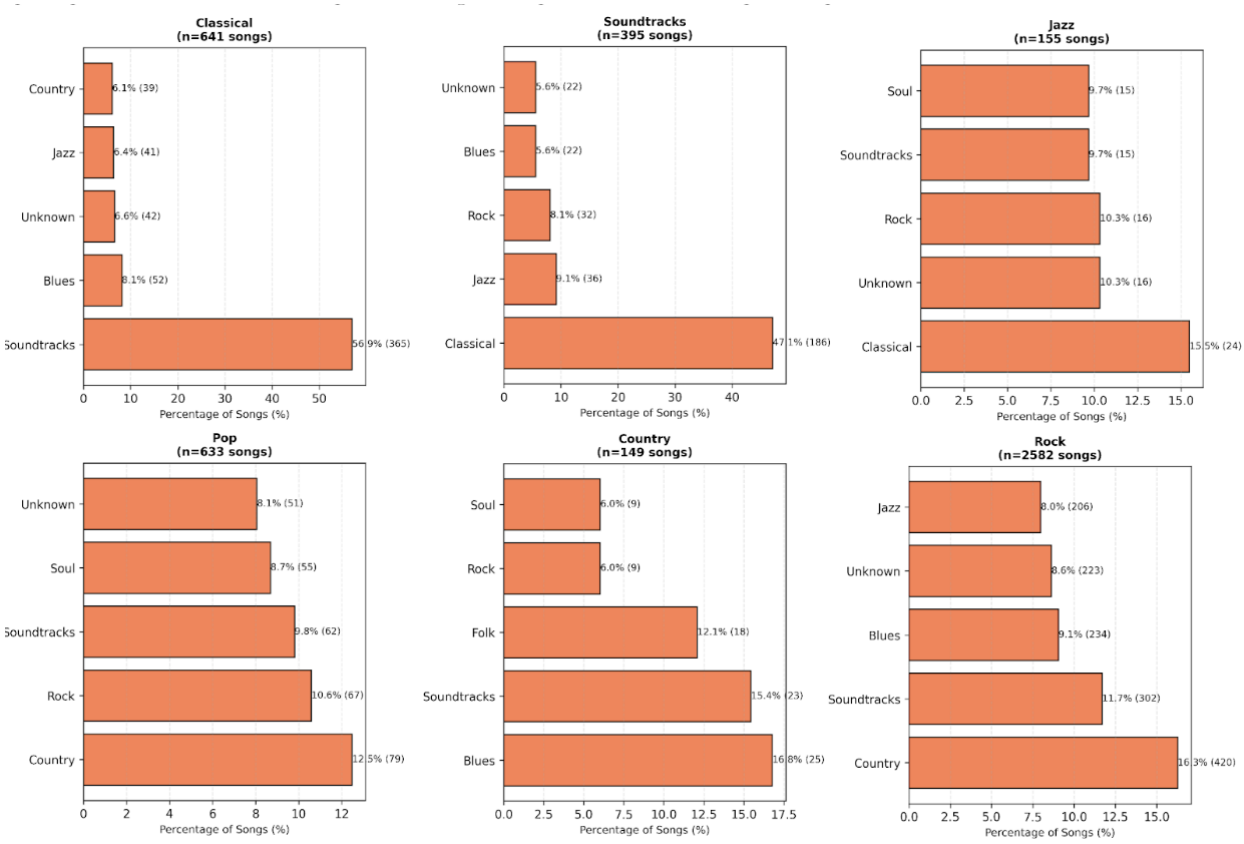
songs may be a subgenre of, or structurally related to, these genres. Electronic music also stands out, as a majority of songs compress better under non-electronic codes derived from jazz, classical, or soundtracks. This likely reflects limitations of the piano-only MIDI representation, which may not adequately represent features central to electronic music, which seldom makes use of traditional instrumentation. A strong relationship also emerges between classical and soundtracks (classical → soundtracks: 26.1%, soundtracks → classical: 19.6%). This is unsurprising, as soundtrack music often *is* classical music, and as such shares its tonal structures and harmonic progressions.

Figure 6



Finally, Figure 7 shows the most common alternative genres under which songs compress more efficiently for a selected subset of genres. While this figure overlaps conceptually with Figure 6, it still offers valuable insights. In particular, country emerges as the most common alternative genre for both pop and rock songs, suggesting structural alignment. The relationship between classical and soundtracks once again becomes prominent, and jazz continues to appear self-contained, with its closest alignment occurring with classical music, rather than more contemporary genres.

Figure 7: Most common other genres compressing better than a songs real genre.



5 Discussion and Applications

Given the results from the previous section, we motivate the discussion of both the implications and potential applications of compression-based analysis. In particular, the observed patterns in compression ratios suggest that Huffman coding captures aspects of musical structure related to repetition and variability, while also demonstrating limitations in how musical genres are represented through piano-only MIDI data.

One practical application of this compression method is the use of its resultant ratios as a proxy for musical complexity. Since the compression ratios reflect the regularity of chord sequences, they may serve as a quantitative indicator of structural irregularities. Such metrics could be incorporated into music recommendation systems to distinguish between songs with differing levels of repetition, allowing users to explore music aligned with their preferences or listening history. Similarly, this measure could support music education tools, where pieces are automatically grouped or ranked by estimated complexity to guide practice selection.

In this discussion, it is important to note that the relatively narrow range of compression ratios across genres may be a result of the piano-only MIDI representations that potentially abstract genre-specific features. While certain genres exhibit higher or lower mean compression ratios consistent with musical intuition, elements such as electronic layering, orchestral variety, and studio effects, are not captured by this representation, which can possibly limit genre differentiation at an aggregate level.

More broadly, the cross-genre compression experiments and aggregate cross-compression rates indicate that compression-based analysis can highlight structural similarities that do not necessarily align with the assigned genre label. At the song level, individual pieces frequently align more closely with other genres, while at the genre level, a majority of genres contain many songs that compress more efficiently under codes

derived from other labels. Together, these patterns emphasize the distinction between genre as a cultural category and genre as a reflection of underlying composition. With this, Huffman-based compression offers a computational approach for investigating genre definition and overlap.

There is also a broader cultural bias inherent in the dataset. Non-Western musical traditions that rely on microtonal intervals, complex rhythms, or non-chord-based structures may not be well represented or accurately captured in MIDI format. For example, within the pop genre, Jacky Cheungs Mei Tian Ai Ni Duo Yi Xie exhibits a notably low compression ratio of 61.35%, a possible reflection of Cantopops distinct pentatonic scales and melodic ornamentations which do not align with Western harmonic conventions.

6 Conclusion

The variability and subjectivity of music genres make them challenging to analyze objectively and to quantify in a consistent manner. In this project, we use Huffman coding with musical chords as symbols to compute compression ratios, providing an objective, algorithmic measure that can be compared across genres, while reducing reliance on subjective judgment. We hypothesized that differences in musical structure across genres would be reflected in differences in compression behavior, and we tested this by comparing compression ratios both within and between genres.

Within genres, compression ratios vary most in classical music and least in reggae, suggesting greater structural diversity among the classical pieces in our dataset and more similarity within reggae. Across genres, the range of mean compression ratios is relatively small at 4.65%, with classical music yielding the lowest average compression ratio and religious music the highest. Although this range is modest, the overall pattern is consistent with the intuition that different genres exhibit distinct note distributions and structural patterns that influence how efficiently they compress. In addition, cross-genre compression results show that many songs, and even entire genres, often align with genres other than their assigned labels.

Taken together, applying Huffman encoding to musical chords across multiple genres offers quantitative examination of genre-level differences in musical structure While this approach is limited by the dataset used, it provides a useful perspective on musical organization and suggests possibilities for further analysis.

7 References

Cataltepe, Z., Sonmez, A., & Adali, E. (2005). Music genre classification using MIDI and audio features. Istanbul Technical University. https://web.itu.edu.tr/~cataltepe/pdf/2005_MusicConf_Abdullah.pdf

Goulart, A. J. H., Maciel, C. D., Guido, R. C., Paulo, K. C. S., & da Silva, I. N. (2011). Music genre classification based on entropy and fractal lacunarity. In 2011 IEEE International Symposium on Multimedia (pp. 533536). IEEE. https://doi.org/10.1109/ISM.2011.94

Huffman, D. A. (1952). A method for the construction of minimum-redundancy codes. Proceedings of the IRE, 40(9), 10981101. https://doi.org/10.1109/JRPROC.1952.273898

Ferreira, L. N. (2019). ADL Piano MIDI (Version bdb4836). GitHub. https://github.com/lucasnfe/adl-piano-midi/tree/master

Ferreira, L. N., Lelis, L. H. S., Whitehead, J. (2020). Computer-Generated Music for Tabletop Role-Playing Games. University of California, Santa Cruz, USA. https://doi.org/10.48550/arXiv.2008.07009

Ma, Suiliang (n.d.). The mathematics and physics of music compression. Rutgers University Mathematics Department. https://sites.math.rutgers.edu/~zeilberg/math436/projects/MaP.pdf

Raffel, C., & Ellis, D. P. W. (2014). Intuitive analysis, creation and manipulation of MIDI data with pretty_midi. Proceedings of the 15th International Society for Music Information Retrieval Conference (IS-

MIR). https://colinraffel.com/publications/ismir2014intuitive.pdf

Raissi, R. (2002, December) MP3 theory and technical information. .http://www.mp3-tech.org/programmer/docs/mp3_theory.pdf

Kartik. (2025, July 23). Huffman coding: Greedy Algo-3. GeeksforGeeks. https://www.geeksforgeeks.org/dsa/huffman-coding-greedy-algo-3/

NAMES + UNIS:

- Anthony Casas (ac5473)
- Teo Maayan (tmm2203)
- Emily Ren (sr4366)
- Lulu Wang (yw4240)
- Matin Kositchutima (mk5155)