Residual Entropy Encoding in the Frequency Domain via Neural Network Predictions

Niccolo Abate Chris Relyea Gabriel Soule

March 2026

Contents

1 Introduction

An audio codec is a (collection of) programs that facilitate the encoding and reconstruction of digital audio. Typically, a perceptual audio codec[1] is designed to encode audio at the lowest possible bitrate while maintaining an acceptable level of perceptual quality to the listener. As such, a codec may be viewed as a lossy compression algorithm designed with respect to the psychoacoustic properties of audio, as interpreted by human listeners. The rich optimization space of codec design has been the subject of many years of fruitful research.

Most digital signals are not uniformly random: a useful signal carries, almost by definition, information, and information is often structured across time and frequency. Musical signals, for instance, are often primarily composed of harmonically related sinusoids, and (most) music demonstrates repeated structure across time. Two techniques that exploit such structure are *entropy coding* and *predictive coding*. Entropy coding exploits the nonuniform distribution of symbols in a signal by assigning shorter codes to symbols that appear more frequently in the data. Predictive coding is concerned with the design and use of an oracle that can predict the next symbol in a signal.

In this work, we experiment with both of these techniques. We implement entropy coding using a Huffman code. Our predictive coding pipeline encodes residuals, that is, our codec encodes the *difference* between the predictor's predicted value and the true value of an encoded symbol. If the predictor is good, then the residual can be encoded more efficiently than the full symbol value. Then, the decoder can use the predictor and the encoded residual to reconstruct the encoded symbol. We describe our techniques in detail in the following section.

Our predictive model alone was able to appreciably improve the average SNR of a dataset of test files, after being trained on an independent training set. Our Huffman entropy coding further improved our compression ratio by a non-negligible margin. We ran ITU-R BS.1116-3 tests with several participants, who rated our codec favorably over the Music 422 baseline codec. We report our results in Section 4.

2 Overview of Techniques

We integrate both predictive coding and Huffman entropy coding in the interest of a more efficient and effective codec. We discuss each in turn below.

2.1 Huffman Entropy Coding

Entropy coding[5] is a class of lossless compression techniques that encode symbols using a number of bits related to the probability with which those symbols occur. If the most common symbols are encoded with smaller numbers of bits, and the rarest symbols are encoded with higher bit amounts, the total number of bits for the stream of data, on average, will be lower than what would be used to encode the raw, uncompressed values.

Huffman coding is a specific entropy coding technique in which a binary tree is constructed based on symbol probabilities, and shorter codes are assigned to more frequent symbols. The result is a table of Huffman codes (symbol to code mappings) which can be easily referenced to encode or decode a stream of data. The generated codes of a Huffman table

are prefix-free, meaning that no code is the beginning of another code. The prefix-free property makes it easy to decode a Huffman stream: a decoder can read from a stream one bit at a time and, without ambiguity, separate the stream into valid code symbols.

In some non-audio applications, it may make sense to generate Huffman tables based entirely on the data being actively encoded and decoded. Such customized tables would be perfectly tuned to their respective signals, and yield the greatest coding gain. However, this approach is not ideal for audio. A generated table tailored to each unique audio file would be a considerable source of overhead, and the added time to generate that table would reduce the speed (and real-time streaming capabilities) of the coder. We took inspiration from the MPEG standard, which ships with a hard-coded set of precomputed tables, derived from symbol probabilities found in a wide set of training data. Information about our generated table set can be found in Section 3.

In this work, we apply Huffman coding to the integer-quantized mantissa values per MDCT block/frequency band; that is, our symbols are distinct mantissas.

2.2 Predictive Coding

Residual predictive coding is a well-appreciated technique in extant audio codec implementations. The basic idea, at its core, is simple: both the encoder and decoder are furnished with a *prediction algorithm* P, which, given a (compressed/truncated representation of a) sequence $x_0, \ldots, x_n$ attempts to predict x_{n+1} as $\hat{x}_{n+1} = P(n+1)$. If the predictor is any good, then $\hat{x}_{n+1}$ will be "close", via some metric of interest, to x_{n+1}. Thus, the encoder may encode the *residual*, $\delta_{n+1} = x_{n+1} - \hat{x}_{n+1}$, more efficiently than the original x_{n+1}. During decoding, the decoder may use its own copy of P to compute $\hat{x}_{n+1}$ and reconstruct x_{n+1} as $x_{n+1} = \hat{x}_{n+1} + \delta_{n+1}$. This reduces the quantization noise under floating point quantization relative to the amplitude, as the prediction accounts for some amplitude which the floating point scale factor doesn't have to account for; in other words, the effective range of the quantizer is reduced.

The proverbial devil, of course, is in the details. The choice of which sequences are predicted and how the predictor is implemented determines the efficacy of the idea in practice. In MPEG-2 AAC [3] Main, prediction is applied to long-window frames on a per-spectral-line basis using a second-order predictor that estimates each MDCT coefficient from reconstructed coefficients in previous frames. MPEG-4 AAC later introduced AAC-LTP, a separate long-term prediction algorithm that exploits temporal periodicity. Opus[6] uses a different hybrid design: its SILK layer uses linear prediction with both long-term and short-term predictors, while its CELT layer uses MDCT coding with prediction for band energies across time and frequency and a pitch predictor for spectral detail.

In our work, we experiment with short-term prediction applied directly to the values of the MDCT lines of our codec. We deviate from AAC in that we implement our MDCT line predictor as a small neural network, which predicts the next MDCT frame as a single predicted vector, rather than predicting individual MDCT lines in parallel, based only on the previous values of that line. Our focal objective throughout this work is to determine whether lightweight neural architectures can successfully learn and predict localized patterns with MDCT line values.

3 Implementation

We implemented our codec on top of our baseline perceptual audio codec implementation from previous coursework. The codec is an MDCT-based frequency domain audio codec that uses a FFT-based psychoacoustic model to minimize perceptual artifacts. The psychoacoustic model implements threshold in quiet and tonal/noise maskers to determine the masking thresholds used by bit allocation. We use 512 MDCT lines.

Block floating-point quantization is implemented such that each critical band shares a scale factor. After taking the MDCT, the prediction model is used and the residuals are then quantized and transmitted to the decoder.

We employ prediction conditionally: since our predictor is imperfect, the encoder only encodes residuals for bands where residual encoding is beneficial; if not, the original MDCT values are quantized and encoded. Finally, after prediction and residual quantization, Huffman coding is applied to the residual MDCT mantissas, using a custom Huffman table trained on residuals produced by the previous stages of the codec.

We chose a relatively small block size of 1024 samples (512 MDCT lines) for the following reasons: We found that our predictive model performed worse on higher MDCT sizes. Additionally, since we did not implement block switching, the lower block size helps a little to address transients, though this is still a large issue for our implementation. On the other hand, our predictor and Huffman coding help to offset the frequency resolution issues caused.

3.1 Implementation of the Neural Predictor

We employ a simple multilayer perceptron (MLP) for our predictor. The network consists of three fully-connected layers (input $\rightarrow$ hidden $\rightarrow$ hidden $\rightarrow$ output) with batch normalization and dropout ($p = 0.1$) after each hidden layer. Input sizes of 2-5 were used with a hidden dimension of 1024.

For training, we extract MDCT spectral frames from each WAV file using a KBD window with 50 percent overlap, and normalize all coefficients by the global standard deviation across the training corpus. The MLP predictor receives the concatenation of the previous N normalized MDCT frames as input and outputs a prediction of the next frame. MSE (L2) prediction loss is used; as such, the model focuses primarily on the highest-energy bins, which often correspond to the most prominent spectral components.

For the experiments documented in this report, we train and test on the SQAM files provided in Canvas. This small dataset lets us iterate quickly, and sanity-test our concept on a limited range of samples before scaling up. We split the SQAM dataset into training and validation examples at the file level, to ensure that the content of our training data does not contaminate our validation and testing results.

3.1.1 Bit Allocation For Residuals

Since we are quantizing a mixture of full MDCT lines and residual lines (depending on whether prediction is used for a given critical band), we should carefully consider how bits are allocated. Typically, bits are allocated such that noise-to-mask ratio (NMR) is minimized, meaning that more bits are given to critical bands with strong components that are above

the threshold in quiet and not masked by other stronger components. NMR is a function of the signal to mask ratios (SMR) and the signal to noise ratios (SNR). In practice, SMRs are computed first, then bits are allocated (setting the SNRs) selectively to control the resulting NMRs. However, given that we are quantizing prediction residuals instead of raw MDCT values, SNR calculation should change since it depends on the accuracy of the prediction. In practice, we found that a prediction-aware bit allocator that actively redistributes bits away from the smaller predicted bands was less effective than the bit allocation procedure of the baseline coder. As such, our bit allocation procedure is unchanged from baseline, and we allocate bits to a band regardless of whether prediction is used for that band. An effective prediction-aware bit allocation algorithm would be an interesting thread of investigation for future research.

3.1.2 Conditional Predictions

There is no such thing as a predictor that is both perfect and compact; it is, by definition, a space-efficient approximation of some pattern(s) in the training data. As such, there are conditions under which disregarding the predictions for a certain set of MDCT lines, and encoding the full MDCT values as implemented in the baseline codec, is preferable. We found that the *conditional encoding* of predictions significantly improves the performance of our codec.

We implement this at the critical band level. For each critical band, the encoder simulates bit allocation and decoding of both the MDCT line residuals and the full MDCT line values of that band. It then compares the reconstruction error of both approaches, effectively simulating what the decoder *would* see in each case. Naturally, it chooses the path that produces the smaller error. This choice is communicated to the decoder via a bit flag corresponding to each critical band. This induces a small (25 bit) bookkeeping overhead in each frame; however, the deleterious effect of bad predictions on SNR makes this overhead well worth the cost.

3.1.3 Snapping Small Predictions to Zero

We noticed that our predictor focused on the high energy main components of the signal, to the detriment of prediction quality for low energy areas. Furthermore, audio spectra often include near-zero values — this is one reason why frequency representations are so profitable in codec design. As such, if the previous values of an MDCT line all are under a given threshold (e.g. 1×10^{-5}), we snap the predictions to zero. This overrides any low energy noise that the MLP might learn as a side effect of focusing on the overall MSE.

3.2 Implementation of Huffman Coding

In the first step of training our precomputed Huffman table, we applied the baseline coder (without the neural predictor) to 78 WAV files from the EBU SQAM collection, provided as Music 422 class resources. We used a Python script to run our coder (pre-Huffman step) on each file, producing the integer-quantized mantissa values for each MDCT block and keeping track of how many times each mantissa value appears. We recognize that the distribution of mantissa values (which values occur the most) can change based on the bit allocation value for that band. As such, we maintained mantissa counts for each possible bit allocation value (2-16 bits). Then, we applied the basic process of constructing a binary Huffman tree to

create prefix-free codes [5], creating a new tree for each bit allocation value's set of mantissa counts. We derived 15 different tables, one for each bit allocation value between 2 and 16. All 15 tables are stored in a single JSON file with objects that map mantissa values to their chosen code (per bit allocation value). At the bitpacking/final file construction step, the coder indexes into the sub-table corresponding to the bit allocation value of the current encoded MDCT block and, for each mantissa symbol, packs the bits of the corresponding Huffman code into the output file.

3.2.1 Escape Codes

It is impractical to include all possible mantissa values in a Huffman table due to the ensuing explosion in average size per Huffman code, so a common approach (also used in MPEG) is to include an "escape code." The rarest mantissa values are not included in the table directly, but instead are accounted for by this escape code. When packing the file, if a mantissa is encountered that does not contain a Huffman code in the precomputed table, the bits of the escape code are packed into the file instead, followed by the raw bits for the integer mantissa value. Since these values are rare, the high number of bits required to encode a raw integer value does not have a devastating impact on the average compression rate. The escape code, functionally, is a signal to the Huffman decoder that, when encountered, the following bits (after the escape code) should be read and directly decoded as a raw integer. Intuitively, we can use the current bit allocation value to know how many bits after the escape code that must be read to recover the raw value.

3.2.2 Connecting to the Predictor

After establishing the infrastructure for the Huffman coding step, we adjusted it to the neural coded version of our codec implementation. Instead of training on the original mantissa values to generate the precomputed table, we first applied neural prediction and then trained a table based on the residual values (described previously). This results in a different set of Huffman codes, but the remainder of the compression/bitpacking steps are left unchanged.

4 Results

Our primary objective in this work is to determine whether a small neural network can act as an effective MDCT line predictor. We also implement and test Huffman coding, both on its own and in conjunction with our predictor. We thus report our results as a series of iterative steps:

1. We first evaluate the compression and coding gains resulting from the Huffman compression. We examine compression ratios and effective bitrate values (adjusted bitrates such that final file size is comparable to baseline encoding; see section 4.1) for versions of the model that include Huffman coding with and without the preceding prediction step. We examine a variety of critical material at effective bitrates of 128kbps and 96kbps.
2. We then evaluate the efficacy of the predictive model in isolation, outside of the codec pipeline. To do this with respect to a particular example signal, we compute (in decibels) the ratio between the input signal total energy and residual energy. Formally,

we define this *prediction gain* as

$$G_p = 10 \cdot \log_{10} \left(\frac{\sum_{i=1}^{N} \sum_{k=1}^{K} x[i,k]^2}{\sum_{i=1}^{N} \sum_{k=1}^{K} \delta[i,k]^2} \right) \text{ dB}, \tag{1}$$

where $x[i,k]$ is the MDCT coefficient at frame i, line k, and $\delta[i,k] = x[i,k] - \hat{x}[i,k]$ is the prediction residual computed by the model. This metric should indicate whether quantization noise would be reduced, in theory, when transmitting residuals.

3. The foregoing metrics help us determine whether the model is theoretically helpful; however, it says little about how our model fares when integrated into our codec machinery. As such, we also compare our codec to the baseline codec directly, by comparing the SNRs of their respective encodings. This metric determines whether our predictive Huffman codec produces better reconstructions than the baseline codec. While this isn't a perceptual metric, good results here strongly suggest good results in a perceptual test, given the existing perceptual bit allocation used by our codec.

   Note that this metric disregards the improved compression ratio from Huffman coding, as Huffman coding happens "outside" the codec, once bits have been allocated and mantissas calculated.

4. Finally, we run perceptual tests. The tests in this report center on five examples from the SQAM dataset. We recruited several participants to engage in an ITU-R BS.1116-3 listening test with all five examples. Participants used headphones in a quiet room at CCRMA with doors closed. Each example was tested at an *effective*[1] bitrate of 128 kbps and 92 kbps, and for each bitrate, we tested the baseline codec, our codec with prediction only, our codec with Huffman coding only, and our codec with both prediction and Huffman coding. As such, participants rated a total of eight test variants for each of the five examples.

   The predictive model used in our perceptual tests was trained on the EBU SQAM dataset. The five examples used in the tests were explicitly used as validation data; they were not used in training, and accurately reflect the model's ability to predict unfamiliar sequences.

Data from (2) and (3), evaluated with respect to a set of validation examples, is shown in Table 3.

As an additional benchmark, we also implement a second-order per-line linear predictor, similar to the predictor used in the AAC codec. For each MDCT line, the predictor independently maintains two coefficients that are updated frame-by-frame via normalized LMS using the reconstructed MDCT coefficients. The linear predictor is not the focus of this work; rather, it acts as a simple and comprehensible baseline against which our (relatively complex) neural predictor may be compared.

4.1 Bitrate Adjustment for Huffman Coding

There is a small complication with perceptual testing Huffman coding at a specific bitrate. Since Huffman coding is applied *after* the bit allocator has already distributed bits and

[1] Note that effective bitrate normalizes input bitrate with respect to the improved compression of Huffman coding, as described in Section 4.1

computed mantissas and scale factors, the resulting compression ratio is signal-dependent and cannot be determined a priori. However, our codec pipeline is always invoked with respect to a specific target bitrate. Therefore, the effective bitrate (measured by the `.pac` file size divided by signal duration) of a Huffman-encoded file will generally be *lower* than the target bitrate seen by the bit allocator.

To fairly compare codec configurations that include Huffman coding, we find an *adjusted bitrate* $b' > b$ for each audio example X such that the effective bitrate of X encoded with Huffman at target bitrate b' equals the effective bitrate of X encoded without Huffman at b. We compute b' via iterative multiplicative search. This adjustment ensures all codec configurations are compared at the same effective bitrate, so that differences in SNR reflect only the coding strategy and not a bitrate mismatch.

4.2 Huffman Evaluation

Table 1 shows isolated Huffman coding gains with tables generated and referenced for raw mantissa values (Huffman coding without prediction step). On average, raw mantissa Huffman coding produces a compression ratio of ∼1.3:1 compared to the baseline coder. This ratio is not significantly affected by effective bitrate.

In Table 2, we display the performance of Huffman coding for residual mantissa values (full coder implementation). The compression ratios are slightly lower than seen in Table 1: close to ∼1.23:1. We believe this may be due to the conditional prediction that we have in place, which results in the Huffman tables needing to represent two different distributions (the normal distribution and the residual distribution). The variance numbers we observed for the prediction residuals in isolation suggested that the residuals should in theory be more redundant than the raw mantissas, which we hoped we could exploit; however, we were unable to achieve this in practice.

It should be noted that the compression ratios/BPS reported for the castanets and SPGM samples are significantly lower than the rest of the critical samples, which reduces the average compression performance. It is difficult to understand why these samples report poorer ratios, but we propose that it may be due to the comparatively small presence of transient-heavy (e.g. castanets) and speech (e.g. SPGM) audio samples in the SQAM WAV dataset used to create our Huffman tables.

Signal	BPS (128)	Ratio (128)	BPS (96)	Ratio (96)
Castanets	2.7234	1.0211	2.0693	1.0347
Glockenspiel	4.2110	1.5792	3.1818	1.5909
Harpsichord	3.5442	1.3293	2.7486	1.3743
Oboe	3.9636	1.4864	2.9062	1.4531
SPGM	2.9238	1.0964	2.1622	1.0811
Average	3.4732	1.3025	2.6136	1.3068

Table 1: Huffman Compression Ratios and Effective BPS using the Huffman-only coding tables.

Signal	BPS (128)	Ratio (128)	BPS (96)	Ratio (96)
Castanets	2.7734	1.0395	2.0704	1.0352
Glockenspiel	3.7168	1.3936	2.7973	1.3987
Harpsichord	3.5608	1.3355	2.6817	1.3409
Oboe	3.3694	1.2632	2.5680	1.2840
SPGM	2.9160	1.0933	2.1498	1.0749
Average	3.2673	1.2250	2.4534	1.2267

Table 2: Huffman Compression Ratios and Effective BPS using the Huffman predictive residual coding tables.

4.3 Predictive Model Testing

Table 3 shows the efficacy of our predictive codec on five familiar validation examples. The validation gain column shows the predictive gain as predicted by the model in isolation, as defined in Equation (1). The baseline SNR corresponds to the SNR from the baseline Music 422 codec. Our SNR is the SNR derived from our neural predictive codec.

We observe positive prediction gain in four out of five examples. As these are validation files, these positive results indicate that the model was able to generalize its training data and predict MDCT lines to some degree of efficacy: the residual signal has less energy than the original signal, as desired. Our loss curve (Figure 1) shows convergence for both training and validation sets.

Similarly, the positive delta in the predictive codec SNR indicates that our codec was able to successfully translate the predictions from the MLP into concrete SNR improvements in the encoded files.

Note that the average quality of the predictive codec is not strictly correlated with the average quality of the underlying predictor (depending, of course, on which metrics are used to define quality). It is possible for the predictor to make "good" predictions most of the time, resulting in a favorable prediction gain; however, we observed that even a small number of particularly erroneous predictions is disproportionally harmful to the codec. The bit allocation/quantization algorithm suffers when its input signals have unusually high dynamic range. For this reason, a single bad prediction can degrade an entire critical band by pushing all the other predictions in that band to zero under quantization and scaling.

Unfortunately, unpredictably terrible outputs are part and parcel of machine learning research. LLM researchers might call these hallucinations; self-driving car researchers might call these very expensive highway accidents. Taming the "long tail" of a model's behavior distribution is not a universally solved problem, and is an active area of frontier research[2]. In our codec, we mitigate such phenomena via conditional prediction.

We observe that our predictions induce small, but non-negligible improvements in SNR over the baseline codec. The validation gain and the actual SNR improvement are not directly comparable, as they are different measurements—however, the data show a correlation between MLP prediction gain and real-world improvement when those predictions are integrated into a codec.

File	Val. Gain	Baseline SNR	Our SNR	Δ
castanets	−0.71	8.82	8.71	−0.10
glockenspiel	+3.89	18.20	19.89	+1.69
harpsichord	+2.74	9.95	10.83	+0.88
oboe	+6.76	17.78	21.37	+3.59
spgm	+2.86	17.35	18.75	+1.40

Table 3: MLP predictor performance across validation files. All figures are in dB.

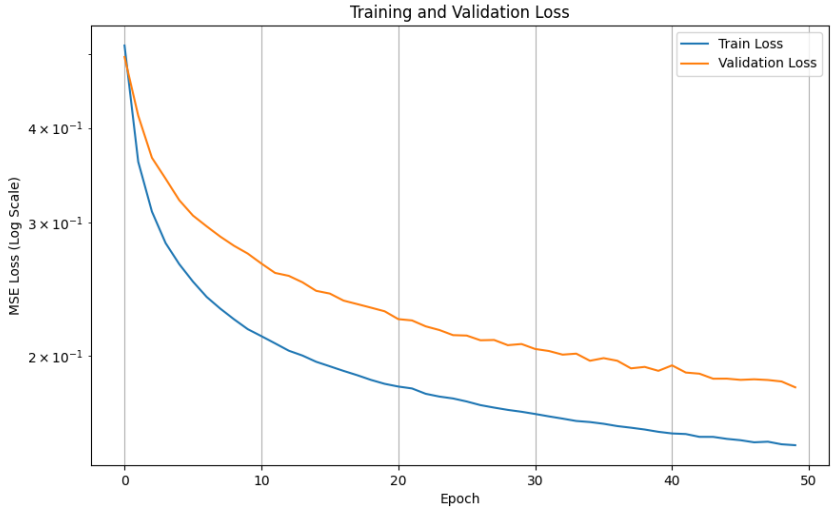

Figure 1: The loss curve of our MLP model, when trained on the EBU SQAM dataset from the Music 422 materials.

4.3.1 Analysis of Predictors on Oboe

In this section, we study the oboe sample in granular detail, to illuminate some interesting observations about our predictive codec.

During testing, the MLP outperformed the linear predictor by a small margin. Specifically, the MLP does a better job at predicting the most important high energy MDCT lines. As shown in Figure 2 and Figure 4, the MLP residual has reduced peak energy in the strong lower frequency components compared to the linear predictor. Figure 3 and Figure 5 also show this, as the highest energy MDCT lines are tracked better over time, especially the high peaks. For the oboe sample shown in these figures, the MLP reported a +6.76dB prediction gain (Equation (1)) compared to a +4.69dB prediction gain from the linear predictor.

4.3.2 Evaluation of Predictors on External Dataset

We also evaluated the MLP predictor and linear predictor on a large selection of files (200) from the Medley Solos dataset [4]. The results are seen in table (Table 4 and table Table 5). In this out-of-domain dataset the MLP scored +4.17dB prediction gain compared to the

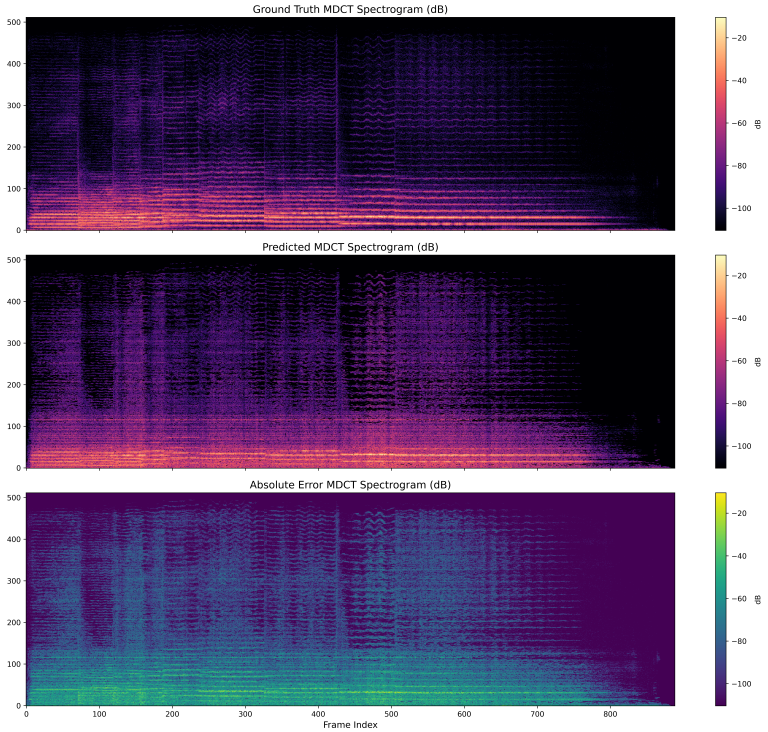

Figure 2: MLP Prediction Spectrograms (Oboe)

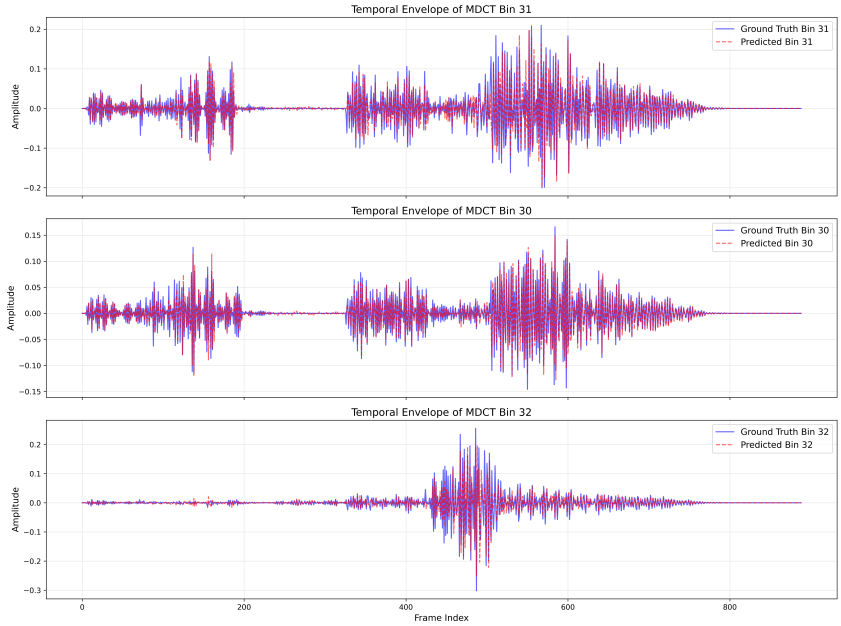

Figure 3: MLP Prediction Highest Energy Bins (Oboe)

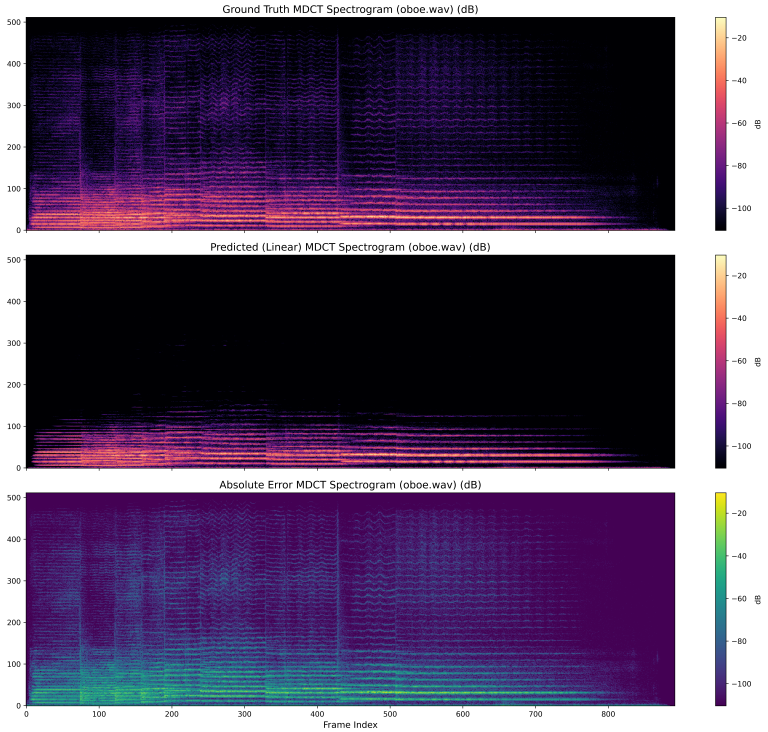

Figure 4: Linear Prediction Spectrograms (Oboe)

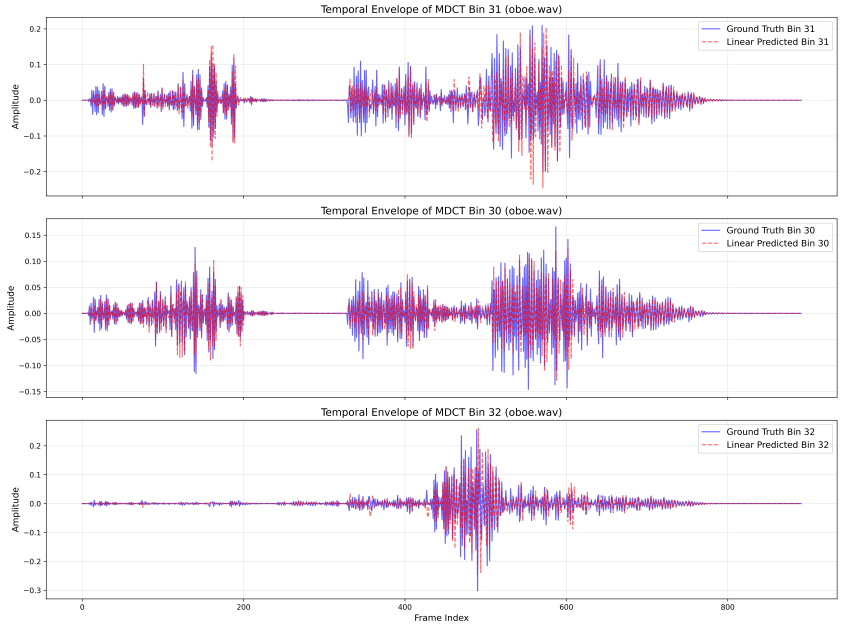

Figure 5: Linear Prediction Highest Energy Bins (Oboe)

linear predictor's +2.12dB prediction gain. Observe that the error distributions differ: in high energy bands the MLP far outperforms the linear predictor, but in low energy bands it suffers. Correspondingly, the MLP performs better with respect to maximum absolute error and MSE / variance, but not with respect to mean absolute error. For the purpose of perceptual audio coding, these metrics should be improvements over the linear predictor.

Metric	Ground Truth	Error	Ratio (%) / dB
Min Absolute	0.00×10^{0}	0.00×10^{0}	0.00%
Max Absolute	3.80×10^{-1}	3.11×10^{-1}	81.98%
Mean Absolute	1.95×10^{-4}	1.86×10^{-4}	95.49%
Mean Squared (MSE)	7.00×10^{-6}	2.68×10^{-6}	38.31%
Variance	7.00×10^{-6}	2.68×10^{-6}	38.31%
Prediction Gain	N/A	N/A	4.17 dB
Banded MSE (Threshold: -60 dB relative to peak)			
High-Energy Bins	Count: 1,069,872	MSE: 6.21×10^{-5}	Ratio: 36.64%
Low-Energy Bins	Count: 24,837,328	MSE: 1.23×10^{-7}	Ratio: 5831.87%

Table 4: Prediction Stats for MLP Predictor evaluated on Medley Solos DB.

Metric	Ground Truth	Error	Ratio (%) / dB
Min Absolute	0.00×10^{0}	0.00×10^{0}	0.00%
Max Absolute	3.80×10^{-1}	6.93×10^{-1}	182.53%
Mean Absolute	1.95×10^{-4}	1.38×10^{-4}	70.73%
Mean Squared (MSE)	6.98×10^{-6}	4.28×10^{-6}	61.38%
Variance	6.98×10^{-6}	4.28×10^{-6}	61.38%
Prediction Gain	N/A	N/A	2.12 dB
Banded MSE (Threshold: -60 dB relative to peak)			
High-Energy Bins	Count: 1,086,312	MSE: 1.03×10^{-4}	Ratio: 61.24%
Low-Energy Bins	Count: 25,128,088	MSE: 1.12×10^{-8}	Ratio: 526.89%

Table 5: Prediction Stats for Linear Predictor evaluated on Medley Solos DB.

4.4 Listening Tests

In Figure 6 and Figure 7, we show the ITU-R BS.1116-3 listening evaluation results on seven of our classmates, comparing the SDG for the baseline codec (no Huffman, no predictor), the prediction only codec, the Huffman only codec, and the full codec (Huffman and predictor). Generally, the predictor showed some improved SDG (over the baseline) and the Huffman coding provided an even more appreciable improvement. We observe some further improved SDG with our full codec (Huffman plus prediction) compared to either technique in isolation. The predictor was more beneficial for some sources, especially very tonal sources such as the oboe and glockenspiel. Huffman coding was effective for most of our chosen examples, aside from the castanets. Our codec did not improve SDG for the transient-heavy castanets sample, since our codec does not implement techniques (such as block switching) that address transient processing.

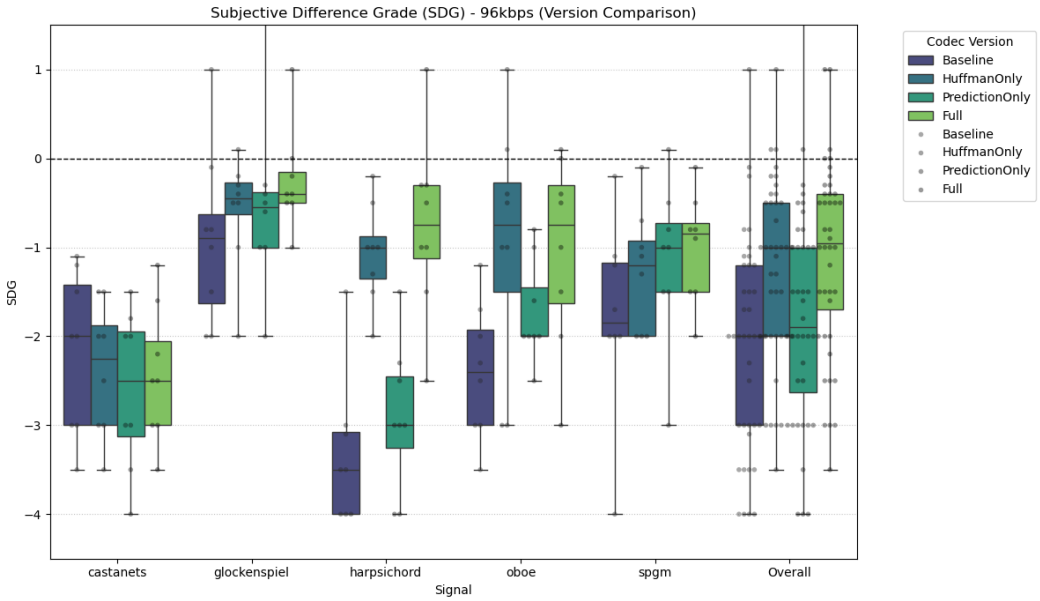

Figure 6: Listening Test Results (96kbps)

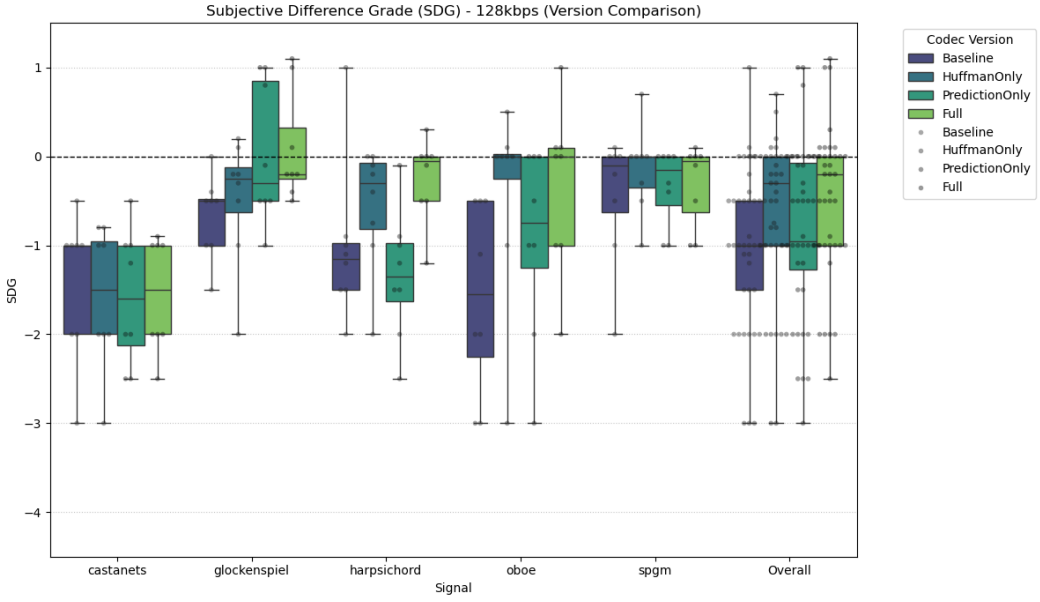

Figure 7: Listening Test Results (128kbps)

5 Conclusion

In this work, we explored two complementary techniques for improving the efficiency of an MDCT-based perceptual audio codec: Huffman entropy coding applied to quantized mantissa values, and residual predictive coding using a lightweight multilayer perceptron.

Huffman coding proved to be a reliable and consistent source of compression gain. Listening tests corroborated these findings, with Huffman coding producing appreciable perceptual improvements at both 96 and 128 kbps.

The neural predictor yielded positive but more modest gains. On validation data, the MLP achieved positive prediction gain on four of five test signals, and these gains translated into small but consistent SNR improvements over the baseline codec. The predictor was most effective on tonal signals such as the oboe (+3.59 dB SNR improvement) where temporal regularity in the MDCT coefficients is most exploitable. On transient-heavy material such as castanets, the predictor offered comparatively little benefit. We observed that conditional prediction, in which residual coding is only used on bands that would benefit from residual coding, is essential for our codec and worth the bookkeeping overhead, as bad predictions can annihilate any coding gains made from good predictions. However, without a more nuanced strategy, we believe this single-table approach may have limited the combined effectiveness of Huffman coding applied in conjunction with the residual coding, as a single Huffman table was built to represent a split set of residuals and raw MDCT values.

Overall, our results demonstrate that even a simple neural architecture can learn meaningful temporal patterns in MDCT spectra and that the resulting predictions, when carefully integrated with conditional encoding and entropy coding, yield measurable improvements in both objective metrics and perceptual quality over the baseline codec. While the gains from prediction alone are modest, we hope that this approach may illuminate promising directions for future research, with more expressive models and larger training corpora.

6 Future Work

This particular foray into predictive coding and entropy coding only goes so far, given our limited time window for conception, development, testing, and documentation. However, we believe that these preliminary results reveal promising horizons for future research.

Larger models and larger datasets. Our model is small, simple, and only ingests a small number of prior samples to make its predictions. Similarly, our dataset is relatively small — we train and test our codec on the EBU SQAM data uploaded to the Music 422 course materials. We do this for two reasons: first, we wanted to sanity-check the feasibility of neural predictions with a small model, before scaling, and second, our compute resources and development time were limited for this project. It is feasible that a larger model with a larger input context window could learn deeper patterns in the data, and thus produce more accurate predictions across a diverse spectrum of audio signals.

Specialized models for specialized signals. The universe of all audio signals that one might wish to encode is immeasurably vast and diverse. It is difficult to imagine that a single model might effectively capture such a complex probability distribution. A one-size-fits-all model may be outperformed, for certain sets of signals, by a specialized prediction model

specifically trained and tuned for signals in that set. For example, a different model could be used for speech versus music[2], or we could even develop different models for different musical genres.

Specialized Huffman tables. Similarly, it is feasible that Huffman tables derived from specific classes of audio (e.g. transient-heavy or speech, as mentioned in Section 4.2) may outperform "general" tables on those particular classes.

Block switching. This work does not address variable block switching. Pre-echo was one of the more notable deficiencies of the baseline codec during testing in previous assignments, and such artifacts are not ameliorated by Huffman coding nor predictive coding. Were we to further develop this codec, we would need to implement block switching and verify that the technique meshes well with our predictive models and entropy coding.

Performance. Computational efficiency remains a perennial concern for machine learning researchers. ML models are slow. Our MLP is likely a couple orders of magnitude less efficient than a simple linear predictor; furthermore, our codec implementation is an academic proof-of-concept and would require a substantial rewrite to be used in anything approaching production-grade software. Tuning our model to be as small and fast as possible, while preserving predictive efficiency, would be an interesting area of further research.

7 Author Contributions

Niccolo and Gabriel built the predictive codec pipeline, and Chris handled the Huffman coding integration. All three authors contributed to the paper in roughly equal measure.

References

[1] Marina Bosi and Richard E. Goldberg. *Introduction to Digital Audio Coding and Standards*. The Springer International Series in Engineering and Computer Science 721. Boston, MA: Springer, 2003. 434 pp. ISBN: 978-1-4615-0327-9. DOI: 10.1007/978-1-4615-0327-9.

[2] Yi Dong et al. *Building Guardrails for Large Language Models*. 2024. arXiv: 2402.01822 [cs.CL]. URL: https://arxiv.org/abs/2402.01822.

[3] ISO/IEC JTC1/SC29. *ISO/IEC 13818-7:2006 Advanced Audio Coding (AAC)*. International Standard. 2006. URL: https://www.iso.org/standard/43345.html.

[4] Vincent Lostanlen et al. *Medley-Solos-DB: A Cross-Collection Dataset for Musical Instrument Recognition*. Version 1.2. International Society for Music Information Retrieval (ISMIR). New York, NY, USA: Zenodo, 2019. DOI: 10.5281/zenodo.3464194. URL: https://doi.org/10.5281/zenodo.3464194.

[5] Alistair Moffat. "Huffman Coding". In: *ACM Comput. Surv.* 52.4 (Aug. 2019). ISSN: 0360-0300. DOI: 10.1145/3342555. URL: https://doi.org/10.1145/3342555.

[2] This is not a novel idea; some extant codecs adapt to the type of audio that they are encoding. Opus[6], for example, processes music and speech differently.

[6] Jean-Marc Valin, Koen Vos, and Timothy B. Terriberry. *Definition of the Opus Audio Codec*. Request for Comments RFC 6716. Num Pages: 326. Internet Engineering Task Force, Sept. 2012. DOI: 10.17487/RFC6716. URL: https://datatracker.ietf.org/doc/rfc6716 (visited on 03/07/2026).