TRANSFORMERS FROM COMPRESSED REPRESENTATIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Compressed file formats are the cornerstone of efficient data storage and transmission, yet their potential for representation learning remains largely underexplored. We introduce **TEMPEST** (TransformErs froM comPressed rEpreSenTations), a method that exploits the structured byte-stream of compressed files to design an effective tokenization and encoding strategy. By leveraging this compact encoding, a vanilla transformer can directly learn semantic representations from compressed data streams. Our proposal reduces the number of tokens required for semantic classification, thereby lowering both computational complexity and memory usage. Moreover, we discover unique properties of compressed byte stream for representations learning, specifically, the existence of complementary information across compression levels. Through extensive experiments across diverse datasets, coding schemes, and modalities, we show that TEMPEST achieves accuracy competitive with raw data baselines while delivering efficiency gains in memory and compute.

1 INTRODUCTION

Transformer architectures were originally developed for language processing (Vaswani et al., 2017) and later extended to various multimedia domains (*e.g.*, images, audio, video (Dosovitskiy et al., 2020; Gong et al., 2021; Arnab et al., 2021)). A major challenge in applying transformers to such modalities is the length of the tokenized sequence: a few seconds of audio or video are mapped to lengthy sequences (Arnab et al., 2021; Gong et al., 2021; Tay et al., 2020). Such large sequences pose a key limitation for the transformer architecture, as the memory and computational requirements of the attention mechanism scale quadratically with the number of tokens. Prior work has attempted to mitigate this issue through approximate attention Choromanski et al. (2021); Xiong et al. (2021), reduced-complexity operations (Wang et al., 2020; Child et al., 2019), or token merging strategies (Haurum et al., 2023; Bolya et al., 2022). In this paper, we address the complexity issue of attention-based architectures from an orthogonal direction: we use the structure of Compressed File Formats (CFFs) to re-map the input data into a shorter, more compact sequence that preserves the relevant semantic information.

CFFs use perceptual modeling and quantization (of Congress, 2024; Pennebaker & Mitchell, 1992) to remove redundant and irrelevant content in the data-stream, resulting in efficient storage and transmission formats. This property has attracted research interest (Pérez et al., 2024; Horton et al., 2023b) as an alternative to shorten the input sequence in transformers. However, current methods remain limited as CFFs are a continuous byte stream, and bytes are poor tokenization units. In a CFF a single byte cannot be individually decoded, as it requires the evaluation of multiple additional symbols in the compressed stream to resolve its meaning. As a consequence, byte-level attention typically consumes the capacity of the transformer by learning a static encoding scheme that the CFF already declares.

Most CFFs are structured into *blocks*, defined as the smallest byte stream that can be decoded independently. Such blocks define a sub-structure that is self-contained by construction, bounded by format-specific markers, and consistent for the entire stream, *i.e.* every block in a file is encoded under the exact same scheme. While these sub-components are primarily designed to enable robust error handling and on-the-fly decoding, we demonstrate that they also provide a natural tokenization scheme for CFFs.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

In this work, we advance representation learning from CFFs, and demonstrate that each sub-component can be independently located and embedded using a lightweight transformer network. Moreover, this sequence of tokenized sub-components is shorter when compared to the raw signal, and can serve as the input to a second transformer architecture specialized in learning semantic representations.

Our approach leverages the compactness and structured nature of compressed data in CFFs and their sub-components, and enables direct semantic modeling across diverse compressed data formats while substantially reducing the number of tokens per sub-component. As a consequence, we can lower the computational complexity and memory requirements in the final model. Finally, operating in the compressed domain also exposes novel properties with no direct counterpart in the raw signal. Sampling bit rates during training acts as an effective data augmentation technique, and pooling predictions across multiple bit rates becomes an effective test-time augmentation.

We name our method TEMPEST (TransformErs froM comPressed rEpreSenTations). We empirically demonstrate that TEMPEST achieves results competitive with the uncompressed data baselines across various datasets, low-entropy coding schemes, while also offering empirical memory savings. Our contributions are three-fold: i) We introduce the a practical tokenizer for compressed data, our proposal can work across multiple CFF and multiple modalities. ii) We show that a standard transformer encoder architecture can learn semantic representations from the compressed data tokenizer, resulting in competitive results with audio classification methods and improving over the current baselines for compressed data. iii) For some compressed formats, after tokenization, we have a small token sequence that results in empirical memory savings.

2 Related Work

Byte Sequence Modeling for Uncompressed Data. Recent advances in byte sequence modeling have introduced techniques to handle the challenges of raw data streams. The MegaByte framework Yu et al. (2024), for example, employs a multi-scale decoder transformer architecture and introduces a "byte patchification" strategy to manage the length and complexity of byte sequences. Similarly, bGPT Wu et al. (2024) adopts a decoder-only Transformer for autoregressive byte sequence generation. While these approaches advance the modelling of byte-level data, they remain limited to uncompressed inputs and overlook the more prevalent and practical case of compressed file formats. In contrast, our work directly addresses this format gap by operating on the compressed byte stream, using MP3 encoding as a primary case study.

Training over partially decoded JPEGs. Rather than relying on raw RGB pixel values, some related studies have proposed Convolutional Neural Network (CNN) Gueguen et al. (2018); Verma et al. (2018) and Vision Transformer (ViT) Park & Johnson (2023) architectures that operate directly on partially decoded JPEG representations. For example, (Gueguen et al., 2018) introduced a CNN (LeCun et al., 1995) capable of processing such inputs, enabling faster image handling by bypassing parts of the decoding pipeline. In a similar spirit, (Park & Johnson, 2023) demonstrated that a ViT Dosovitskiy et al. (2020) can effectively leverage partially decoded JPEGs, allowing the model to engage with data in a more compressed form. Together, these works show an initial shift towards exploiting the compactness and some structural characteristics of compressed formats, while still retaining some degree of decoding. In comparison our proposal fully relies on the sub-component structure of the CFF, moreover, no decompression (not even partial) is performed at any stage of our pipeline.

Direct Operations on Compressed JPEG Byte Sequences. ByteFormer Horton et al. (2023a) explored the processing of compressed formats at the byte level, demonstrating that JPEG byte sequences present substantial challenges due to their non-linear encoding and variable length. The study further showed that conventional byte patching strategies can hinder performance, given the high information density of compressed data. The work of (Pérez et al., 2024) is perhaps the closest approach to our method, however it is more focused on byte-level modeling and error handling. This method incurs a large computational penalty given that the length of the sequence is never reduced. In contrast, our approach formulates a novel byte tokenization strategy that significantly reduces the token length of CFFs sequences, moreover, our design operates on significantly less tokens per CFF component thus reducing the overall computational complexity of the method.

Table 1: **Organization of the block sequence across CFFs.** MP3 frames form a 1D sequence indexed by time, whereas JPEG MCUs form a 2D grid that is flattened in raster order before being fed to the classification network.

	MP3	**JPEG**
Block unit	Frame	Minimum Coded Unit (MCU)
Block Layout	1D array	2D array
Sequence order	Temporal, $k \geq 0$	Raster, $k = r\,C + c$
Number of blocks	$\lceil T/31\,\mathrm{ms} \rceil$	$R \cdot C = \lceil H/8 \rceil \lceil W/8 \rceil$
Block support	≈ 31 ms of audio	8×8 px region
Block length L	144 bytes	variable
Boundary marker	byte-aligned sync word	sub-byte, approximate

3 Transformers from Compressed Representations

Bytes constitute the default vocabulary of Compressed File Formats (CFFs), with a fixed alphabet of 256 possible values (*i.e.* `0x00`, ..., `0xFF`). However, a byte is not a self-contained unit, compression algorithms often operate at the bit-level packing information across byte boundaries. The bits of an individual symbol may simultaneously be packed across multiple bytes of the compressed stream, thus requiring the decoding of adjacent bytes to be fully understood.

To better approach representation learning in a CFF, we leverage the structural design of many compression schemes (e.g., MP3, Opus, JPEG), which are organized around *blocks*. A block represents the smallest encoded unit that can be decoded independently of the rest of the stream. By construction, blocks encapsulate self-contained information, making them a more suitable unit for tokenization than individual bytes. Each block is encoded independently, and all blocks follow the same compression scheme, this inter-block consistency enables the learning of regularities of the encoding scheme from individual blocks, and ultimately to estimate useful feature embeddings from a CFF. More importantly, these blocks reflect meaningful units of the original signal, an MP3 frame corresponds to a fixed-duration audio segment, while a JPEG MCU encodes a local spatial region of an image. Table 1 shows a brief overview of the role of blocks in each format.

Building on this structural regularity, we treat blocks as the basic unit for tokenization in a CFF. TEMPEST (TransformErs froM comPressed rEpreSenTations) combines a novel token embedding strategy designed for compressed data representations with a vanilla transformer encoder (Vaswani et al., 2017). TEMPEST aggregates the features across the sequences of the embedded blocks by jointly optimizing the tokenization and aggregation steps, learning an embedding that captures the most relevant semantic content within the CFF, and allowing downstream tasks such as classification. TEMPEST components and their overall workflow are illustrated in Figure 1.

3.1 Embedding Compressed Byte-streams

In TEMPEST we seek the start and end boundary markers in the selected CFF and map each individual sub-component to one or a few tokens. For instance, the MP3 iso (1993) standard organizes data into frames, and the JPEG itu (1992) format defines Minimum Coded Units (MCUs) as local blocks. TEMPEST will initially search for individual Frames or MCUs, according to the format, and then start the embedding process.

Learnable Block Embedding. Let $S = \{b_0, b_1, \ldots, b_{n-1}\}$ denote a compressed byte stream of length n, where each b_i represents a single byte in $\{0, \ldots, 255\}$. We read b_i as the integer representation of the underlying 8-bit value (i.e., $0 \equiv$ `0x00`, $255 \equiv$ `0xFF`). Using the format specific markers we partition S into $m \ll n$ blocks:

$$B = \{B_0, B_1, \ldots, B_{m-1}\}, \quad B_j = \{\, b_i \in S \;:\; u_j \leq i < v_j \,\}$$

Where B is the block sequence inside of the stream S, B_j denotes the j-th block inside the CFF, and u_j and v_j are the start and end position of block j. To simplify the operation in our architecture, we fix the length of B_j at l whose value is dependent on the CFF. If a block is shorter than l we pad with

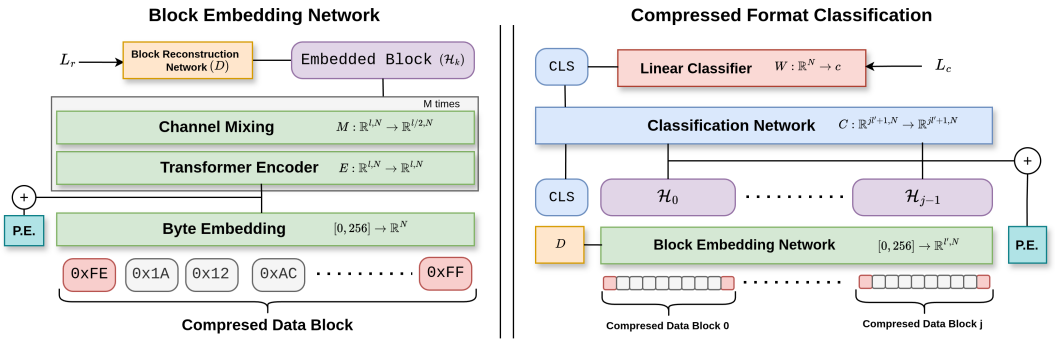


Figure 1: **TEMPEST architecture.** TEMPEST consists of three sub-networks: the block embedding network (green), the classification network (blue), and the block reconstruction network (orange). The input to TEMPEST is a compressed byte stream (gray and red), which is split into sub-components (compressed data blocks) according to the special byte markers defined in the CFF (the byte values shown in red, values are for illustration only). Each compressed block is mapped to an embedding (purple), whose representation is regularized by the reconstruction network (orange). The classification network is a ViT-like architecture: it prepends a [CLS] token and produces the final classification from the sequence of embedded blocks.

special tokens which are outside of the vocabulary of the compressed representation (e.g., integer value 256).

After parsing and padding the block sequence B, we embed each B_j independently (see Figure 1 left). First, each byte is mapped into an N-dimensional vector space, producing $B_j \in \mathbb{R}^{l \times N}$, then we apply a sinusoidal positional encoding P_E . Each projected sequence is then processed independently by a lightweight stack of transformer encoder layers E which performs intra-block attention. Each encoder layer is followed by a channel-mixing sub-network M, this layer is inspired by the MLP-Mixer Tolstikhin et al. (2021), it projects and halves the sequence length (l to $l/2$ after E_0, $l/2$ to $l/4$ after E_1) except in the final layer where the length of B_j is reduced to $l' \ll l$ regardless of the initial value of l. The result is a compact, high-dimensional representation of B_j, we name this sequence the *Block Embedded Sequence* ($\mathcal{H}$) and corresponds to the tokens in TEMPEST :

$$\mathcal{H}_j = M \circ E(B_j + P_E), \quad \mathcal{H}_j \in \mathbb{R}^{l' \times N}. \tag{1}$$

For simplicity, the notation in Equation 1 has a single instance of $M \circ E$, in practice, we stack multiple such layers for the tokenizer.

To ensure that the Block Embedded Sequence remains informative, we regularize it with a reconstruction objective, which makes the block embedding process similar to an autoencoder Hinton & Salakhutdinov (2006); Goodfellow et al. (2016). Specifically, a single-layer transformer decoder D is trained to regress the original block:

$$\hat{B}_j = D(\mathcal{H}_j),$$

with a 257-way classification (256 unique bytes and the padding byte) head and cross-entropy loss applied at each position, treating each byte as a categorical variable. Unlike a conventional autoencoder, our goal is not to achieve perfect reconstruction of the original signal, but rather to encourage $\mathcal{H}_j$ to serve as both (i) a semantically meaningful representation and (ii) a compact approximation of the original compressed block.

Learning From Block Embedded Sequences. Once the block Embedded Sequence $\mathcal{H} = \{\mathcal{H}_0, \mathcal{H}_1, \ldots, \mathcal{H}_{m-1}\}$ has been estimated, we work towards building a semantic representation from $\mathcal{H}$ (see Figure 1 right). Our approach follows the original design of a Vision Transformer (ViT) Dosovitskiy et al. (2020), where the elements of $\mathcal{H}$ are treated as tokens. Each token corresponds to a contiguous segment of the original data, already mapped into a compact embedding by the

block embedding network. We apply a sinusoidal positional encoding to $\mathcal{H}$, and prepend a learnable `[CLS]` token. The sequence is processed by a transformer encoder C, and the `[CLS]` output is mapped by a linear classifier W to the predicted class $\hat{c}$:

$$\hat{c} = W \circ C([\texttt{CLS}, \mathcal{H} + P_E])$$

Overall, TEMPEST consists of three sub-networks: (i) the *block embedding network* $M \circ E$, (ii) the *block reconstruction network* D, and (iii) the *classification network* $W \circ C$. These components are trained jointly with two objectives: a reconstruction loss L_r, and a classification loss L_c. The final training objective can be expressed as:

$$L = L_r + \lambda L_c = \mathcal{L}(\hat{b}_j, b_j) + \lambda \mathcal{L}(\hat{c}, c)$$

4 EXPERIMENTS

We evaluate TEMPEST across both audio and image domains to assess its effectiveness and generality. Specifically, we consider one audio format (MP3) and one image format (JPEG). For each format, we provide implementation details, training setup, and evaluation protocols, followed by a discussion of the results.

Compressed Audio. We evaluate TEMPEST on MP3 Streams. This compressed format is based on perceptual audio coding, where the block-level units are referred to as *frames*, the frame size in MP3 depends on both the bit rate and the sampling rate. In our setup, we fix the sampling rate at 44.1 kHz and vary the bit rate between $20 - 32$ kbps. Under these settings, frames are nearly uniform in length with an average of 144 bytes, corresponding to approximately 32 milliseconds of audio. Whenever a shorter frame occurs, we pad it with the special token value of 256.

An important property of compressed audio is that varying the bit rate implicitly serves as a form of data augmentation. For instance, MP3 streams generated at 20 kbps and 32 kbps differ in approximately 90% of their byte values for the same underlying audio signal. While the decoded waveforms are not identical, the differences are designed to be perceptually negligible to human listeners. We show that TEMPEST is robust to such variability and can even benefit from it, leveraging the diversity in the compressed representations to improve generalization.

Compressed Images. We follow a similar methodology for images. However the block units in JPEG *MCU* are far more complex as the default encoding does not require a byte marker between MCUs. Rather, it favors a scan-line (a set of consecutive MCUs which align left to right and cover an $8 \times n$ horizontal region image where n is the image width) of MCUs encoded together. Nevertheless, we use the encoder settings to generate byte level boundaries of MCUs over JPEG files encoded with a quality setting of 80. Under this settings, the layout of MCUs resembles the consecutive, non overlapping structure of MP3 frames.

We present the detailed implementation details, along with the set of optimal hyper parameters for each dataset in the **Supplementary Material**.

4.1 DATASETS

ESC-50. ESC-50 Piczak (2015) is a dataset of environmental audio recordings spanning 50 classes such as animal sounds, human activities, natural phenomena, and background noises. It contains a total of 2,000 clips, each 5 seconds long, recorded at 44.1 kHz. The dataset is evenly balanced across classes, with 40 examples per class. Following standard protocol, we report results using 5-fold cross-validation, ensuring that clips from the same fold are held out during training. This setup allows for robust comparison with prior work on ESC-50.

Speech Commands v2 (SC2). The Speech Commands v2 dataset Warden (2018) is designed for keyword recognition. It consists of 105,829 one-second audio clips of 35 spoken words, recorded from thousands of speakers at 16 kHz. The official dataset split contains 84,843 clips for training, 9,981 for validation, and 11,005 for testing. Each example is short and homogeneous in length, making the dataset well-suited for small-footprint audio classification models.

Table 2: **Results on audio datasets.** We evaluate TEMPEST against the baseline of AST (Gong et al., 2021) in the Speech Commands V2, ESC50 and AudioSet (mAP for the balanced and imbalanced training sets) datasets. We show improved or competitive results when learning from scratch. In addition, we report the number of *Tokens Per Second* in the model (TPS), TEMPEST shows a significant reduction in the sequence length per second of audio. We also report the computational cost in GFLOPs per second of audio. The *Compressed Data* column designate methods that operate directly on the compressed byte stream.

	Compressed	AudioSet ↑					
Method	Data	Bal.	Imb.	ESC-50 ↑	SC2 ↑	TPS ↓	GFLOPs ↓
AST	×	14.80	**36.60**	41.90	**92.60**	108	19.2
TEMPEST	✓	**15.24**	33.11	**52.28**	91.24	**32**	4.68
ByteFormer	✓	-	-	-	88.69/89.19	809/1619	**4.53**/5.49

AudioSet. AudioSet Gemmeke et al. (2017) is a large-scale dataset of 10-second YouTube clips labeled with a hierarchical ontology of 527 sound event classes. It contains approximately 2.1 million annotated clips with a highly imbalanced class distribution. The average clip duration is 10 seconds sampled at 16 kHz. In our experiments, we also present results for the balanced subset with 20,371 clips.

CIFAR-10 and CIFAR-100. The CIFAR-10 and CIFAR-100 datasets (Krizhevsky et al., 2009) each contain 60,000 RGB images at a resolution of 32×32 pixels. The datasets contain 10 and 100 semantic classes respectively. Both datasets are divided into 50,000 training and 10,000 test images, with a balanced distribution across classes.

Tiny ImageNet. Tiny ImageNet (Le & Yang, 2015) is a subset of ImageNet (Deng et al., 2009) containing 120,000 RGB images at a scale of 64×64 pixels and 200 classes. The dataset is divided into 100,000 training, 10,000 validation and 10,000 test images.

4.2 Comparison against Transformer Baseline

We begin evaluating TEMPEST classification accuracy in audio and image data against its most direct baselines, we consider baselines that use raw or compressed data. For the audio experiments we choose Audio-spectrogram transformer (AST) Gong et al. (2021) as the raw data baseline. Similar to AST, TEMPEST uses a standard transformer encoder and a `[CLS]` token for classification. Their only difference lies in the input data, while AST uses a patchified spectrogram as the transformer input, our method uses the compressed version of the same audio. Although there are not many equivalent works in the compressed domain for audio, we complement the comparison with the compressed baseline of Horton et al. (2023a). ByteFormer also trains and validates on compressed audio data, however, ByteFormer proposes a simpler tokenization instantiated as a strided 1D convolution.

There is currently no large scale pretraining available for our tokenization scheme. Therefore, we compare TEMPEST against models trained from scratch. We use the performance metrics reported in (Gong et al., 2022) and Horton et al. (2023a) for both methods. Table 2 summarizes our results.

We observe that TEMPEST can improve over AST in two benchmarks. In the data-limited dataset ESC-50, and also in the more challenging AudioSet balanced, remaining close in the SC2 and the imbalanced set for AudioSet. When compared to AST, TEMPEST has much shorter token sequence. We measure the length of the token sequence with the number of *Tokens Per Second* (TPS) TEMPEST only adds 32 tokens for each second of audio, meanwhile AST adds 108 tokens per second.

The comparison on TPS is far more favorable against ByteFormer, this model adds 1619 or 809 TPS depending on the size of the initial convolutional layer (k=4 or k=8). We note that in both cases the memory footprint of downstream computations ($\mathcal{C}$) is drastically improved due to our tokenization scheme, in particular, for a single second of audio the attention matrix in TEMPEST is about an order of magnitude smaller than in AST (11664 total elements in AST vs 1024 elements in our proposal). Byteformer long token sequence forces the model to performs windowed attention with

Table 3: **Results on RGB datasets.** We evaluate TEMPEST against a ViT baseline (Dosovitskiy et al., 2020) on MNIST, CIFAR-10, CIFAR-100 and TinyImageNet. All models are trained from scratch. The number in parentheses next to each accuracy is the number of *Tokens Per Image* (TPI) processed by the model on that dataset.

Method	Compressed Data	MNIST ↑	CIFAR-10 ↑	CIFAR-100 ↑	TinyImageNet ↑
ViT	×	98.88(16)	88.47(64)	68.2 (64)	57.18(64)
TEMPEST	✓	98.79(16)	83.13(64)	62.61(64)	51.84(64)
Perez *et al.*	✓	97.1(561)	56.9(921)	-	-

a window size of 32, each individual attention window inside Byteformer has the same size as full second of audio in TEMPEST .

Table 3 shows the comparison of TEMPEST in the image domain using JPEG image data, against a ViT baseline working in RGB data. We also include the closest compressed data method in the literature Pérez et al. (2024). Although TEMPEST cannot exceed the ViT baseline, it remains close, and the gap does not seem to increase according to the complexity or scale of the data set. More importantly TEMPEST clearly overcomes the compressed baseline of Pérez et al. (2024) in terms of accuracy, and with far less computational complexity. The numbers in parenthesis represent the average number of tokens per image in the dataset. TEMPEST has 1 or 2 orders of magnitude less tokens per image than the baseline of Pérez et al. (2024).

4.3 Ablations

We now conduct a series of ablation studies to evaluate the impact of key architectural design choices in TEMPEST. Specifically, we focus on three critical components: (i) l', the final length of the Block Embedded Sequence ($\mathcal{H}$), which directly controls the memory usage and is expected to influence downstream performance, (ii) the contribution of the reconstruction loss (L_r), which serves as a regularizer for the block embedding network, and (iii) the number of encoder layers used in the block embedding network, which controls the model representation capacity of the block embedding network ($M \circ E$).

Length of Block Embedded Sequence (l'). As described in Section 3.1, the block embedding module reduces the sequence length from l to l'. This design significantly reduces the memory usage of $\mathcal{C}$. We study the effect of varying l' in a range $1 \leq l' \leq 3$ and report the classification accuracy of the audio datasets. In addition to classification accuracy, we also report the *Token-to-Byte Ratio* (TBR), defined as the number of embedded tokens relative to the number of bytes in the original compressed block (l'/l).

Table 4: **Ablation on the Block Embedded Sequence Length (l').** Classification accuracy (mAP for Audioset) of TEMPEST as a function of the number of tokens per block (l') in the Block Embedded Sequence $\mathcal{H}$. We observe a slight loss in performance when $l > 1$, suggesting redundancy in the extended versions of $\mathcal{H}$.

l'	ESC-50 ↑	SC2 ↑	AudioSet Bal. ↑	TPS	TBR
1 Token	**52.28**	**91.24**	**15.24**	32	1/144
2 Token	51.30	91.18	14.92	64	1/72
3 Token	50.85	91.20	14.94	96	1/48

We observe in Table 4 that increasing l' from one to three tokens does not improve the accuracy and in fact results in a minimal performance penalty, with the best results obtained at $l' = 1$ on all datasets. This suggests that our tokenizer can effectively pack all the relevant information into a single token, making $l' > 1$ redundant. In addition, $l' > 1$ diminishes the gains in memory efficiency (see TPS). We fix $l' = 1$ for every experiment in this paper as it provides the most favorable accuracy–efficiency sequence length.

Table 5: **Ablation by depth of the Block Embedding Network.** We observe nearly identical performance by using encoder deeper than 2 layers. The network clearly diminishes its performance for a single layer block embedding. More encoder layers add a computational penalty, so we choose 2 layers.

$M \circ E$ Layers	ESC-50 ↑	SC2 ↑	AudioSet Bal. ↑	$M \circ E$ GFLOPs ↓
1	45.14	90.53	13.81	2.83
2	**52.28**	**91.24**	15.24	4.24
3	52.15	91.11	15.09	4.82
4	52.14	91.09	**15.37**	5.08

Depth of Block Embedding Network $M \circ E$. We analyze the effect of varying the number of encoder layers in the block embedding network $M \circ E$. Results are shown in Table 5. Across all datasets we observe that 2 layers is the best setting. Overall the performance for 2, 3 or 4 layers are nearly identical across datasets. For the audioset balanced increasing $M \circ E$ to 4 layers shows a slight increase in performance with an approximate penalty of 0.84 GFLOPS. Another consistent result across datasets is that one layer is not enough to learn meaningful representations from the compressed stream. This suggests a 2 layer tokenizer already allows the model to extract features from compressed data. Deeper embeddings incur higher computational cost, and are unnecessary, for this reason, we adopt 2 layers of $M \circ E$ as a practical compromise between accuracy and efficiency.

Reconstruction Loss L_r. We now investigate the contribution of the reconstruction loss L_r as a regularizer, Table 6 summarizes the results. As expected, the main driver of TEMPEST performance is the supervised classification loss L_c. However, adding L_r provides consistent improvements on all datasets, indicating that enforcing reconstruction of the compressed blocks helps the model learn more informative embeddings.

Table 6: **Ablation by loss objectives.** The main driver of TEMPEST performance is the Classification loss L_c. However the reconstruction loss provides an empirical improvement across all datasets.

Loss Setting	ESC-50	SC2	AudioSet Bal.
L_c	43.80	89.69	13.10
$L_c + L_r$	**52.28**	**91.24**	**15.24**

Table 6 shows that including L_r improves accuracy from 43.80 to 52.28 on ESC-50 (+8.48 absolute, +19.4% relative), from 89.69 to 91.24 on SC2 (+1.55 absolute, +1.7% relative) and from 13.10 to 15.24 on audioset (+2.14 absolute, +16.3% relative). These consistent gains validate that adding the reconstruction loss is beneficial.

Training with Multiple Bit Rates. As discussed in Section 3, compressed audio streams generated at different bit rates can differ substantially at the byte level, even when the underlying waveform is perceptually similar. To assess whether leveraging this variability is beneficial, we train TEMPEST using either a single bit rate (32 kbps) or a mixture of bit rates (20, 26, and 32 kbps). During multi-bit rate training, the bit rate is sampled uniformly at random for each training example.

Results are reported in Table 7. On SC2, we observe a small improvement from 90.15 to 91.24 (+1.09 absolute, +1.2% relative), while on AudioSet Bal. accuracy improves from 14.40 to 15.24 (+0.84 absolute, +5.8% relative). These results show that exposing the model to compressed representations at different bit rates acts as a form of data augmentation, improving accuracy and generalization.

Inference With Multiple Bit Rates. We observe that training with multiple bit rates serves as a form of data augmentation. Building on this property of CFFs, we further investigate whether

Table 7: **Training with multiple bit rates.** We empirically observe that including multiple bit rates at training time results in an improved performance. For CFFs the coding rate acts as data augmentation.

Training Bit Rates	SC2 ↑	AudioSet Bal. ↑	AudioSet Imb. ↑
32 kbps	90.15	14.40	32.47
20, 26, 32 kbps	**91.24**	**15.24**	**33.11**

Table 8: **Inference with multiple bit rates.** Accuracy on SC2, mAP on AudioSet Balanced and AudioSet Imbalanced increases when combining progressively more bit rates, suggesting complementary information across rates. A ✓ indicates that the encoding at that bit rate is included in the forward pass and its prediction is aggregated.

Inference Bit Rate (kbps)			SC2 ↑	AudioSet Bal. ↑	AudioSet Imb. ↑
20	26	32			
✓	✗	✗	90.64	15.22	32.91
✗	✓	✗	90.64	15.32	32.93
✗	✗	✓	91.24	15.24	33.11
✓	✓	✗	90.91	15.32	32.93
✗	✓	✓	**92.31**	**15.84**	**33.68**
✓	✓	✓	91.76	15.79	33.58

performing inference at multiple bit rates for the same input can improve accuracy. In this setting, an audio file is re-encoded at different bit rates, and the corresponding predictions are aggregated at inference time.

As shown in Table 8, TEMPEST empirically benefits from the complementary information preserved across bit rates, achieving higher accuracy than using a single encoding. This effect is analogous to multi-crop evaluation in vision tasks, where multiple views of the same input provide additional robustness. Our results indicate that multi-bitrate inference offers a simple yet effective strategy for improving compressed-domain audio classification.

5 Conclusions

We have introduced TEMPEST a novel approach for efficient semantic understanding of compressed multimedia data directly from their byte representations. Our method is a lightweight ensemble of networks, that leverages the inherent structure of compressed byte streams to enable efficient semantic classification from a compact data representation without requiring full decoding. We demonstrated TEMPEST effectiveness for classification tasks across multiple datasets, and 2 Compressed file formats. Our proposal achieves classification accuracy on par with raw data transformer models while reducing the length of this feature sequence by a factor of 3, and the size of the attention matrix by a factor of 11.

These efficiency gains are particularly significant for large-scale applications that involve processing millions of media files, in addition our approach bypasses the need for file-format decoding or raw form storage. Our findings indicate that some standard techniques (like data augmentation) are directly transferable to compressed data, moreover, there are some interesting novel properties in the compressed domain as the multi bit rate inference, or the use of multiple bit rates at training time as data augmentation. Beyond the current findings, this work lays the foundation for efficient byte-level semantic modelling of compressed files. Future directions include exploring modality-agnostic evaluations, large-scale pre-training, and further architectural optimizations to enhance efficiency.

AI USE STATEMENT

In this work, we used generative AI tools for: implementing methods.

We have not used generative AI tools for: generate synthetic data sets, help develop theoretical models or conceptual frameworks, formulate mathematical claims, provide critical ingredients for proving mathematical claims, assist in the writing of proofs, propose or refine hypotheses, design or provide feedback on research methodology or experiments, assist with translation, clean and reformat dataset, support qualitative and thematic data analysis, and interpret results.

Additionally, we used generative AI tools for: create or edit software code, edit a research paper to improve readability, and draft parts of a research paper.

We have reviewed all AI-assisted work manually, inspecting and verifying the suggested code, and proof reading every section where an LLM assisted on the writing. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We regard reproducibility as a priority. For experimentation we set the random seeds for PyTorch, CUDA, and python, which enforces deterministic behavior in our experiments. Upon acceptance, we will release the TEMPEST model weights, along with the full code to reproduce its training. We also include the hyper-parameter configuration in this paper.

REFERENCES

Information technology — digital compression and coding of continuous-tone still images — requirements and guidelines, 1992. URL https://www.itu.int/rec/T-REC-T.81/en.

Information technology — coding of moving pictures and associated audio for digital storage media at up to about 1,5 mbit/s — part 3: Audio, 1993. URL https://www.iso.org/standard/22412.html.

Anurag Arnab, Mostafa Dehghani, Georg Heigold, Chen Sun, Mario Lučić, and Cordelia Schmid. Vivit: A video vision transformer. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 6836–6846, 2021.

Daniel Bolya, Cheng-Yang Fu, Xiaoliang Dai, Peizhao Zhang, Christoph Feichtenhofer, and Judy Hoffman. Token merging: Your vit but faster. *arXiv preprint arXiv:2210.09461*, 2022.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, David Belanger, Lucy Colwell, and Adrian Weller. Rethinking attention with performers. In *International Conference on Learning Representations (ICLR)*, 2021.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Jort F Gemmeke, Daniel PW Ellis, Dylan Freedman, Aren Jansen, Wade Lawrence, R Channing Moore, Manoj Plakal, and Marvin Ritter. Audio set: An ontology and human-labeled dataset for audio events. In *2017 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pp. 776–780. IEEE, 2017.

Yuan Gong, Yu-An Chung, and James Glass. Ast: Audio spectrogram transformer. *arXiv preprint arXiv:2104.01778*, 2021.

Yuan Gong, Cheng-I Lai, Yu-An Chung, and James Glass. Ssast: Self-supervised audio spectrogram transformer. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 10699–10709, 2022.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. URL http://www.deeplearningbook.org.

Lionel Gueguen, Alex Sergeev, Ben Kadlec, Rosanne Liu, and Jason Yosinski. Faster neural networks straight from jpeg. *Advances in Neural Information Processing Systems*, 31, 2018.

Sylvain Gugger, Lysandre Debut, Thomas Wolf, Philipp Schmid, Zachary Mueller, Sourab Mangrulkar, Marc Sun, and Benjamin Bossan. Accelerate: Training and inference at scale made simple, efficient and adaptable. https://github.com/huggingface/accelerate, 2022.

Joakim Bruslund Haurum, Sergio Escalera, Graham W Taylor, and Thomas B Moeslund. Which tokens to use? investigating token reduction in vision transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 773–783, 2023.

Geoffrey E. Hinton and Ruslan R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

Maxwell Horton, Sachin Mehta, Ali Farhadi, and Mohammad Rastegari. Bytes are all you need: Transformers operating directly on file bytes. *arXiv preprint arXiv:2306.00238*, 2023a.

Maxwell Horton, Sachin Mehta, Ali Farhadi, and Mohammad Rastegari. Bytes are all you need: Transformers operating directly on file bytes. *arXiv preprint arXiv:2306.00238*, 2023b.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

Yann Le and Xuan Yang. Tiny imagenet visual recognition challenge. *CS 231N*, 7(7):3, 2015.

Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

Samuel G Müller and Frank Hutter. Trivialaugment: Tuning-free yet state-of-the-art data augmentation. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 774–782, 2021.

Library of Congress. Mp3 file format. Library of Congress Digital Preservation, 2024. URL https://www.loc.gov/preservation/digital/formats/fdd/fdd000105.shtml. Accessed: 2024-05-27.

Jeongsoo Park and Justin Johnson. Rgb no more: Minimally-decoded jpeg vision transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 22334–22346, 2023.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf.

William B Pennebaker and Joan L Mitchell. *JPEG: Still image data compression standard*. Springer Science & Business Media, 1992.

Juan C Pérez, Alejandro Pardo, Mattia Soldan, Hani Itani, Juan Leon-Alcazar, and Bernard Ghanem. Compressed-language models for understanding compressed file formats: a jpeg exploration. *arXiv preprint arXiv:2405.17146*, 2024.

Karol J Piczak. Esc: Dataset for environmental sound classification. In *Proceedings of the 23rd ACM international conference on Multimedia*, pp. 1015–1018, 2015.

Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. Efficient transformers: A survey. *arXiv preprint arXiv:2009.06732*, 2020.

Ilya O Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, Mario Lucic, and Alexey Dosovitskiy. Mlp-mixer: An all-mlp architecture for vision. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 24261–24272. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/cba0a4ee5ccd02fda0fe3f9a3e7b89fe-Paper.pdf.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Vinay Verma, Nikita Agarwal, and Nitin Khanna. Dct-domain deep convolutional neural networks for multiple jpeg compression classification. *Signal Processing: Image Communication*, 67:22–33, 2018.

Sinong Wang, Belinda Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*, 2020.

Pete Warden. Speech commands: A dataset for limited-vocabulary speech recognition. *arXiv preprint arXiv:1804.03209*, 2018.

Shangda Wu, Xu Tan, Zili Wang, Rui Wang, Xiaobing Li, and Maosong Sun. Beyond language models: Byte models are digital world simulators. *arXiv preprint arXiv:2402.19155*, 2024.

Yunyang Xiong, Zhanpeng Zeng, Rudrasis Chakraborty, Mingxing Tan, Glenn Fung, Yin Li, and Vikas Singh. Nyströmformer: A nyström-based algorithm for approximating self-attention. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2021.

Lili Yu, Dániel Simig, Colin Flaherty, Armen Aghajanyan, Luke Zettlemoyer, and Mike Lewis. Megabyte: Predicting million-byte sequences with multiscale transformers. *Advances in Neural Information Processing Systems*, 36, 2024.

Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 6023–6032, 2019.

A Appendix

Implementation Details. We implement TEMPEST in PyTorch Paszke et al. (2019) using the Accelerate library Gugger et al. (2022). The same overall architecture is employed across all datasets and formats. Both networks share the same dimensionality for the Key, Query, and Value projections, which we fix at 216, and use feed-forward layers of dimension 864. Unless otherwise stated, we set $l' = 1$ (i.e., a single token per block $\mathcal{H}_k$) and weight the reconstruction and classification losses equally ($\lambda = 1.0$). We use two sinusoidal positional encodings Vaswani et al. (2017), one for the individual bytes in the compressed block (see light blue P.E. block left in figure 1), and another one for the encoded block inside the stream (blue P.E. block right in figure 1). All reported TEMPEST models contain 5.8M parameters unless otherwise stated.

For audio experiments, we adopt a data augmentation regime inspired by Trivial Augment Müller & Hutter (2021) and CutMix Yun et al. (2019). Trivial Augment is applied directly to the raw audio waveform, which is subsequently transcoded into a noisy MP3 stream. CutMix is applied to both audio and image data: a subset of compressed blocks is replaced with blocks at the same starting and ending positions from another element in the batch, while the corresponding labels are

Table 9: Optimization and augmentation hyperparameters used to train TEMPEST on CIFAR-10.

Hyperparameter	**Value**
Optimization	
Optimizer	AdamW
Base learning rate	5
Tokenizer learning-rate multiplier	10
Weight decay	0.05
Gradient-norm clipping	1.0
LR schedule	cosine
Warmup epochs	22
Epochs after warmup	240
Augmentation	
RandAugment operations	5
RandAugment magnitude	1
Random-resized-crop min. area	0.8
CutMix α	2.0
CutMix β	2.0
CutMix probability	1.0
Masked-pixel ratio	0.0

linearly blended, enabling soft class assignments during training. For ESC-50 and SC2, models are trained on entire audio clips. For AudioSet, we employ a two-stage training strategy: pretraining on 2-second clips followed by fine-tuning on 4-second clips.

For JPEG we use 4:4:4 chroma sampling (*i.e* no subsample) and the standard Huffman tables, and we insert a restart marker after every MCU. The MCU bytes depend only on the content of one 8×8 pixel region, across the three color components. For CIFAR-10 and CIFAR-100 we rescale the image to 64x64 such that we have 64 total MCUs (tokens), this is the same number of tokens for the ViT baseline.

A.1 Compressed Encoding Details

Both MP3 and JPEG formats allow for a limited coupling between adjacent blocks. While this results in a small gain in the coding efficiency, it breaks the premise of independent block decoding in our paper. To ensure that each block can be decoded on its own, we set one standard encoder flag in the JPEG and MP3 encoder.

MP3 MP3 streams are encoded with the bit reservoir disabled, this forces the data of each individual frame to lie entirely within a single block.

JPEG For JPEG files, we set a restart interval of one MCU, this places a byte-aligned marker after every MCU at the cost of two additional bytes per MCU. MCU encoding remains unchanged, their lengths remain variable, and are padded as described in Section **??**. For simplicity, we also instruct the encoder to remove the JPEG header. For the same quality setting, this preamble is byte-identical across the entire dataset and brings no information about the image data.

A.2 Hyper parameters

We close the supplementary material reporting the training hyper-parameters for each individual datasets.

Table 10: Optimization and augmentation hyperparameters used to train TEMPEST on CIFAR-100.

Hyperparameter	**Value**
Optimization	
Optimizer	AdamW
Base learning rate	5×10^{-4}
Tokenizer learning-rate multiplier	10
Weight decay	0.05
Gradient-norm clipping	1.0
LR schedule	cosine
Warmup epochs	24
Epochs after warmup	240
Augmentation	
RandAugment operations	5
RandAugment magnitude	9
Random-resized-crop min. area	0.8
CutMix α	2.0
CutMix β	2.0
CutMix probability	1.0
Masked-pixel ratio	0.0

Table 11: Optimization and augmentation hyperparameters used to train TEMPEST on Tiny-ImageNet.

Hyperparameter	**Value**
Optimization	
Optimizer	AdamW
Base learning rate	1×10^{-3}
Tokenizer learning-rate multiplier	2
Weight decay	0.05
Gradient-norm clipping	1.0
LR schedule	cosine
Warmup epochs	30
Epochs after warmup	270
Augmentation	
RandAugment operations	5
RandAugment magnitude	9
Random-resized-crop min. area	0.9
CutMix α	5.0
CutMix β	1.0
CutMix probability	1.0
Masked-pixel ratio	0.0

Table 12: Optimization and augmentation hyperparameters used to train TEMPEST on Speech Commands v2.

Hyperparameter	**Value**
Optimization	
Optimizer	AdamW
Base learning rate	5×10^{-3}
Tokenizer learning-rate multiplier	1
Weight decay	0.2
Gradient-norm clipping	0.5
LR schedule	cosine
Warmup epochs	13
Epochs after warmup	150
Augmentation	
CutMix α	1.5
CutMix β	3.5
CutMix probability	0.5
Masked-byte ratio	0.01

Table 13: Optimization and augmentation hyperparameters used to train TEMPEST on AudioSet with 2 s and 4 s crops.

Hyperparameter	**2 s**	**4 s**
Optimization		
Optimizer	AdamW	AdamW
Base learning rate	1×10^{-3}	1×10^{-4}
Tokenizer learning-rate multiplier	5	5
Weight decay	0.05	0.05
Gradient-norm clipping	0.5	0.5
LR schedule	cosine, 2 cycles	cosine
Warmup epochs	20	5
Epochs after warmup	210	50
Augmentation		
CutMix α	3.0	3.0
CutMix β	4.0	4.0
CutMix probability	0.3	0.3
Masked-byte ratio	0.01	0.01

Table 14: Optimization and augmentation hyperparameters used to train TEMPEST on ESC-50.

Hyperparameter	**Value**
Optimization	
Optimizer	AdamW
Base learning rate	2×10^{-4}
Tokenizer learning-rate multiplier	20
Weight decay	0.1
Gradient-norm clipping	1.0
LR schedule	cosine
Warmup epochs	8
Epochs after warmup	80
Augmentation	
CutMix α	1.0
CutMix β	4.5
CutMix probability	0.8
Masked-byte ratio	0.2