Efficient Mismatch-Tolerant Coding for Model-Driven Compression

Aviv Adler [1] **Jennifer Tang** [2]

Abstract

A central insight in lossless data compression is the close connection between probabilistic next-symbol prediction and efficient sequence compression, whereby predictive models can be combined with classical coding techniques to achieve strong compression performance. Applying this approach with powerful modern learned models, such as LLMs, has been shown to achieve markedly better compression than traditional techniques across a wide range of domains. However, significant practical challenges remain, including *model non-determinism*, in which a model produces different predictions on different machines despite identical parameters and inputs; such mismatches between the encoder and decoder can lead to complete decoding failure. Probability Matching Interval Coding (PMATIC) was recently introduced as a drop-in framework for mismatch-robust coding and shown to enable reliable compression and decompression in the presence of bounded prediction mismatch (Adler & Tang, 2026). In this work, we present a generalization of PMATIC that allows the incorporation of tight theoretical results into the design and more flexible parameter optimization, resulting in substantial improvements in compression efficiency and robustness.

1. Introduction

Lossless data compression is the problem of encoding data from some general domain (e.g., text, video, and so forth) using as few bits as possible, while still allowing the original data to be recovered exactly, in order to facilitate storage or transmission. Since lossless compression works by assigning shorter representations for likelier inputs, all lossless compression algorithms rely, at least implicitly, on a probabilistic model of the data, and its effectiveness depends on how well this model matches the true data distribution (Shannon, 1948). This insight inspired the development of *model-driven* compression, which explicitly uses a predictive model to sequentially estimate the probability of each symbol given its context and is paired with an entropy coding scheme, most commonly arithmetic coding (Pasco, 1976; Rissanen, 1976), to produce a compact representation. Arithmetic coding is particularly well suited to this setting due to its ability to adapt to context-dependent probability distributions, and achieves rates approaching the entropy of the true data distribution when the model is accurate. A major advantage of model-driven compression is that, in principle, it works with any predictive model, so developments in predictive models in any domain can be converted directly into compression algorithms.

Model-driven compression has been well-studied and has been implemented with models ranging from early Markov models to modern learned predictors (Cleary & Witten, 1984; Schmidhuber & Heil, 1996; Deletang et al., 2024). These modern predictors, including recurrent networks, transformers, and large language models (LLMs) have been shown to be effective for lossless compression across domains ranging from text and images (Bellard, 2021; Deletang et al., 2024; Chen et al., 2025) to time-series power data (Ma et al., 2022) and neural network training checkpoints (Kim & Belyaev, 2025), achieving better compression ratios than the standard compression tools.

1.1. Prediction Mismatch and Model Non-determinism

Despite its proven effectiveness at achieving excellent compression rates across many domains, model-driven compression faces several challenges. First, the model itself must be available to both the encoder and decoder, which may be difficult particularly when using large models such as LLMs on resource-constrained devices. Second, encoding or decoding requires a prediction to be made for each token, which can be prohibitively expensive (in contrast to extremely fast traditional compression algorithms such as gzip (Mittu et al., 2024)). While these are important challenges, reducing the size and computational burden of learned models is an extremely active area of research in machine learning which will directly benefit future appli-

[1]Analog Devices Inc., Boston, MA, USA [2]Department of Mathematics and Computer Science, College of the Holy Cross, Worcester, MA, USA. Correspondence to: Jennifer Tang <jtang@holycross.edu>.

Proceedings of the 43^{rd} International Conference on Machine Learning, Seoul, South Korea. PMLR 306, 2026.

cations of model-driven compression.

Instead, we consider another challenge to model-driven compression: *model non-determinism*, a well-known phenomenon where different machines running the same model (using identical weights) on the same input can nevertheless get different predictions for the next token, which can arise from floating-point arithmetic, GPU parallelism, or library- and hardware-level differences. Non-determinism has been widely observed in practice (Cooper et al., 2022; Chen et al., 2022; Eryilmaz et al., 2024; Atil et al., 2025), and GPU libraries often even warn users about it (NVIDIA Corporation, 2025a;b).

Non-determinism poses a serious problem for explicit model-driven compression pipelines because arithmetic coding requires the decoder to reconstruct exactly the same probability distribution as the encoder at every step (Witten et al., 1987), and even minute discrepancies can quickly desynchronize them. It has also been noted that this phenomenon poses a serious problem for other neural compression methods (Ballé et al., 2018) and for reproducibility in machine learning research (Semmelrock et al., 2025).

Prior work on addressing the problem of non-determinism has focused on ensuring fully deterministic computation, such as using integer computation rather than floating-point (Ballé et al., 2018), or through modified backends which enforce deterministic order of operations (He & Thinking Machines Lab, 2025). However, these approaches impose unavoidable efficiency or performance costs on the models and require considerable modification of the networks or systems used.

1.2. Mismatch-Tolerant Coding

Recently, (Adler & Tang, 2026) formulated a novel alternative approach to this problem, in which the arithmetic coding step is replaced by a coding algorithm that is robust to bounded amounts of 'prediction mismatch'; to solve it, they introduced Probability Matching Interval Coding (PMATIC) as a drop-in replacement for arithmetic coding which is robust against bounded mismatch between the encoder and decoder logit scores, and validated it with synthetic mismatches. PMATIC extends arithmetic coding by quantizing the space of distributions and adding buffers between quantization bins to ensure that the encoder and decoder distributions are quantized to the same values.

However, PMATIC as originally proposed incurs considerable additional code length in order to tolerate relatively modest amounts of mismatch, with larger mismatches being impossible to accommodate. This was due to two factors: first, the algorithm relies on a loose theoretical result to set the buffers, leading to unnecessarily large buffers, particularly for probability values near 0 and 1; second, it uses fixed-size quantization bins, which may not be well-suited for the predictions given by the model. Together, these factors limit the practical operating regime of the original method, motivating the need for sharper theoretical bounds and more adaptive algorithmic design.

Another approach to mismatch-tolerant coding, motivated by model-driven compression under a slightly different mismatch setting, was also recently introduced (Hu & Tang, 2026), in which the encoder gives the decoder a range in which the true next token's probability lies, plus a bit-string identifying the true token from all the other tokens whose probability may lie in this range.

1.3. Contributions

This work builds on the Probability Matching Interval Coding (PMATIC) and bounded logit mismatch framework introduced in (Adler & Tang, 2026) by providing a more general formulation that enables a number of improvements:

1. *Tight theoretical mismatch bounds:* We introduce Proposition 1 in place of a weaker result from (Adler & Tang, 2026), allowing PMATIC to better account for the potential mismatch between the encoder and decoder distributions and send less additional information. This result is worst-case tight given the bounded mismatch setting.

2. *Non-uniform binning:* We allow the quantization bins (and the centers representing them) to be non-uniform in order to optimize their placement, which is especially helpful for reducing entropy.

3. *Message-optimized parameters:* The encoder now optimizes the bin boundaries, bin centers, and helper-bit encoding probabilities according to the message, which are then attached for the decoder as a prefix to the compressed message.

This achieves a significantly better compression-tolerance tradeoff than the original version of PMATIC, providing 10 to 15 times as much robustness with approximately the same compression ratio. We validate our theoretical results and demonstrate the improved performance on text data with both synthetic and real instances of non-determinism. For the remainder of this work, we use the term 'PMATIC' to refer to the algorithm in general; references to the original formulation of (Adler & Tang, 2026) are made explicitly.

Remark 1. *PMATIC can also use only a subset of these additions, if desired – for instance, in settings where the message itself cannot be directly used to optimize the algorithm parameters, prearranged bins can be used according to a statistical description of the expected distribution of*

messages instead (thus keeping the tight theoretical result and non-uniform binning).

2. Problem Statement

This paper builds on the original PMATIC framework introduced in (Adler & Tang, 2026). We therefore largely follow the same problem formulation and notation, and restate certain definitions and assumptions from (Adler & Tang, 2026) for clarity and self-containment.

We consider a lossless data compression problem in which an encoder must compress an input token string $\boldsymbol{x} = x(1)\dots x(n)$ whose entries are taken from a finite *token alphabet* $\mathcal{A}$; we denote the substring up to $x(i)$ (inclusive) as $\boldsymbol{x}^i := x(1)\dots x(i)$. The encoder and decoder have respective prediction models[1] $\mathrm{M}^{\mathrm{Enc}}$ and $\mathrm{M}^{\mathrm{Dec}}$ which produce a set of *next-token logits* for each $x(i)$ using its *context* $\boldsymbol{x}^{i-1}$:

$$\boldsymbol{u}(i) := \mathrm{M}^{\mathrm{Enc}}(\boldsymbol{x}^{i-1}) \quad \text{and} \quad \boldsymbol{v}(i) := \mathrm{M}^{\mathrm{Dec}}(\boldsymbol{x}^{i-1}) \tag{1}$$

where $\boldsymbol{u}(i), \boldsymbol{v}(i)$ are real-valued (or, specifically, floating-point valued) weight vectors on $\mathcal{A}$. These logit vectors then induce their respective next-token probability distributions $\boldsymbol{p}(i), \boldsymbol{q}(i)$ through the *softmax* function

$$p_x(i) = \mathrm{softmax}_x(\boldsymbol{u}(i)) := \frac{e^{u_x(i)}}{\sum_{x'\in\mathcal{A}} e^{u_{x'}(i)}} \tag{2}$$

$$\text{and} \quad q_x(i) = \mathrm{softmax}_x(\boldsymbol{v}(i)) := \frac{e^{v_x(i)}}{\sum_{x'\in\mathcal{A}} e^{v_{x'}(i)}}. \tag{3}$$

This is the typical way that LLMs (and many other modern predictive or generative models) represent their predictions.

We denote the encoder and decoder as functions depending on prediction models $\mathrm{M}^{\mathrm{Enc}}$ and $\mathrm{M}^{\mathrm{Dec}}$ respectively: the encoder uses $\mathrm{M}^{\mathrm{Enc}}$ and input token string $\boldsymbol{x}$ to produce the *compressed message* $\boldsymbol{y}$, which the decoder then turns into a *decoded token string* $\hat{\boldsymbol{x}}$ using $\mathrm{M}^{\mathrm{Dec}}$. Since the goal is robust lossless compression, we require that $\hat{\boldsymbol{x}} = \boldsymbol{x}$ provided that certain conditions (which we discuss in Section 2.1) hold on the difference between $\mathrm{M}^{\mathrm{Enc}}$ and $\mathrm{M}^{\mathrm{Dec}}$'s outputs; following (Adler & Tang, 2026), we say this is *mismatch-tolerant* given the constraints.

Finally, the objective is not only to be mismatch-tolerant but also *efficient*, both in terms of compression (minimizing the size of the compressed message $\boldsymbol{y}$) and in terms of computation (time, memory, and power usage). However, the computational performance of model-driven compression is often dominated by the performance of the model, particularly when using larger and more powerful prediction models such as modern LLMs, since a prediction must be generated for every token in the message. Therefore, aside from ensuring that PMATIC runs reasonably efficiently compared to the LLMs we test with, we focus on the compression efficiency.

[1] In implementation, these are nominally the same model, but their predictions may differ due to non-determinism.

2.1. The Bounded Prediction Mismatch Setting

Because the encoder and decoder evaluate the same LLM on identical inputs, it is natural to model any discrepancy between their resulting logits as being relatively limited. We therefore assume the existence of some $\varepsilon > 0$ such that the logits differ by at most ε in the L_∞ norm, i.e.,

$$\|\boldsymbol{u} - \boldsymbol{v}\|_\infty := \max_{x\in\mathcal{A}} |u_x - v_x| \le \varepsilon\,. \tag{4}$$

As noted, this bounded logit mismatch assumption is identical to that used in (Adler & Tang, 2026). However, our analysis of its implications differs substantially, leading to much tighter bounds than those previously available.

Due to the relationship between logits and probabilities, it is useful to consider probabilities both in the original probability space and also when transformed into *log-odds space* by the function $\mathrm{logodds}(p) := \ln\left(\frac{p}{1-p}\right)$; this transforms probabilities in $(0,1)$ to the real numbers $\mathbb{R}$, with probabilities of $< 1/2$ having negative log-odds and probabilities of $> 1/2$ having positive log-odds.

We now consider how a bound on logit differences (4) translates into a bound on the difference between the induced probabilities, including conditional probabilities:

Proposition 1. *Let $\boldsymbol{u}, \boldsymbol{v}$ be logit vectors inducing probability distributions $\boldsymbol{p} = \mathrm{softmax}(\boldsymbol{u})$ and $\boldsymbol{q} = \mathrm{softmax}(\boldsymbol{v})$ over $\mathcal{A}$, such that $\|\boldsymbol{u} - \boldsymbol{v}\|_\infty \le \varepsilon$. Let $S' \subset S \subseteq \mathcal{A}$ such that $S, S' \neq \emptyset$ and $S' \neq S$, and let $p^* := \mathbb{P}_{x\sim\boldsymbol{p}}[x \in S' \,|\, x \in S]$ and $q^* := \mathbb{P}_{x\sim\boldsymbol{q}}[x \in S' \,|\, x \in S]$. Then,*

$$|\,\mathrm{logodds}(q^*) - \mathrm{logodds}(p^*)| \le 2\varepsilon\,. \tag{5}$$

Proof. We write $\ln(\frac{p^*}{1-p^*})$ and $\ln(\frac{q^*}{1-q^*})$ in terms of $\boldsymbol{u}, \boldsymbol{v}$:

$$\begin{aligned} p^* = \frac{\sum_{x\in S'} e^{u_x}}{\sum_{x\in S} e^{u_x}} \implies 1 - p^* &= \frac{\sum_{x\in S\setminus S'} e^{u_x}}{\sum_{x\in S} e^{u_x}} \\ \implies \frac{p^*}{1-p^*} &= \frac{\sum_{x\in S'} e^{u_x}}{\sum_{x\in S\setminus S'} e^{u_x}} \end{aligned} \tag{6}$$

since their denominators cancel. We can treat q^* similarly, and then invoke $v_x \in [u_x \pm \varepsilon]$ for all $x \in \mathcal{A}$ to get:

$$\begin{aligned} \frac{q^*}{1-q^*} &= \frac{\sum_{x\in S'} e^{v_x}}{\sum_{x\in S\setminus S'} e^{v_x}} \\ &\le \frac{\sum_{x\in S'} e^{u_x+\varepsilon}}{\sum_{x\in S\setminus S'} e^{u_x-\varepsilon}} = \exp(2\varepsilon)\frac{p^*}{1-p^*}\,, \end{aligned} \tag{7}$$

with the result that $\frac{q^*}{1-q^*} \ge \exp(-2\varepsilon)\frac{p^*}{1-p^*}$ derived analogously. Then, applying $\ln(\cdot)$ to both sides yields $\mathrm{logodds}(q^*) \in [\mathrm{logodds}(p^*) \pm 2\varepsilon]$ and we are done. □

Remark 2. *Proposition 1 is tight in the worst case under bounded mismatch, as the inequality (7) is tight.*

3. The Framework and Algorithm

We first give a high level description of the PMATIC framework, generalized from (Adler & Tang, 2026). PMATIC works by first representing each token with a unique, fixed-length[2] bitstring, which we call the *longform*; then each next-token probability distribution can be converted into a sequence of next-bit probability values in $[0, 1]$ (representing the probability that the next bit is `1`, conditional on all previous bits). The interval $[0, 1]$ is quantized into a set of bins, each with a representative value; for each bit in the input bitstring, the encoder determines which bin contains the next-bit probability. If the encoder next-bit probability is sufficiently in the interior of the bin, then the decoder next-bit probability is guaranteed to fall into the same bin (given the logit error bound ε), and the encoder and decoder can both use the representative value of that bin to encode / decode the next bit (via arithmetic coding). Otherwise, if the encoder's next-bit probability falls too close to a boundary between two bins, the encoder and decoder can use the boundary value itself as the encoding probability instead. However, since the decoder does not automatically know whether the encoder is using the bin representative or a boundary value as the next-bit encoding probability, the encoder must include a *helper bit* which denotes which one to use; this helper bit is then itself included in the message via arithmetic coding, with its own (fixed and known) encoding / decoding probability.

3.1. Arithmetic Coding Overview

Arithmetic coding (AC) is an entropy-coding algorithm which encodes a sequence of symbols by using a sequence of predicted next-symbol probability distributions to efficiently allocate information. At each step, the encoder takes as input (i) a probability distribution $\boldsymbol{p}$ over the next symbol, (ii) the true next symbol x^*, and (iii) an internal state representing all previously encoded symbols. The encoding update refines this state accordingly; we denote this operation by $\mathrm{Enc}_{\mathrm{AC}}(\boldsymbol{p}, x^*)$. After all symbols are processed, the encoder outputs a finite-length bitstring $\boldsymbol{y}$ representing the final state.

The decoder receives this bitstring and runs essentially the same sequence of updates, extracting the original symbols along the way. Given the same internal state and the same predicted distribution $\boldsymbol{p}$, the decoding update $\mathrm{Dec}_{\mathrm{AC}}(\boldsymbol{y}, \boldsymbol{p})$ both recovers the next symbol and performs the same update to the internal state, thus keeping the encoder and decoder synchronized over the whole message.

If the encoder and decoder predicted distributions match exactly at each step, arithmetic coding recovers the original message, and the expected code length converges to the cross entropy between the predicted and true next-symbol distributions. However, even extremely small mismatches between the encoder and decoder distributions will cause their internal states to diverge, leading to rapid and irreversible desynchronization. This sensitivity to probability mismatch is the central challenge addressed by PMATIC.

3.2. PMATIC Overview

We now define PMATIC in terms of a set of parameters which the encoder and decoder share:

- A *robustness guarantee* of $\varepsilon > 0$, corresponding to the maximum mismatch (in L_∞) between the encoder and decoder token prediction logits under which the PMATIC encoding guarantees correctness.
- A *longform mapping* of each token $x \in \mathcal{A}$ to a unique bitstring $\boldsymbol{b}(x)$ of fixed length $\ell := \lceil \log_2 |\mathcal{A}| \rceil$.[3]
- A set of m *quantization bins* partitioning $[0, 1]$ with boundaries $0 = a_0 \le a_1 \le \cdots \le a_{m-1} \le a_m = 1$, so that the kth bin is $I_k := [a_{k-1}, a_k)$; we also denote by $r_k \in I_k$ their *representative values*.[4] We assume the log-odds of the boundaries are sufficiently spaced $|\operatorname{logodds}(a_k) - \operatorname{logodds}(a_{k-1})| \ge 8\varepsilon$ for all k.
- A helper-bit encoding value $\rho \in [0, 1]$.
- A set of *bin interiors* $I_k^\varepsilon \subset I_k$, which guarantee (given that the next-token logits differ by at most ε) that

$$p^* \in I_k^\varepsilon \implies q^* \in I_k \tag{8}$$

  where p^*, q^* are the next-bit probabilities of the encoder and decoder, respectively (see below for more).

  Formally, we define:

$$I_k^\varepsilon = \big[\operatorname{logodds}^{-1}(\operatorname{logodds}(a_{k-1}) + 2\varepsilon), \operatorname{logodds}^{-1}(\operatorname{logodds}(a_k) - 2\varepsilon)\big] \tag{9}$$

  with special cases that if $k = 1$ or m, then the lower and upper boundaries of I_k^ε are 0 and 1 respectively.[5]

[2] In theory one could use variable-length representations, but there are no obvious advantages to doing so.

[3] In this work, we treat this assignment as arbitrary, though there may be ways to optimize it.

[4] In (Adler & Tang, 2026), the quantization bins were of uniform size and their representative values were their centers; here we generalize to allow non-uniform binning and arbitrary centers.

[5] In (Adler & Tang, 2026) these interiors were defined according to a weaker theoretical result.

- We define the complements of the bin interiors, which we call *danger-zones* and denote as open intervals

$$J_k^\varepsilon = \big(\text{logodds}^{-1}(\text{logodds}(a_k) \pm 2\varepsilon)\big) \tag{10}$$

for $k = 1, \ldots, m-1$. Note that each danger-zone is associated with a bin boundary a_k, and that if the encoder next-bit probability falls into J_k^ε then (given the ε logit error bound) the decoder next-bit probability can fall into *either* I_k or I_{k+1}.

Note that these parameters may be: (i) prearranged or hard-coded; (ii) given as a prefix to the encoded message (allowing the encoder to optimize them, at the cost of having to communicate the parameters within the message itself); (iii) computed in a fixed manner from other parameters; or (iv) a combination of these.

PMATIC both encodes and decodes the input $\boldsymbol{x} = x(1)\ldots x(n)$ from its longform representation, in which each token $x(i)$ is replaced with its longform bitstring $\boldsymbol{b}(i) := \boldsymbol{b}(x(i))$, which are then concatenated:

$$\boldsymbol{b}^* = \boldsymbol{b}(1)\ldots\boldsymbol{b}(n) = b_1(1)\ldots b_\ell(1)\ldots b_1(n)\ldots b_\ell(n) \tag{11}$$

We refer to the bits b_j^* in $\boldsymbol{b}^*$ as the *token bits*.

For $j \in 1, \ldots, n\ell$, let $j = (i-1)\ell + j'$, so that $b_j^* = b_{j'}(i)$, and let $\boldsymbol{p}(i) = \text{softmax}(\boldsymbol{u}(i))$ and $\boldsymbol{q}(i) = \text{softmax}(\boldsymbol{v}(i))$ be the respective predictions of the encoder and decoder models for token $x(i)$ (given context $x(1)\ldots x(i-1)$). We then define the encoder and decoder next-bit probabilities:

$$p_j^* := \mathbb{P}_{\boldsymbol{p}(i)}[b_{j'}(i) = \texttt{1} \,|\, b_1(i), \ldots, b_{j'-1}(i)] \tag{12}$$

$$q_j^* := \mathbb{P}_{\boldsymbol{q}(i)}[b_{j'}(i) = \texttt{1} \,|\, b_1(i), \ldots, b_{j'-1}(i)] \tag{13}$$

This is the conditional probability of b_j^* given all previous bits $b_1^*, \ldots, b_{j-1}^*$ and the encoder or decoder model, since $b_1^*, \ldots, b_{(i-1)\ell}^*$ define the context for $\boldsymbol{p}(i)$ or $\boldsymbol{q}(i)$, while $b_{(i-1)\ell+1}^* = b_1(i)$ through $b_{(i-1)\ell+(j'-1)}^* = b_{j'-1}(i)$ condition the distribution of $b_j^* = b_{j'}(i)$ given $\boldsymbol{p}(i)$ or $\boldsymbol{q}(i)$.

PMATIC works by pairing each token bit b_j^* with a *helper bit* b_j', and then using arithmetic coding to encode an *expanded message* alternating helper and token bits:

$$\boldsymbol{b}'^* := b_1' b_1^* b_2' b_2^* \ldots b_{n\ell}' b_{n\ell}^* \tag{14}$$

The helper bit b_j' is used to help the encoder and decoder coordinate on a probability value to use for b_j^*.

3.3. The PMATIC Encoder

For bit b_j^*, the encoder determines helper bit b_j' and token-bit-encoding probability $\hat{p}_j$ as follows:

$$(b_j', \hat{p}_j) = \begin{cases} (\texttt{0}, r_k) & \text{if } p_j^* \in I_k^\varepsilon \text{ for some } k \\ (\texttt{1}, a_k) & \text{if } p_j^* \in J_k^\varepsilon \text{ for some } k \end{cases} \tag{15}$$

and encodes helper bit b_j' and token bit b_j^* by executing

$$\text{Enc}_{\text{AC}}(b_j', \rho), \text{Enc}_{\text{AC}}(b_j^*, \hat{p}_j)\,. \tag{16}$$

After all the bits in $\boldsymbol{b}'^*$ have been added, PMATIC generates $\boldsymbol{y}$ in the same manner as standard arithmetic coding.

3.4. The PMATIC Decoder

PMATIC decodes the message $\boldsymbol{y}$ sequentially in pairs of helper and token bits. The decoder first decodes the helper bit using the fixed probability ρ:

$$b_j' = \text{Dec}_{\text{AC}}(\boldsymbol{y}, \rho)\,, \tag{17}$$

and then decodes token bit b_j using probability $\hat{q}_j$:

$$b_j = \text{Dec}_{\text{AC}}(\boldsymbol{y}, \hat{q}_j) \text{ where } \hat{q}_j = \begin{cases} r(q_j^*) & \text{if } b_j' = \texttt{0} \\ a(q_j^*) & \text{if } b_j' = \texttt{1} \end{cases} \tag{18}$$

where $r(q_j^*) = r_k$ is the representative value of the quantization bin I_k that q_j^* is in, and $a(q_j^*)$ is the bin boundary point minimizing $|\,\text{logodds}(q_j^*) - \text{logodds}(a(q_j^*))|$.

After decoding all bits (both helper and token), the helper bits are discarded and the token bits converted back to the token string (and then to the original message) using the longform representations of the tokens. PMATIC successfully communicates the message if $\hat{q}_j = \hat{p}_j$ for all token bits b_j; this is discussed further in Section 4.

4. Analysis

Since this work updates PMATIC from the original formulation in (Adler & Tang, 2026), particularly in how the danger-zones are computed, we must give a new analysis of its correctness and performance.

4.1. Correctness

Theorem 1. *Consider an instance of PMATIC with robustness guarantee of $\varepsilon > 0$ applied to the encoder and decoder whose models always return prediction logits $\boldsymbol{u}, \boldsymbol{v}$ satisfying $|\boldsymbol{u} - \boldsymbol{v}|_\infty \leq \varepsilon$ (given identical context). Then, for any input $\boldsymbol{x}$, the decoder will successfully recover $\boldsymbol{x}$ from the encoded message $\boldsymbol{y}$.*

Proof. Arithmetic coding guarantees correctness as long as each symbol is encoded and decoded according to the exact same probability distribution. We consider the *expanded* message generated by PMATIC (including helper bits):

$$\boldsymbol{b}'^* = b_1' b_1^* b_2' b_2^* \ldots b_{n\ell}' b_{n\ell}^*\,. \tag{19}$$

We then consider the probabilities used to encode each bit. All helper bits b_j' are encoded and decoded with probability

ρ, so we can consider only the token bits b_j^*, which are respectively encoded and decoded with probability $\hat{p}_j, \hat{q}_j$. Thus, showing that $\hat{q}_j = \hat{p}_j$ for all j suffices.

Suppose that there is some j for which $\hat{q}_j \neq \hat{p}_j$, and suppose (without loss of generality) that j is the first such index. Then we know that all prior tokens match and hence $|\boldsymbol{u} - \boldsymbol{v}|_\infty \leq \varepsilon$ for the current token. Let $S \subseteq \mathcal{A}$ denote the set of tokens which agree with prior bits, and let $S' \subset S$ denote those whose next longform bit is `1`. If $S' = \emptyset$, then $p_j^* = q_j^* \in \{0, 1\}$ and thus $p_j^*, q_j^* \in I_k^\varepsilon$ for $k = 1$ or m. Thus, $\hat{q}_j = \hat{p}_j = r_k$, contradicting the definition of j.

If $S' \neq \emptyset$, then $p_j^* = \frac{\sum_{i \in S'} e^{u_i}}{\sum_{i \in S} e^{u_i}}$ and $q_j^* = \frac{\sum_{i \in S'} e^{v_i}}{\sum_{i \in S} e^{v_i}}$, so by Proposition 1, $|\operatorname{logodds}(q_j^*) - \operatorname{logodds}(p_j^*)| \leq 2\varepsilon$. There are then two cases: (i) $p_j^* \in I_k^\varepsilon$ (a bin interior) for some k; (ii) $p_j^* \in J_k^\varepsilon$ (a danger-zone) for some k.

(i): $\operatorname{logodds}(I_k^\varepsilon) = [\operatorname{logodds}(a_{k-1}) + 2\varepsilon, \operatorname{logodds}(a_k) - 2\varepsilon]$ and $\operatorname{logodds}(I_k) = [\operatorname{logodds}(a_{k-1}), \operatorname{logodds}(a_k)]$. Thus, $\operatorname{logodds}(p_j^*) \in \operatorname{logodds}(I_k^\varepsilon)$ (since $p_j^* \in I_k^\varepsilon$) and $|\operatorname{logodds}(q_j^*) - \operatorname{logodds}(p_j^*)| \leq 2\varepsilon$ imply that $\operatorname{logodds}(q_j^*) \in \operatorname{logodds}(I_k)$, which means $q_j^* \in I_k$. Then, since $b_j' = \mathtt{0}$ in (i), we have $\hat{p}_j = \hat{q}_j = r_k$, ruling out (i).

In (ii), we similarly have that $p_j^* \in J_k^\varepsilon$, so

$$|\operatorname{logodds}(p_j^*) - \operatorname{logodds}(a_k)| \leq 2\varepsilon \tag{20}$$

$$\implies |\operatorname{logodds}(q_j^*) - \operatorname{logodds}(a_k)| \leq 4\varepsilon \tag{21}$$

and this additionally means that (since the log-odds of the boundaries are spaced at least 8ε apart) that a_k is the closest boundary point in log-odds space to both p_j^* and q_j^*. Thus, in this case, we have $b_j' = \mathtt{1}$ so $\hat{p}_j = \hat{q}_j = a_k$, ruling out case (ii) and completing the proof. □

4.2. Overhead Cost of ε-Robust PMATIC

For this analysis, we make a simplifying assumption that the encoder predictions reflect the actual probabilities of the data. Since compression is done with respect to bits, in this analysis $\log$ denotes base-2 while the natural logarithm is denoted by $\ln$. We also define some useful functions:

- Let $h(p) := p \log\left(\frac{1}{p}\right) + (1-p)\log\left(\frac{1}{1-p}\right)$ be the binary entropy function (the entropy of a Bernoulli random variable with probability p).
- Let $D_{\mathrm{KL}}(p\|q) := p \log\left(\frac{p}{q}\right) + (1-p)\log\left(\frac{1-p}{1-q}\right)$ be the binary Kullback-Leibler divergence.

Similar to (Adler & Tang, 2026), we consider the total *compression overhead* of PMATIC, corresponding to the additional length required to ensure robustness against ε logit error mismatch. This comes from two sources which each contribute overhead in terms of entropy per token bit:

- *Helper bit entropy:* Since we consider the case where the encoder can set the helper bit encoding / decoding probability based on the data itself, given the robustness bound ε and bin boundaries $a_1, \ldots, a_{m-1}$, the encoder can optimize ρ by setting it to be the fraction of helper bits which have value `1` (since that is the true probability of a helper bit being equal to 1). The expected additional encoding length (defining ρ to be the fraction of helper bit `1`'s) is then $h(\rho)$ overhead bits per token bit.
- *Quantization loss:* If the true next-bit probabilities are given by p_j^*, then each token bit carries an expected penalty of $D_{\mathrm{KL}}(p_j^*\|\hat{p}_j)$ bits in quantization loss (loss of efficiency of arithmetic coding due to inaccurate probabilities). It is well known that if a single representation r_k must be chosen for a set of true n_k probabilities p_j^* falling in the same bin interior I_k^ε with the objective of minimizing $\sum_{p_j^* \in I_k^\varepsilon} D_{\mathrm{KL}}(p_j^*\|r_k)$, the optimal value is the average $r_k = \frac{1}{n_k}\sum_{p_j^* \in I_k^\varepsilon} p_j^*$.

Furthermore, if we apply message-optimized parameters, in which the encoder pre-computes their next-bit probabilities on the message and uses those probabilities to generate parameters optimized for the given message, there is an additional *one-time* (per message) overhead cost since those parameters must be attached to the compressed message so that the decoder can use them. In this work, we do not consider optimizing the longform mapping $\boldsymbol{b}(\cdot)$ (either per-message or in advance based on statistical relationships between the tokens), so the parameters optimized are: (i) the quantization bins, including both the number m of bins and their (non-0 or 1) boundaries $a_1, a_2, \ldots, a_{m-1}$; (ii) the bin representative values $r_1, \ldots, r_m$; (iii) the helper-bit encoding value $\rho \in [0, 1]$. Thus, $2m$ values are added for a cost of $\approx 2\gamma m$, where γ is the number of bits needed per value.[6]

As noted above, once the bin boundaries and next-bit probabilities are known, the optimal values of the bin representative values and the helper bit entropy can be immediately computed. Thus, the encoder can compute the full set of next-bit probabilities $p_1^*, \ldots, p_{n\ell}^*$ and then solve an optimization problem where the bin boundaries $\boldsymbol{a} = (a_1, \ldots, a_{m-1})$ (and thus also the number of bins) are selected in order to minimize the theoretical total overhead:

$$f(\boldsymbol{a}) = \sum_{j=1}^{n\ell} D_{\mathrm{KL}}(p_j^*\|\hat{p}_j) + n\ell \cdot h(\rho) + 2\gamma m \tag{22}$$

This depends on the robustness parameter ε through the bin interior boundary function (9).

[6] Although it may be possible to reduce this fixed cost somewhat by using structured bin boundaries, this will be negligible for longer files and we do not consider it in this work.

However, the loss function (22) does not have any obvious structure or smoothness (in fact, it has discontinuities when shifting a boundary changes whether a probability value falls into a bin interior or a danger-zone). Thus, for this work, we use a heuristic greedy bin selection optimization algorithm, where bin boundaries are chosen one at a time, each time minimizing the total loss, stopping once no additional bin can reduce the loss (which is guaranteed because each additional bin incurs a penalty of 2γ). More efficient or effective bin optimization is left for future work.

5. Experiments

We run experiments compressing text data with three LLMs in the 7-8B parameter range, using both standard arithmetic coding (as a baseline) and PMATIC under various levels of mismatch robustness. As a traditional compression baseline, we also compress the files using gzip. See (Adler & Tang, 2026) for comparisons to other baselines.

It is important to note that compression is done with PMATIC in conjunction with the LLM predictor. We thus primarily consider PMATIC's performance in terms of how much additional code length is required to guarantee robustness against a desired level of prediction mismatch, and how this, in combination with the LLM predictor, compares to traditional compression algorithms like gzip.

5.1. Setup and Objectives

We test our PMATIC implementation by compressing text files using three LLMs: Llama 3.1 8B Q4_K_S (4-bit quantized) (Grattafiori et al., 2024); Mistral 7B v0.1 (3-bit quantized); and Qwen 2.5 Instruct 7B (3-bit quantized).[7] These models have vocabularies of size 128256, 32000, and 151643 tokens respectively, and thus require longforms of length 17, 15, and 18 bits per token.

We used five text datasets: (i) articles drawn randomly from Wikipedia, of varying lengths, ranging from 13 to $59,768$ bytes in size; (ii) *Emma* by Jane Austen and (iii) *Hamlet* by William Shakespeare (both in English); (iv) *Candide* by Voltaire (in French); and (v) *Dream of the Red Chamber* by Cao Xueqin (in Chinese). The Wikipedia dataset is divided into individual articles, while the book datasets are each broken up into sections of $5,120$ bytes.[8] We omit experiments in cases where the LLM does not support the language of the text. This dataset overlaps heavily with the one used in (Adler & Tang, 2026), in order to allow for direct comparisons with the original PMATIC formulation.

[7]These are the same models used in (Adler & Tang, 2026), and we use the same rolling-context system; this enables a direct comparison of our results with the original formulation of PMATIC.

[8]Except the last section, which is smaller.

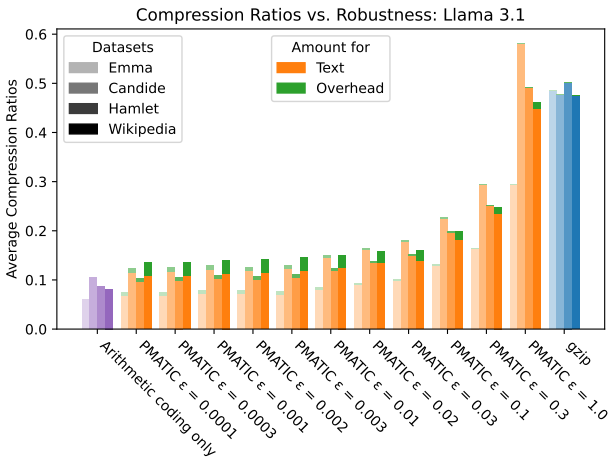

Figure 1. Compression ratios for PMATIC on Llama 3.1 (compressed file size divided by original file size) with different robustness, with comparison to arithmetic coding and gzip. Additional code length due to parameter specification shown in green. These are negligible in some datasets with uniformly large size of files.

To set the bin boundaries, we use a greedy optimization algorithm which considers the set of next-bit probability values for the each message and recursively divides bins in order to minimize (an approximation of) the objective function (22) at each step, stopping when no improvement is possible; see Appendix B for details.[9] We use $\gamma = 8$, resulting in a constant 'penalty' of $2\gamma = 16$ for each bin added. We note that the performance of PMATIC depends on the quantization, so a more refined or precise bin optimization algorithm may be able to improve on these results.

The files were compressed on a 2024 MacBook with Apple M4 Pro chipset (20 GPU cores, Metal 4) using llama-cpp-python 0.3.16; as we found runs on this machine to be identical, to confirm the theoretical correctness results (Theorem 1), we decoded the encoded files using the same device while adding uniformly random synthetic noise corresponding to the robustness guarantee. To test robustness to real non-determinism, we also decoded the encoded files with a 2023 MacBook with Apple M2 Pro chipset (16 GPU cores, Metal 3) using llama-cpp-python 0.2.88. All LLM inference used the same quantized model files and llama-cpp-python with GPU acceleration via Apple's Metal API.

5.2. Results

Compression ratio We measure the compression ratios (compressed file size divided by original input size) achieved by PMATIC with each model against those of

[9]Note that the optimization algorithm is only meant as a proof-of-concept for message-optimized parameter setting; we do not consider the specific details of the algorithm as a main contribution of this work, but give them for completeness.

arithmetic coding (AC) and gzip; the comparison with AC shows the cost of the added robustness. Although we optimized the bins for $\gamma = 8$, we compute PMATIC overhead assuming a 16-bit float (such as a bfloat16) is assigned for every value. The results are given in Figure 1 which shows the compression ratios on Llama 3.1 (other plots are included in Appendix A); even adding robustness against $\varepsilon = 0.3$ logit mismatches yields an average compression ratio considerably better than gzip, whereas the original formulation of PMATIC in (Adler & Tang, 2026) was only evaluated up to $\varepsilon = 0.02$.

For Llama and Qwen, updated PMATIC at 0.02 is usually better than the original formulation at 0.002, meaning over $10\times$ times robustness for no loss in compression performance.[10] Furthermore, the updated PMATIC formulation presented in this work often achieves comparable or even better compression ratio at $\varepsilon = 0.03$ as the original formulation of PMATIC with robustness achieves with $\varepsilon = 0.002$, meaning that there is a $15\times$ robustness improvement for no loss in compression performance. For Mistral, original formulation at 0.002 falls between updated PMATIC at 0.01 and 0.02, i.e. we have 5 to 10 times the robustness for the same compression level. See Table 3 in Appendix A for further details.

Mismatch Robustness To validate Theorem 1, we used PMATIC to decode all encoded files on the same device (no mismatch) with uniformly random synthetic noise in $[-\varepsilon, \varepsilon]$ (where ε is the robustness guarantee parameter) added to the logits. All files were decoded successfully.

We tested PMATIC on the real non-determinism between our two machines by encoding the dataset with one and decoding with the other, using PMATIC with $\varepsilon = 1.0, 0.3, 0.1, 0.03, 0.01, 0.003, 0.001, 0.0003, 0.0001$, representing a wide (and approximately equally logarithmically spaced) range of mismatch robustness. To facilitate a direct comparison with the original formulation of PMATIC, we also use $\varepsilon = 0.02, 0.002$, as these values of ε correspond to the experiments in (Adler & Tang, 2026).[11]

Figure 2 shows the fraction of files from the datasets which were correctly decoded at each setting. Differences in the accuracy of different datasets likely have to do with the length of the files and number of tokens. No files were decoded incorrectly for any $\varepsilon \geq 0.03$, and (as expected) standard arithmetic coding failed to correctly decode any files, and almost always failed within the first 5 tokens. To more precisely characterize how often decoding failed, we also measured the *per token* error rate, given by the number of decoding failures per *clean-context tokens*, i.e. those tokens for which all previous tokens were decoded correctly and hence have the correct model predictions (Figure 3). More detailed results can be found in Appendix A. .

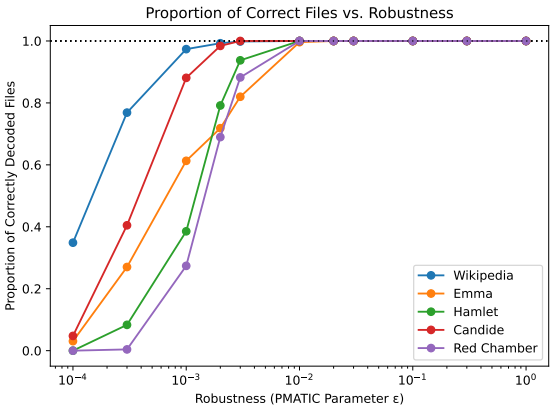

Figure 2. Proportion of correctly decoded files for PMATIC with varying robustness guarantee (ε) in real-mismatch testing. We note that many files from the Wikipedia dataset were considerably shorter than the sections of other datasets, leading to a higher success rate. Arithmetic coding did not successfully decode any files.

Remark 3. *We note that the $\|\boldsymbol{u} - \boldsymbol{v}\|_\infty \leq \varepsilon$ robustness guarantee should be viewed as a conservative worst-case certificate rather than a sharp failure threshold. In particular, even if $\|\boldsymbol{u} - \boldsymbol{v}\|_\infty > \varepsilon$, errors can still be very unlikely because they require correlated mismatch patterns and next-bit probabilities that fall in specific bands (generally near a bin boundary point but outside the associated danger-zone). Thus, Figure 2 does not reflect how often the mismatch violates the robustness guarantee, but rather the rate of the sequence of events needed to produce an error.*

Computational Efficiency While the computational efficiency bottleneck of model-driven compression is typically the model inference step at each token and our implementation was not optimized for performance, we wanted to ensure that PMATIC did not significantly affect the overall encoding or decoding speed. To validate this, we compared the encoding time dedicated to PMATIC computations to the time dedicated to model inference; LLM inference required an average of 24.92 milliseconds per token, while PMATIC operations required an average of 0.722 milliseconds per token ($\approx 2.9\%$ of the LLM inference time).

[10] Note that '$10\times$ robustness' here refers to the size of the worst-case mismatches it can theoretically tolerate, *not* empirical decrease in the per-token error rate. To see how improvements in theoretical tolerance translate into reductions in the error rate in practice, see Table 2 in Appendix A.

[11] In (Adler & Tang, 2026), a theoretical result representing a weaker version of Proposition 1 was used to correspond ε to danger-zones of uniform width $\delta = \varepsilon/2$. Thus, since (Adler & Tang, 2026) experimented with $\delta = 0.01, 0.001$, we correspondingly use $\varepsilon = 0.02, 0.002$ for comparison.

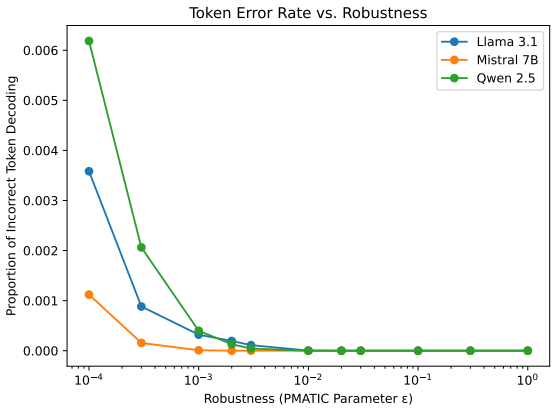

Figure 3. Fraction of tokens decoded wrongly by PMATIC for various robustness guarantee settings, computed over 'clean-context' tokens (for which *all previous tokens* were correct). Results average the fraction of wrong tokens over all experiments under a specific LLM; see Appendix A for additional plots and full results table by LLM and dataset. All tokens were correctly decoded for all LLMs for $\varepsilon \geq 0.02$.

6. Conclusion and Future Work

In this work, we extended the mismatch-robust PMATIC coding framework introduced in (Adler & Tang, 2026), incorporating a tight theoretical analysis of the bounded mismatch setting and flexible and data-dependent parameters. We proved the correctness of the updated framework and showed a theoretical analysis of its performance, validated it on synthetic data, and evaluated how its robustness performs on logit mismatches induced by a real instance of model non-determinism. However, numerous directions for future work on PMATIC remain:

Other Data Modalities So far, PMATIC has only been developed and tested on text data. However, model-driven compression has shown considerable potential for many different data modalities, such as images, scientific or industrial data, and even LLM training checkpoints, presenting an avenue for further development and application for PMATIC. This also opens the question of extending PMATIC to provide robustness for *lossy* compression, which is common for certain modalities, e.g. images.

Variable and Indexwise Binning PMATIC can also be extended by allowing the quantization bins and helper bit encoding probability to vary from bit to bit; as long as the encoder and decoder agree on the parameters to use for each bit, the correctness proof from Section 4.1 holds. This can be used to implement *indexwise binning*, where the bits are encoded using binning parameters determined by the position of the bit within their token's longform. This allows PMATIC to take advantage of the fact that the pool of possible tokens narrows with each additional bit (since tokens that do not match that bit get removed), resulting in considerable differences in the distribution of next-bit probabilities for each index.

Robustness level setting While this work leaves the value of the robustness guarantee ε as a design decision which will vary case-by-case according to the setting and the user's needs, a principled and reliable way to select ε would make PMATIC much more immediately applicable. One way of doing so is to empirically measure the next-bit mismatches between a variety of expected encoder and decoder machines, and then to use a statistical analysis of the mismatch sizes to select an ε that yields a sufficiently low rate of error, and further work on characterizing the statistical properties of mismatches is needed to determine how best to set ε. Additionally, other avenues for determining good values of ε to use, such as those based on mathematical analysis of how numerical errors propagate in the model, may be considered.

Stochastic Logit Error Models While the bounded logit error model offers a reasonable starting point and clean mathematical results, logit mismatch in real settings generally does not have a clearly-defined hard upper bound on the mismatch of individual logit scores. At the same time, the bounded logit error model in some ways assumes the worst case, i.e. that the bounded logit mismatches conspire to make the conditional next-bit probabilities as different as possible. Thus, it is natural to extend PMATIC to explicitly account for stochastic logit mismatch.

Impact Statement

This paper presents work focusing on a fundamental algorithmic component of data compression, designed for use with LLMs and other learned models, and does not directly affect model behavior or decision-making. The societal implications are thus indirect and aligned with those of data compression and machine learning more broadly. We do not feel that any specific impacts must be highlighted here.

References

Adler, A. and Tang, J. Synchronizing probabilities in model-driven lossless compression. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=nGzYkW4FSB.

Atil, B., Aykent, S., Chittams, A., Fu, L., Passonneau, R. J., Radcliffe, E., Rajagopal, G. R., Sloan, A., Tudrej, T., Ture, F., Wu, Z., Xu, L., and Baldwin, B. Nondeterminism of "deterministic" LLM system settings in hosted environments. In Akter, M., Chowdhury, T., Eger, S., Leiter, C., Opitz, J., and Çano, E. (eds.), *Proceedings of the 5th Workshop on Evaluation and Comparison of NLP Systems*, pp. 135–148, Mumbai, India, December 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.eval4nlp-1.12. URL https://aclanthology.org/2025.eval4nlp-1.12/.

Ballé, J., Johnston, N., and Minnen, D. Integer networks for data compression with latent-variable models. In *International Conference on Learning Representations*, 2018.

Bellard, F. Nncp v2: Lossless data compression with transformer. 2021. URL https://api.semanticscholar.org/CorpusID:231917764.

Chen, B., Wen, M., Shi, Y., Lin, D., Rajbahadur, G. K., and Jiang, Z. M. J. Towards training reproducible deep learning models. In *Proceedings of the 44th International Conference on Software Engineering*, ICSE '22, pp. 2202–2214. ACM, May 2022. doi: 10.1145/3510003.3510163. URL http://dx.doi.org/10.1145/3510003.3510163.

Chen, K., Zhang, P., Liu, H., Liu, J., Liu, Y., Huang, J., Wang, S., Yan, H., and Li, H. Large language models for lossless image compression: Next-pixel prediction in language space is all you need. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=FXBBy1caOX.

Cleary, J. and Witten, I. Data compression using adaptive coding and partial string matching. *IEEE Transactions on Communications*, 32(4):396–402, 1984. doi: 10.1109/TCOM.1984.1096090.

Cooper, A. F., Frankle, J., and De Sa, C. Non-determinism and the lawlessness of machine learning code. In *Proceedings of the 2022 Symposium on Computer Science and Law*, CSLAW '22, pp. 1–8. ACM, November 2022. doi: 10.1145/3511265.3550446. URL http://dx.doi.org/10.1145/3511265.3550446.

Deletang, G., Ruoss, A., Duquenne, P.-A., Catt, E., Genewein, T., Mattern, C., Grau-Moya, J., Wenliang, L. K., Aitchison, M., Orseau, L., Hutter, M., and Veness, J. Language modeling is compression. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=jznbgiynus.

Eryilmaz, B., Koras, O. A., Schlotterer, J., and Seifert, C. Investigating the Impact of Randomness on Reproducibility in Computer Vision: A Study on Applications in Civil Engineering and Medicine . In *2024 IEEE 6th International Conference on Cognitive Machine Intelligence (CogMI)*, pp. 265–274, Los Alamitos, CA, USA, October 2024. IEEE Computer Society. doi: 10.1109/CogMI62246.2024.00042. URL https://doi.ieeecomputersociety.org/10.1109/CogMI62246.2024.00042.

Grattafiori, A., Dubey, A., Jauhri, A., et al. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

He, H. and Thinking Machines Lab. Defeating nondeterminism in llm inference. *Thinking Machines Lab: Connectionism*, 2025. doi: 10.64434/tml.20250910. https://thinkingmachines.ai/blog/defeating-nondeterminism-in-llm-inference/.

Hu, C. and Tang, J. A model-driven lossless compression algorithm resistant to mismatch, 2026. URL https://arxiv.org/abs/2601.17684.

Kim, Y. and Belyaev, E. An efficient compression of deep neural network checkpoints based on prediction and context modeling, 2025. URL https://arxiv.org/abs/2506.12000.

Ma, Z., Zhu, H., He, Z., Lu, Y., and Song, F. Deep lossless compression algorithm based on arithmetic coding for power data. *Sensors*, 22(14), 2022. ISSN 1424-8220. doi: 10.3390/s22145331. URL https://www.mdpi.com/1424-8220/22/14/5331.

Mittu, F., Bu, Y., Gupta, A., Devireddy, A., Ozdarendeli, A. E., Singh, A., and Anumanchipalli, G. Finezip: Pushing the limits of large language models for practical lossless text compression. *arXiv preprint arXiv:2409.17141*, 2024.

NVIDIA Corporation. *NVIDIA cuBLAS Library Documentation*, 2025a. URL https://docs.nvidia.com/cuda/cublas/. Version: CUDA cuBLAS.

NVIDIA Corporation. *NVIDIA cuDNN Backend API: Odds and Ends (Determinism and Reproducibility)*, 2025b. URL https://docs.nvidia.com/

deeplearning/cudnn/backend/latest/developer/misc.html. cuDNN Developer Documentation.

Pasco, R. C. *Source coding algorithms for fast data compression*. PhD thesis, Stanford University CA, 1976.

Rissanen, J. J. Generalized kraft inequality and arithmetic coding. *IBM Journal of research and development*, 20 (3):198–203, 1976.

Schmidhuber, J. and Heil, S. Sequential neural text compression. *IEEE Transactions on Neural Networks*, 7(1): 142–146, 1996. doi: 10.1109/72.478398.

Semmelrock, H., Ross-Hellauer, T., Kopeinik, S., Theiler, D., Haberl, A., Thalmann, S., and Kowald, D. Reproducibility in machine-learning-based research: Overview, barriers, and drivers. 46(2), April 2025. ISSN 0738-4602. doi: 10.1002/aaai.70002. URL https://doi.org/10.1002/aaai.70002.

Shannon, C. E. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948. doi: 10.1002/j.1538-7305.1948.tb01338.x.

Witten, I. H., Neal, R. M., and Cleary, J. G. Arithmetic coding for data compression. *Commun. ACM*, 30:520–540, 1987. URL https://api.semanticscholar.org/CorpusID:3343393.

A. Full Results Tables and Plots

In this appendix, we give our full results tables and plots for our results on using PMATIC with three different LLMs (Llama 3.1 8B Q4_K_S (4-bit quantized), Mistral 7B v0.1 (3-bit quantized), and Qwen 2.5 Instruct 7B (3-bit quantized)) tested on our five datasets. Figure 4 shows the compression ratios on Mistral and Qwen. Table 1 shows the compression ratio results for all experiments and Table 2 shows the token error rates for all experiments. Table 3 shows comparison to the previous version of PMATIC in (Adler & Tang, 2026) where bin sizes were uniform.

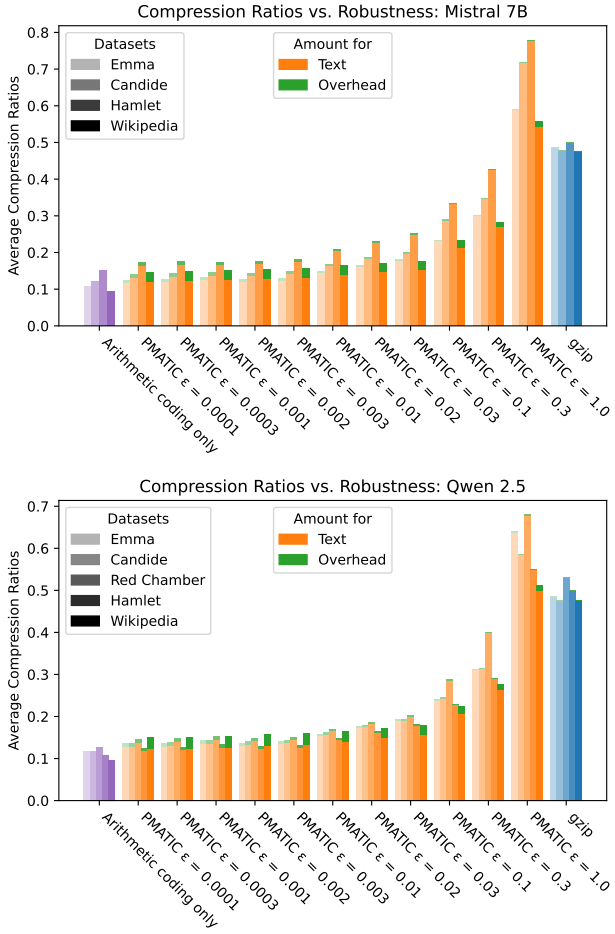

Figure 4. Similar to Figure 1, this plot shows compression ratios for PMATIC on Mistral 7B and Qwen 2.5. (compressed file size divided by original file size) with different robustness, with comparison to arithmetic coding and gzip.

LLM Tested	**Method**	**Robustness**	*Candide* (French)	*Emma*	*Hamlet*	*Dream of the Red Chamber* (Chinese)	Wikipedia
Llama 3.1	Arithmetic Coding		0.1047	0.0600	0.0873	–	0.0815
	PMATIC	$\varepsilon = 0.0001$	0.1155	0.0677	0.0961	–	0.1085
		$\varepsilon = 0.0003$	0.1174	0.0684	0.0979	–	0.1094
		$\varepsilon = 0.001$	0.1220	0.0717	0.1028	–	0.1122
		$\varepsilon = 0.002$	0.1190	0.0731	0.1006	–	0.1157
		$\varepsilon = 0.003$	0.1232	0.0709	0.1038	–	0.1182
		$\varepsilon = 0.01$	0.1453	0.0801	0.1197	–	0.1244
		$\varepsilon = 0.02$	0.1616	0.0893	0.1352	–	0.1349
		$\varepsilon = 0.03$	0.1775	0.0977	0.1485	–	0.1387
		$\varepsilon = 0.1$	0.2246	0.1291	0.1956	–	0.1822
		$\varepsilon = 0.3$	0.2934	0.1631	0.2499	–	0.2341
		$\varepsilon = 1.0$	0.5797	0.2932	0.4915	–	0.4475
Mistral 7B	Arithmetic Coding		0.1215	0.1073	0.1512	–	0.0944
	PMATIC	$\varepsilon = 0.0001$	0.1321	0.1170	0.1629	–	0.1201
		$\varepsilon = 0.0003$	0.1347	0.1190	0.1666	–	0.1211
		$\varepsilon = 0.001$	0.1382	0.1247	0.1649	–	0.1241
		$\varepsilon = 0.002$	0.1368	0.1203	0.1690	–	0.1280
		$\varepsilon = 0.003$	0.1422	0.1237	0.1747	–	0.1309
		$\varepsilon = 0.01$	0.1646	0.1450	0.2037	–	0.1407
		$\varepsilon = 0.02$	0.1820	0.1614	0.2269	–	0.1479
		$\varepsilon = 0.03$	0.1982	0.1770	0.2495	–	0.1541
		$\varepsilon = 0.1$	0.2870	0.2317	0.3335	–	0.2144
		$\varepsilon = 0.3$	0.3472	0.2996	0.4246	–	0.2710
		$\varepsilon = 1.0$	0.7157	0.5885	0.7769	–	0.5409
Qwen 2.5	Arithmetic Coding		0.1174	0.1175	0.1093	0.1276	0.0959
	PMATIC	$\varepsilon = 0.0001$	0.1282	0.1276	0.1188	0.1380	0.1236
		$\varepsilon = 0.0003$	0.1304	0.1295	0.1205	0.1401	0.1245
		$\varepsilon = 0.001$	0.1354	0.1359	0.1269	0.1459	0.1273
		$\varepsilon = 0.002$	0.1323	0.1312	0.1223	0.1423	0.1309
		$\varepsilon = 0.003$	0.1360	0.1350	0.1263	0.1451	0.1332
		$\varepsilon = 0.01$	0.1576	0.1549	0.1447	0.1658	0.1411
		$\varepsilon = 0.02$	0.1745	0.1727	0.1613	0.1827	0.1492
		$\varepsilon = 0.03$	0.1910	0.1893	0.1771	0.1991	0.1560
		$\varepsilon = 0.1$	0.2430	0.2386	0.2263	0.2850	0.2058
		$\varepsilon = 0.3$	0.3133	0.3120	0.2905	0.3993	0.2640
		$\varepsilon = 1.0$	0.5852	0.6372	0.5498	0.6768	0.4989

Table 1. Average compression ratio over different LLMs and datasets. Compression ratio for an individual file is $\frac{\text{size of compressed file}}{\text{size of raw file}}$, following the convention of (Deletang et al., 2024); the reported value is the average of the compression ratios of the individual file sizes. Dashes ('–') indicate that the experiment was omitted because the LLM is not supported in the given language.

LLM Tested	**Robustness**	*Candide* (French)	*Emma*	*Hamlet*	*Dream of the Red Chamber* (Chinese)	Wikipedia
Llama 3.1	$\varepsilon = 0.0001$	0.002786	0.007758	0.004302	–	0.003183
	$\varepsilon = 0.0003$	0.000637	0.005448	0.003354	–	0.000502
	$\varepsilon = 0.001$	0.000048	0.002797	0.000997	–	0.000050
	$\varepsilon = 0.002$	·	0.001324	0.000111	–	0.000012
	$\varepsilon = 0.003$	·	0.000610	0.000021	–	0.000002
	$\varepsilon = 0.01$	·	0.000009	·	–	·
	$\varepsilon = 0.02$	·	·	·	–	·
	$\varepsilon = 0.03$	·	·	·	–	·
	$\varepsilon = 0.1$	·	·	·	–	·
	$\varepsilon = 0.3$	·	·	·	–	·
	$\varepsilon = 1.0$	·	·	·	–	·
Mistral 7B	$\varepsilon = 0.0001$	0.001172	0.001503	0.002487	–	0.000975
	$\varepsilon = 0.0003$	0.000180	0.000234	0.000864	–	0.000095
	$\varepsilon = 0.001$	·	0.000017	0.000057	–	0.000003
	$\varepsilon = 0.002$	·	·	·	–	·
	$\varepsilon = 0.003$	·	·	·	–	·
	$\varepsilon = 0.01$	·	·	·	–	·
	$\varepsilon = 0.02$	·	·	·	–	·
	$\varepsilon = 0.03$	·	·	·	–	·
	$\varepsilon = 0.1$	·	·	·	–	·
	$\varepsilon = 0.3$	·	·	·	–	·
	$\varepsilon = 1.0$	·	·	·	–	·
Qwen 2.5	$\varepsilon = 0.0001$	0.004306	0.005995	0.007452	0.007349	0.005784
	$\varepsilon = 0.0003$	0.001519	0.001908	0.004333	0.004196	0.001321
	$\varepsilon = 0.001$	0.000222	0.000146	0.001473	0.000971	0.000087
	$\varepsilon = 0.002$	0.000032	0.000019	0.000416	0.000274	0.000025
	$\varepsilon = 0.003$	·	0.000005	0.000111	0.000092	0.000006
	$\varepsilon = 0.01$	·	·	·	·	·
	$\varepsilon = 0.02$	·	·	·	·	·
	$\varepsilon = 0.03$	·	·	·	·	·
	$\varepsilon = 0.1$	·	·	·	·	·
	$\varepsilon = 0.3$	·	·	·	·	·
	$\varepsilon = 1.0$	·	·	·	·	·

Table 2. Error rate of decoding a token under conditions where the context is correct (no previous tokens were incorrectly decoded). Dots ('·') indicate that no tokens were decoded incorrectly, while dashes ('–') indicate that the given LLM was not tested on the given dataset due to lack of language support (as in Table 1).

LLM Tested	**Robustness**	**PMATIC Version**	*Candide* (French)	*Emma*	*Hamlet*	*Dream of the Red Chamber* (Chinese)	Wikipedia
Llama 3.1	$\varepsilon = 0.002$	V1 (uniform)	0.1683	0.1099	0.1514	–	0.1330
		V2 (optimized)	0.1181	0.0730	0.1005	–	0.0980
	$\varepsilon = 0.02$	V1 (uniform)	0.2971	0.2113	0.2772	–	0.2085
		V2 (optimized)	0.1610	0.0892	0.1351	–	0.1236
	$\varepsilon = 0.03$	V1 (uniform)	–	–	–	–	–
		V2 (optimized)	0.1769	0.0976	0.1483	–	0.1347
Mistral 7B	$\varepsilon = 0.002$	V1 (uniform)	–	0.1595	0.2167	–	0.1198
		V2 (optimized)	0.1357	0.1203	0.1690	–	0.1039
	$\varepsilon = 0.02$	V1 (uniform)	–	0.2610	0.3447	–	0.2106
		V2 (optimized)	0.1816	0.1614	0.2269	–	0.1299
	$\varepsilon = 0.03$	V1 (uniform)	–	–	–	–	–
		V2 (optimized)	0.1978	0.1771	0.2494	–	0.1416
Qwen 2.5	$\varepsilon = 0.002$	V1 (uniform)	0.1831	0.1738	0.1755	0.1879	0.1315
		V2 (optimized)	0.1311	0.1311	0.1223	0.1423	0.1089
	$\varepsilon = 0.02$	V1 (uniform)	0.3171	0.2825	0.3067	0.3073	0.2297
		V2 (optimized)	0.1741	0.1727	0.1613	0.1828	0.1349
	$\varepsilon = 0.03$	V1 (uniform)	–	–	–	–	–
		V2 (optimized)	0.1908	0.1893	0.1771	0.1991	0.1471

Table 3. Comparison of the original formulation of PMATIC presented in (Adler & Tang, 2026) (denoted V1) with the updated formulation presented in this work (denoted V2). Compression ratios here are computed by total compressed size of the dataset divided by total original size, in order to match the computation done in (Adler & Tang, 2026). Dashes ('–') mean the experiment was omitted either because the LLM did not support the language of the dataset or because it was not included in (Adler & Tang, 2026). We included $\varepsilon = 0.03$ for the optimized PMATIC in this work so that we can capture where there is 10 to 15 times improvement in robustness at a fixed compression ratio level.

B. On Setting the Bin Boundaries

In this appendix, we describe in detail how the bin boundaries $0 = a_0 \leq a_1 \leq \cdots \leq a_{m-1} \leq a_m = 1$ (which we denote by the vector $\boldsymbol{a}$) used in our main experiments were chosen. In line with the use of message-optimized parameters, each file to be encoded is given its own bin boundaries, bin centers, and helper-bit encoding probability.

Remark 4. *We do not view the specifics of this bin-setting procedure as part of the primary contribution of this work, but as a proof of concept for message-optimized parameter setting. While we believe that the algorithms described here can likely be considerably improved, we include these details for completeness.*

We use the theoretical overhead cost function given in (22) as the objective function:

$$f(\boldsymbol{a}) = \sum_{j=1}^{n\ell} D_{\mathrm{KL}}(p_j^* \| \hat{p}_j) + n\ell \cdot h(\rho) + 2\gamma m$$

Fundamentally, minimizing this requires minimizing three different terms:

1. The overall quantization loss $\sum_{j=1}^{n\ell} D_{\mathrm{KL}}(p_j^* \| \hat{p}_j)$ (binary KL divergence between encoder-predicted probabilities and quantized probabilities); this favors relatively small bins (and hence more of them) so that all values within them are close to the bin representative (which is optimally placed at the average of the probability values in the bin).

2. The helper-bit overhead $n\ell \cdot h(\rho)$, representing the cost of sending the helper bits, where ρ is (optimally) set at the empirical fraction of helper bits which are equal to `1` (i.e. the number of next-bit probability values that fall inside danger-zones). This favors bin boundary points which minimize ρ (which we assume to be $< 1/2$ since danger-zones are generally smaller than bin interiors[12]) by minimizing the number of next-bit probability values within the danger-zones, which also favors having fewer bins in general (and hence fewer danger-zones).

3. The penalty term $2\gamma m$, representing the cost of attaching the parameter values to the message so the decoder knows what to use. This favors having fewer bins. Furthermore, while the first two terms scale with the number of tokens n, this term scales with the number of bins m; this makes this term relatively less important as the message length grows.

As stated in Remark 4, solving this optimization problem in a full, principled, and computationally-efficient way is itself difficult (since the problem is non-convex and fundamentally discrete) and outside the scope of this work. Instead, we use a simplified framework and greedy bin boundary selection process; we also use approximation and discretization in order to make the process more computationally efficient, particularly when optimizing bin boundary points for messages with many tokens.

B.1. The Overall Optimization Problem

For the overall optimization problem, the input is the full set of conditional next-bit probability values $p_1^*, \ldots, p_{n\ell}^*$ (each of n tokens generates ℓ conditional next-bit probability values), and the objective is to select:

- $m-1$ bin boundary points $0 < a_1 \leq \cdots \leq a_{m-1} < 1$;
- m bin representative values $r_1, \ldots, r_m$;
- and a helper-bit encoding probability $\rho \in [0, 1]$.

We note that the number of parameters to use is $2m$ and that m is itself a design decision, and that once $\boldsymbol{a} = (a_1, \ldots, a_{m-1})$ is set, the optimal values for $r_1, \ldots, r_m$ and ρ are easy to set as well: the optimal r_k is the average of the probabilities p_j^* which fall in bin interior I_k^ε (i.e. the average of the next-bit probability values that will end up quantized to r_k), while the optimal ρ is the empirical fraction of next-bit probabilities that fall into dangerzones.

To formally define this, we have:

$$r_k = \frac{\sum_{j:p_j^* \in I_k^\varepsilon} p_j^*}{|\{j : p_j^* \in I_k^\varepsilon\}|} \quad \text{where } I_k = [a_{k-1}, a_k) \tag{23}$$

Let $J^\varepsilon = \bigcup_k J_k^\epsilon$ be the union of all danger-zones, where J_k^ε is the danger-zone for robustness ε around bin boundary a_k (see (10)). Thus, the helper bit b'_j is defined by whether the next-bit probability p_j^* falls in J^ε:

$$b'_j = \begin{cases} 0 & \text{if } p_j^* \notin J^\varepsilon \\ 1 & \text{if } p_j^* \in J^\varepsilon \end{cases} \tag{24}$$

The empirical fraction of helper bits which are `1` is then

$$\rho = \frac{\sum_j b'_j}{n\ell} = \frac{|\{j : p_j^* \in J^\varepsilon\}|}{n\ell}\,. \tag{25}$$

Note that ρ is both the fraction of helper bits which are `1` and (therefore) the optimal probability value used to encode the helper bits with arithmetic encoding.[13]

[12] This assumption is extremely solid for any reasonable distribution of next-bit probability values; additionally, if the danger-zones contain more than half the next-bit probability values, the danger-zones and bin interiors should be swapped.

[13] Being able to use the exact helper bit fraction ρ as the encoding probability requires message-based encoding; using fully pre-set parameters requires the encoder and decoder to guess an appropriate value instead.

However, for simplicity, our overall optimization problem will drop the quantization error terms corresponding to next-bit probabilities p_j^* that fall into danger-zones.[14] Thus, we are really minimizing an approximate theoretical distortion function which drops those terms:

$$\hat{f}(\boldsymbol{a}) = \sum_{j:p_j^* \notin J^\varepsilon} D_{\mathrm{KL}}(p_j^* \| \hat{p}_j) + n\ell\, h(\rho) + 2\gamma m \tag{26}$$

$$= \sum_{k=1}^{m} \sum_{j:p_j^* \in I_k^\varepsilon} D_{\mathrm{KL}}(p_j^* \| r_k) + n\ell\, h(\rho) + 2\gamma m \tag{27}$$

B.2. Recursive Greedy Optimization

While the ideal case is to globally minimize (22) (or approximate by minimizing (26)) the problem is not straightforward since these are not convex functions. Since our aim is to provide a proof-of-concept for message-optimized quantization and to show how it can work within the broader PMATIC framework, we instead use a heuristic greedy approach: given a boundary set $\boldsymbol{a}$ and a target bin $I_k = [a_{k-1}, a_k]$, we want to place a new boundary a^{new} in I_k which minimizes the approximate objective function (26); if any such boundary point would increase (26) (typically due to the constant penalty for more bins), then we do not add that boundary point at all and return `null`.

The value a^{new} (if not `null`) must also respect the constraints $|\operatorname{logodds}(a^{\text{new}}) - \operatorname{logodds}(a_{k-1})| \geq 8\varepsilon$ and $|\operatorname{logodds}(a^{\text{new}}) - \operatorname{logodds}(a_k)| \geq 8\varepsilon$ to avoid decoding ambiguities. We denote this range as

$$I_k^{\text{new}} := [\operatorname{logodds}^{-1}(\operatorname{logodds}(a_{k-1}) + 8\varepsilon), \operatorname{logodds}^{-1}(\operatorname{logodds}(a_k) - 8\varepsilon)]\,. \tag{28}$$

We then define the new boundary set (with a^{new} added) as

$$\boldsymbol{a}^{\text{new}} := \begin{cases} \boldsymbol{a} \cup \{a^{\text{new}}\} & \text{if } a^{\text{new}} \neq \texttt{null} \\ \boldsymbol{a} & \text{if } a^{\text{new}} = \texttt{null} \end{cases} \tag{29}$$

The single-interval optimization problem is then:

$$\begin{aligned} &\text{minimize } \hat{f}(\boldsymbol{a}^{\text{new}}) - \hat{f}(\boldsymbol{a}) \\ &\text{subject to } a^{\text{new}} \in I_k^{\text{new}} \cup \{\texttt{null}\} \end{aligned} \tag{30}$$

Note that I_k^{new} can be empty, which happens when

$$\operatorname{logodds}(a_{k-1}) + 8\varepsilon > \operatorname{logodds}(a_k) - 8\varepsilon \tag{31}$$

In this case, a^{new} must be `null`.

Additionally, note that the optimal value is always $\hat{f}(\boldsymbol{a}^{\text{new}}) - \hat{f}(\boldsymbol{a}) \leq 0$ since you can always set $a^{\text{new}} = \texttt{null}$, which produces $\boldsymbol{a}^{\text{new}} = \boldsymbol{a}$ and hence $\hat{f}(\boldsymbol{a}^{\text{new}}) - \hat{f}(\boldsymbol{a}) = 0$.

We denote by $\mathsf{Alg}(\boldsymbol{a}, I)$ a generic single-interval optimization algorithm which takes in an existing quantization $\boldsymbol{a}$ and a target interval I, and returns a^{new}, which either is: (i) a new boundary point in I which divides into left and right parts I_-, I_+, or (ii) `null`, indicating that Alg cannot find such a boundary point. Alg can then be used in a recursive optimizer Rec which, given existing quantization $\boldsymbol{a}$ and interval I, does the following:

Algorithm 1 Rec (Recursive Greedy Bin Division)

Require: Current quantization $\boldsymbol{a}$, target interval I
Ensure: Updated quantization $\boldsymbol{a}$
  $a^{\text{new}} \leftarrow \mathsf{Alg}(\boldsymbol{a}, I)$
  if $a^{\text{new}} = \texttt{null}$ **then**
    return $\boldsymbol{a}$
  else
    $\boldsymbol{a} \leftarrow \boldsymbol{a} \cup \{a^{\text{new}}\}$
    $\boldsymbol{a} \leftarrow \mathsf{Rec}(\boldsymbol{a}, I_-)$
    $\boldsymbol{a} \leftarrow \mathsf{Rec}(\boldsymbol{a}, I_+)$
    return $\boldsymbol{a}$
  end if

Then, $\mathsf{Rec}(\{\,\}, [0,1])$ (i.e. the only bin boundaries are the defaults at 0 and 1 and interval $I = [0,1]$ is the target interval) will try to greedily (according to Alg) add a new boundary in its target interval; if it finds a new boundary to add, it will recurse to the left and right sub-intervals in that order, while it returns if it does not find a new boundary.

B.3. Single-Interval Optimization in More Depth

We now consider the single-interval optimization problem (30) in more depth, by splitting up $\hat{f}(\boldsymbol{a}^{\text{new}}) - \hat{f}(\boldsymbol{a})$ into the quantization loss, helper-bit entropy, and penalty terms.

Since the quantization loss error can be decomposed into a sum of terms contributed by each bin, we can ignore all other bins (which cancel when subtracting $\hat{f}(\boldsymbol{a}^{\text{new}}) - \hat{f}(\boldsymbol{a})$); however, we do need to know how many points are already in danger-zones (across all danger-zones) in order to know how the binary entropy changes when we add more. Finally, regardless of how many bins already exist, we know that adding one more imposes a penalty of 2γ.

The individual-bin optimization problem thus takes in the following parameters:

- the robustness parameter ε and bin count penalty γ;
- the bin $I_k = [a_{k-1}, a_k]$ (with bin interior I_k^ε);

[14] Due to the helper-bit entropy, it is generally optimal to strongly reduce the number of probabilities that fall into danger-zones. This both reduces the number of such probabilities, and it makes bin interiors (the other source of quantization loss) generally much wider than danger-zones; thus the quantization error from danger-zones will both include far fewer terms and have far less average distortion per term. It probably would not be difficult to include these terms but, as this is a proof-of-concept and they are expected to be relatively negligible, we ignore them.

- the next bit probability values p_j^* within I_k^ε (i.e. ignore all next-bit probability values outside this range);
- the current helper-bit fraction ρ (see (25)) and the total number of bits $n\ell$.

For any $a^{\text{new}} \neq \texttt{null}$, let $J^\varepsilon(a^{\text{new}})$ denote the new danger-zone around a^{new} ($\pm 2\varepsilon$ in log-odds space), and let I_-, I_+ be the new bins on the left and right with bin-interiors denoted $I_-^\varepsilon, I_+^\varepsilon$ and bin representatives (which are the averages of the p_j^* values in their interiors)[15] respectively.

Then (for $a^{\text{new}} \neq \texttt{null}$) we can cancel terms and break the objective function into its three parts:

$$\begin{aligned}\hat{f}(\boldsymbol{a}^{\text{new}}) - \hat{f}(\boldsymbol{a}) = \\ \sum_{k=1}^{m} \sum_{j:p_j^* \in I_k^\varepsilon} D^{\text{new}}(p_j^*) - \sum_{k=1}^{m} \sum_{j:p_j^* \in I_k^\varepsilon} D_{\text{KL}}(p_j^* \| r_k) \\ + n\ell \left(h\left(\rho + \frac{|\{j : p_j^* \in J^\varepsilon(a^{\text{new}})\}|}{n\ell} \right) - h(\rho) \right) \\ + 2\gamma\end{aligned} \tag{32}$$

where $D^{\text{new}}(p_j^*)$ denotes the distortion of $p_j^* \in I_k^\varepsilon$ under the new quantization, formally defined as

$$D^{\text{new}}(p_j^*) = \begin{cases} D_{\text{KL}}(p_j^* \| r_-^{\text{new}}) & \text{if } p_j^* \in I_-^\varepsilon \\ D_{\text{KL}}(p_j^* \| r_+^{\text{new}}) & \text{if } p_j^* \in I_+^\varepsilon \\ 0 & \text{if } p_j^* \in J^\varepsilon(a^{\text{new}}) \end{cases} \tag{33}$$

Note that the exact value of a^{new} does not affect $\hat{f}(\boldsymbol{a}^{\text{new}}) - \hat{f}(\boldsymbol{a})$; rather, what matters is which values of p_j^* fall into the danger-zone $J^\varepsilon(a^{\text{new}})$ around a^{new} (and all smaller p_j^* fall into I_-^ε, while all larger p_j^* fall into I_+^ε).[16] Thus, the optimal set of p_j^* to include in $J^\varepsilon(a^{\text{new}})$ can be computed by sweeping across the sorted values of p_j^*, adding or dropping values as they fall in or out of $J^\varepsilon(a^{\text{new}})$.

We refer to this algorithm as $\mathsf{Alg}_{\text{full}}$. However, we note that $\mathsf{Alg}_{\text{full}}$ requires the exact values of all the next-bit probability values; to make it more efficient, we discretize the problem by partitioning the log-odds space into intervals of width ε/s (with boundaries at integer multiples of ε/s), where $s \geq 1$ is an integer we call the *subdivision ratio*. For each such interval in the partition (which are not, themselves, quantization bins) we note the number of next-bit probability values p_j^* that fall in it, as well as the average of the next-bit probability values in it; we then approximate each p_j^* as the this average. This allows us to use these counts (like a histogram) and averages rather than the full list of next-bit probability values in order to determine the (approximate) value of a new bin boundary. In order to ensure that we know exactly which next-bit probability values fall into a new boundary's danger-zone, we require that a^{new} (if not $\texttt{null}$) be such that $\text{logodds}(a^{\text{new}})$ is an integer multiple of ε/s, so that the danger-zones (and hence the bin interiors as well) exactly cover a set of partition intervals. We call this modified boundary selection algorithm $\mathsf{Alg}_{\text{discrete}}$.

We then use $\mathsf{Alg}_{\text{discrete}}$ with the recursive greedy bin division algorithm Rec to get the quantization $\mathsf{Rec}(\{\}, [0,1])$ used in the experiments.

The subdivision ratio is a hyperparameter of the discrete bin optimization algorithm, whose values in our experiments (across all LLMs and all datasets) is given in Table 4. We used a larger subdivision ratio for larger ε in order to balance two competing concerns: if ε/s is too big, the set of allowable boundaries is too sparse and we may get a very inefficient quantization; however, if ε/s is too small, the optimization algorithm may take a long time since its input size now depends on the range of non-empty partition intervals.

Robustness ε	**Subdivision ratio** s
1.0	32
0.3	24
0.1	16
0.03	12
0.02	12
0.01	8
0.003	6
0.002	6
0.001	4
0.0003	3
0.0001	2

Table 4. Subdivision ratio used to compute the message-optimized quantizations for each robustness level.

Remark 5. *The subdivision ratios in Table 4 were chosen to broadly satisfy the requirement of making the partition fine enough to be a good approximation while also sparse enough to allow fast computation, but were not fine-tuned or further optimized.*

[15] If there are no p_j^* in the interior, the bin is an empty bin and its representative is arbitrarily defined (it does not affect anything because no value will be quantized to it). This generally does not occur as empty bins are very inefficient.

[16] Thus, this is really a *discrete* optimization problem.