CAN LLMZIP ENCODE WITHOUT AN LLM?

Anonymous authors
Paper under double-blind review

ABSTRACT

Predictive compression methods such as LLMZip combine large language models (LLMs) with arithmetic coding to achieve strong text compression, but require costly LLM inference at both the encoder and decoder. We introduce text compression schemes that eliminate LLM inference at the encoder while retaining compression rates close to those of LLMZip. Our approach interleaves text into a matrix and compresses each column using syndrome source coding with error-correcting codes. Using the Slepian–Wolf (SW) theorem for general correlated sources, we establish that the scheme asymptotically matches LLMZip's compression rate as the block length grows. For finite lengths, we develop a construction using low-density parity-check (LDPC) codes over $\mathrm{GF}(263)$, with a decoder that combines belief propagation with sequential LLM predictions across parallel text streams. On text8, using the byte-level BLT-7B model, our scheme achieves 1.2 and 1.73 bits per character (bpc) at block lengths of 4096 and 256 respectively compared to the 0.972 bpc of LLMZip. We further improve finite-length performance by introducing a weak language model at the encoder. Its predictions enable adaptive rate selection and optimized assignment of tokens to variable nodes in the LDPC Tanner graph, and the assignment requires no signaling. The two methods are complementary and together offer reductions of 0.09 and 0.264 bpc at block lengths of 4096 and 256 respectively, with feedback to account for rare predicted errors in rate to be truly lossless. Using a LoRA fine-tuned BLT-7B model for images as a pixel model at the decoder, we show that the scheme can also compress images losslessly. These results demonstrate that strong LLM-based compression can be achieved with substantially reduced encoder computation.

1 INTRODUCTION

We consider lossless compression of time series (exemplified by text) and images when the encoder is computationally constrained while the decoder can afford substantially higher computational complexity. In the absence of computational constraints, the best compression ratios for text come from pairing a language model with arithmetic coding (Valmeekam et al., 2023; Delétang et al., 2024), where the model predicts each token from its context and the arithmetic coder converts the predictions to an encoded bit sequence achieving 0.972 bits/character on our corpus. The major drawback of this scheme is that it requires running inference using a large language model at the encoder, the computational cost of which is often prohibitive. In this paper, we address the question: is the huge inference cost at the encoder avoidable? We answer it affirmatively by showing that asymptotically (in the length), the inference cost can be eliminated at the encoder without loss in compression under mild conditions on the source distribution. For finite lengths, there is a small penalty, which can be further reduced by using a weak language model at the encoder. Our encoder is universal and it does not depend on the decoder's model. Any model with the same tokenizer and an average log-loss below the chosen rate can decode successfully with high probability.

Computationally lighter alternatives to LLMZip exist, but each scheme has limitations. General-purpose compressors are cheap but their compression performance leaves a lot of room for improvement: on our data `gzip -9` requires 2.927 bits/character and `xz -9e` 2.135 bits/character. Context mixing, the strongest classical approach, closes part of the gap: `paq8h` compresses text8 to 1.40 bits/character (Mahoney, 2006), using 854 MB of memory at the encoder and an ensemble of models evaluated for every bit. Because it is symmetric, the decoder must run the same model, so the rate is limited by the model the encoder can afford, however much compute the decoder can

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.

Table 1: Encoder compute and compression rate on text8, with BLT-7B at the decoder and block length $n = 4096$. Rates (ii) to (v) are empirical lossless rates (Table 2); operations are per source character. SW: Slepian–Wolf. *BLT-7B with context $M_e = 16$ (Section 5).

	scheme	encoder computes	rate [bits/character]	ops/char
(i)	LLM + arithmetic coding	full LLM (M=128)	0.972	5×10^{11}
(ii)	SW-LDPC syndrome (ours)	**no model**	1.20	**4**
(iii)	SW with adaptive rate (ours)	weak predictor*	1.163	10^{11}
(iv)	SW with adaptive placement (ours)	weak predictor*	1.14	10^{11}
(v)	SW with both (ours)	weak predictor*	**1.110**	10^{11}
	`xz -9e` / `zstd -19` / `gzip -9`	no model	2.135 / 2.129 / 2.927	10^2–10^3
	context mixing (`paq8h`, quoted)	no pretrained model	1.40	10^5

afford. The same limitation applies to LLMZip with a smaller language model at the encoder: arithmetic coding requires bit-identical probabilities at both ends, so a weaker encoder model means a weaker decoder model and a worse rate, and each token still costs a forward pass. None of these options allows a well-provisioned decoder to improve on what a constrained encoder can compute.

We propose a lossless compression scheme that rearranges a tokenized sequence into multiple streams and applies syndrome source coding across them. In its base form, the encoder evaluates no model; the decoder uses a language model to predict each stream's next token from its recovered context and the syndrome to recover the tokens jointly. We measure the finite-length cost of removing the model from the encoder and what a deliberately weak encoder-side predictor can recover. Table 1 summarizes the resulting tradeoff between encoder computation and compression rate.

Contributions. We introduce the interleaved scheme for encoding a single source and we build the scheme at scale, pairing a 7B byte-level language model with an LDPC coset code over $\mathrm{GF}(263)$ decoded by belief propagation (Section 3). To our knowledge this is the first lossless compressor with a language model as decoder-only side information and no model at the encoder, which spends two (finite-field) multiply-accumulates per coded symbol.

Second, we measure the finite-length cost at five block lengths from 256 to 16384 (Section 4). At $n = 4096$, every codeword in the text8 corpus decodes at 1.20 bits/character, so the whole corpus is recovered losslessly.

Third, we quantify the benefits a deliberately weak encoder-side predictor can provide. It helps choose each column's rate and it can optimize symbol placement in the Tanner graph of the LDPC code (Section 5). Such placement requires no signaling, since both terminals derive the same permutation. Together, with one round of feedback for the rate, the two lower the empirical lossless rate on text8 by 0.083 to 0.264 bits/character, more than either alone.

Fourth, we apply the scheme to two further text corpora and to Kodak images, retraining only the rate predictor and, for images, adding a LoRA adapter to the decoder's model (Section 6). The gain from adaptive rate on top of adaptive placement grows with the excess log loss of the hardest codeword: 2 to 5% on every source at $n = 4096$, and at $n = 256$, 7 to 9% on text against 32% on pixels.

Related work. Most of the related work is in the literature on distributed source coding of *multiple* correlated sources unlike the encoding of a single source considered here. Syndrome-based Slepian–Wolf (SW) coding was made practical by DISCUS (Pradhan & Ramchandran, 2003), with binary (Liveris et al., 2002) and $\mathrm{GF}(q)$ (Davey & MacKay, 1998) LDPC realizations; Csiszár (1982) showed that linear encoders achieve the bound universally. These works assume memoryless correlation. For correlation with memory, one can decode jointly through the factor graph of the correlation model (Garcia-Frias & Zhao, 2001). Bhattad and Narayanan use a decision feedback scheme for encoding two correlated sources with Markov memory in the correlation (Bhattad & Narayanan, 2006). The interleaved structure they use is identical to the one proposed here; however, they consider distributed compression unlike the single source compression here. Language-model compressors for text (Valmeekam et al., 2023; Delétang et al., 2024) and images (Tian et al., 2026; Chen et al., 2024) attain rates close to the conditional entropy but run the model both at the encoder and decoder; we treat them as the reference point and measure what removing the encoder-side model

costs. Rate-adaptive distributed source coding (Varodayan et al., 2006) sets the rate by decoder feedback, whereas our encoder predicts each codeword's rate before sending and uses feedback only to repair the rare miss (Appendix F); neural distributed coding (Mital et al., 2022) trains an encoder-side network, whereas our encoder runs none.

Sections 2 and 3 present the setup and the system; Sections 4 to 6 report the fixed-rate results, the two encoder-side decisions, and considers extension to other sources.

2 SETUP

The source (text) consists of N_c characters each represented by a byte and tokenized into $\mathbf{x} = (x_1, \ldots, x_{N_T})$, a realization of the random vector $\mathbf{X}$. We use a byte-level tokenizer and, hence, $N_c = N_T$. The model vocabulary is $\mathcal{X} = \{0, 1, \ldots, D-1\}$ with $D = 260$: the 256 byte values and four special tokens. Special tokens are not counted as source bytes. The encoder $\mathcal{E}$ maps $\mathbf{x}$ to a variable-length bit sequence $\mathbf{b} \in \{0,1\}^{K(\mathbf{x})}$, and the decoder reconstructs $\hat{\mathbf{x}} \in \mathcal{X}^{N_T}$. We define the document error probability as $P_e := \Pr(\hat{\mathbf{X}} \neq \mathbf{X})$, with probability taken over the source and any randomization in the coding protocol. Our reliability target is $P_e \leq \epsilon$ for a small $\epsilon > 0$. Error detection and feedback can further decrease ϵ to render the scheme practical even for applications that require stringent quality of service. At the reported error-free operating points, every evaluated document was recovered exactly.

We measure compression performance for ASCII text by the expected rate $\rho := \mathbb{E}[K]/N_c$ in bits/character. For the full protocol, K should include all communicated bits: the compressed payload, initial contexts, error-detection information, signaling, feedback, and repair transmissions. In this paper, we focus only on the rate over the coded tokens and ignore the signaling overhead. Encoder computation is measured in operations per source byte, distinguishing finite-field operations for syndrome encoding, operations for error detection, and floating-point operations for any neural predictor.

3 THE PROPOSED CODING SYSTEM

Encoder. We divide the source into contiguous streams of $M + L$ symbols and arrange each set of n streams as the rows of an array in a shared, fixed order (Figure 1). Let $X_i^{(r)}$ denote the symbol at position i in stream r, with lowercase letters denoting realizations. The first M symbols of each stream are transmitted with a conventional lossless compressor to initialize the decoder's context. Each remaining column, $\mathbf{X}_i = (X_i^{(1)}, \ldots, X_i^{(n)})^\mathsf{T}$, is a codeword of *block length* n and is encoded as discussed below. Embed $\mathcal{X}$ in $\mathrm{GF}(q)$, where $q \geq D$ is a prime power, and fix matrices $\mathbf{H}_{i,n} \in \mathrm{GF}(q)^{m_{i,n} \times n}$ known to the encoder and decoder. For $i = M+1, \ldots, M+L$, the encoder computes and sends

$$\mathbf{s}_i = \mathcal{E}_{i,n}(\mathbf{x}_i) := \mathbf{H}_{i,n}\mathbf{x}_i, \tag{1}$$

with computations over $\mathrm{GF}(q)$. $\mathbf{H}_{i,n}$ is the parity check matrix of a linear block code over $\mathrm{GF}(q)$ and $\mathbf{s}_i$ is the syndrome corresponding to $\mathbf{x}_i$. The syndrome rate is $R_{i,n} = m_{i,n} \log_2 q / n$ bits per coded symbol since the binary representation of $\mathbf{s}_i$ requires $\lceil m_{i,n} \log_2 q \rceil$ bits per column. Initial contexts and protocol overhead are counted separately, as specified in Section 2. An important special case is to make $R_{i,n}$ independent of i and to use the same matrix $\mathbf{H}$ for all columns. Notice that the encoder does not use source probabilities or language-model inference.

Decoder. After recovering the initial contexts, the decoder processes the coded columns in order. Define the true context of symbol $X_i^{(r)}$ and the collection of contexts for column i by

$$Y_i^{(r)} = (X_{i-M}^{(r)}, \ldots, X_{i-1}^{(r)}), \qquad \mathbf{Y}_i = (Y_i^{(1)}, \ldots, Y_i^{(n)}). \tag{2}$$

Let P denote the true source law and $Q_i^{(r)}(a \mid y)$ the probability assigned to symbol a by the fixed language model given context y. The decoder uses

$$Q_{i,n}(\mathbf{x} \mid \mathbf{y}) := \prod_{r=1}^{n} Q_i^{(r)}(x_r \mid y_r), \tag{3}$$

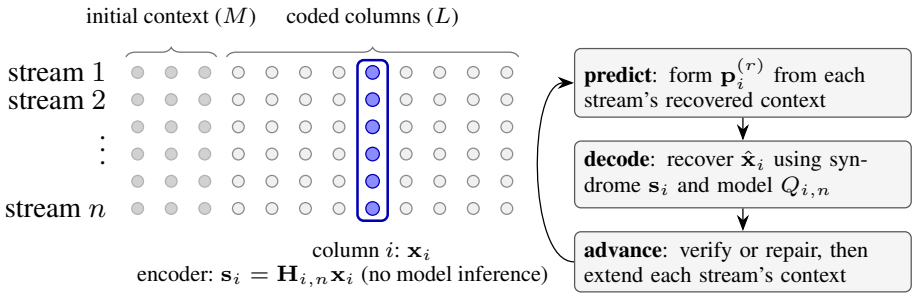


Figure 1: Compression of one set of n streams. Each row contains M initial context symbols, transmitted with a conventional lossless compressor, followed by L coded symbols. The encoder sends a syndrome for each column (blue). The decoder predicts along rows and recovers columns in order; verification and repair are included in the deployment protocol. The text8 experiments use $M = 128$ and $L = 372$.

a normalized probability model that may differ from the true conditional distribution; no independence across source streams is assumed. We denote the prediction vector by $\mathbf{p}_i^{(r)}$, with entries $p_i^{(r)}(a) = Q_i^{(r)}(a \mid y_i^{(r)})$. Given a syndrome and contexts, the maximum *a posteriori* (MAP) decoder computes

$$\mathcal{D}_{i,n}(\mathbf{s}, \mathbf{y}) := \arg \max_{\mathbf{x}' \in \mathcal{X}^n : \mathbf{H}_{i,n}\mathbf{x}' = \mathbf{s}} Q_{i,n}(\mathbf{x}' \mid \mathbf{y}), \tag{4}$$

with a fixed rule for breaking ties. At column i, the decoder forms $\hat{\mathbf{y}}_i$ from previously recovered symbols, computes $\hat{\mathbf{x}}_i = \mathcal{D}_{i,n}(\mathbf{s}_i, \hat{\mathbf{y}}_i)$, and appends each recovered symbol to its stream. These contexts equal the true contexts until the first decoding error. The analysis below uses the MAP decoder, whereas our implementation uses a belief propagation decoder.

Achievable rate. Define the total model log loss of column i, evaluated with its true contexts, by

$$\ell_{i,n} := -\log_2 Q_{i,n}(\mathbf{X}_i \mid \mathbf{Y}_i) = \sum_{r=1}^{n} -\log_2 Q_i^{(r)}(X_i^{(r)} \mid Y_i^{(r)}). \tag{5}$$

The information-spectrum approach characterizes achievable coding rates for general correlated sources without stationarity or ergodicity assumptions (Miyake & Kanaya, 1995). Inspired by this approach, we give a direct random-linear-binning bound in terms of the decoder's model log loss, allowing Q to differ from the true source law P. Expectations below are taken under P.

Theorem 1 (Column coding with a decoder model)**.** *Fix a coded column i, a joint law of $(\mathbf{X}_i, \mathbf{Y}_i)$, and the model $Q_{i,n}$. Draw $\mathbf{H}_{i,n}$ with independent uniform entries in $\mathrm{GF}(q)$, independently of the source. Let $P_{e,i,n}(\mathbf{H}_{i,n}) := \Pr\{\mathcal{D}_{i,n}(\mathbf{H}_{i,n}\mathbf{X}_i, \mathbf{Y}_i) \neq \mathbf{X}_i\}$ be the error probability when the decoder is supplied with the true contexts. For every $\delta > 0$,*

$$\mathbb{E}_{\mathbf{H}_{i,n}}[P_{e,i,n}(\mathbf{H}_{i,n})] \leq \Pr\{\ell_{i,n} > nR_{i,n} - n\delta\} + 2^{-n\delta}. \tag{6}$$

No independence, stationarity, or ergodicity assumption on the source or its contexts is required.

The proof is given in Appendix A. The parameter δ specifies a rate margin in bits per coded symbol, and the two terms account for large model log loss and competing candidates with the same syndrome. For fixed M and L, define

$$\overline{C}_i := \inf\{c : \Pr\{\ell_{i,n}/n > c\} \longrightarrow 0\}. \tag{7}$$

For any $R_i > \overline{C}_i$, choosing $m_{i,n} = \lceil nR_i / \log_2 q \rceil$ and $0 < \delta < R_i - \overline{C}_i$ makes (6) tend to zero as $n \to \infty$. Averaging therefore establishes a vanishing error for each fixed source law and model, although the random ensemble requires no source statistics. A variance-based bound and empirical concentration results are given in Appendices A and C.

Sequential reconstruction. If initialization is exact, the decoder has the true contexts when its first column error occurs. Applying the union bound over L coded columns therefore gives

$$\mathbb{E}_{\mathcal{H}}[P_e(\mathcal{H})] \leq \sum_{i=M+1}^{M+L} \left[\Pr\{\ell_{i,n} > nR_{i,n} - n\delta_i\} + 2^{-n\delta_i}\right], \tag{8}$$

where $\mathcal{H}$ denotes the collection of coding matrices and $\delta_i > 0$. The argument allows dependence among columns and coding matrices, including matrix reuse at a common rate. For fixed L, vanishing column error implies vanishing document error; if the number of columns or sets grows with n, the sum must tend to zero. In Appendix B, we show that under mild conditions on the convergence of the log loss to its mean, the compression performance of the proposed scheme approaches that of LLMZip as $N_c \to \infty$.

LDPC implementation and cost. Theorem 1 establishes existence for random matrices $\mathbf{H}$ and exact maximum a posteriori decoding under Q (4); however, it is computationally infeasible to implement the decoder in (4). In our implementation, at the encoder, we choose $\mathbf{H}$ to be the parity-check matrix of a low-density parity-check (LDPC) code over $\mathrm{GF}(263)$, the smallest prime field containing the $D = 260$ model symbols. Tokens are embedded by the identity map, with zero model probability assigned to the three unused field elements. We choose degree profiles for the LDPC code using differential evolution with left degree $d_v \leq 4$ and right degree $d_c \leq 20$, using simulated decoding. At block lengths from 512 to 16384, the search selects variable degree $d_v = 2$ and a nearly regular check side. The code design thus uses the decoder model offline and the deployed syndrome encoder performs two finite-field multiply-accumulates per coded symbol. By contrast, one BLT-7B prediction from an exact M-byte window, which cannot reuse a key-value cache, costs about 5×10^{11} FLOPs at $M = 128$ and 10^{11} at $M = 16$; the classical compressors in Table 1 are given as orders of magnitude from typical throughput.

The decoder evaluates the model once per symbol, batched across streams, and produces the prediction vector $\mathbf{p}_i^{(r)}$ for every r in the ith column. The decoder then uses sum-product belief propagation algorithm with FFT-based check-node updates (Davey & MacKay, 1998) with $\mathbf{p}_i^{(r)}$ as the initialization for the messages at the variable nodes and the observed syndromes $\mathbf{s}_i$ when computing the check-to-variable messages.

4 Fixed-rate results

This section measures how far the LDPC implementation above falls short of the asymptotic guarantee at finite block length, in the fixed-rate special case of Section 3 ($R_{i,n} = R$ and $\mathbf{H}_{i,n} = \mathbf{H}$).

Corpus construction. We take 512 excerpts of text8 (Mahoney, 2011), about 9.0 million tokens in all, and cut them into 20,480 streams of 500 tokens (Appendix D). Each stream supplies $M = 128$ initial context tokens and $L = 372$ coded tokens, with every prediction using exactly the preceding 128 tokens. At block length n, the streams form $S = \lfloor 20{,}480/n \rfloor$ disjoint sets, giving LS column codewords. The average log loss of the decoder model over all coded bytes, which we denote $\hat{C}_Q$, is 0.934 bits/character. Per-symbol log losses in different streams are uncorrelated ($|r| < 0.01$).

Corpus and resampled codewords. *Corpus codewords* are the columns of these sets, exactly the codewords the encoder emits when it compresses the text8 slice. *Resampled codewords* instead take one coded token, drawn uniformly at random, from each of the n streams of a set. Showing that the document decodes requires the corpus codewords, whereas estimating failure rates requires far more codewords than the document contains (only 1,860 at $n = 4096$), which resampled codewords provide. A resampled codeword is a valid codeword of the scheme: it is the one the encoder would emit if each stream had started at a different token. Each of its tokens is decoded with its true preceding M tokens as context, as if the earlier columns had already been decoded correctly. The *critical rate* $R^*(\mathbf{x})$ of a codeword $\mathbf{x}$ is the lowest rate at which it decodes; decoding on the rate grid measures $\bar{R}(\mathbf{x})$, the lowest grid rate at and above which $\mathbf{x}$ decodes, so $R^*(\mathbf{x})$ lies between the grid rate below $\bar{R}(\mathbf{x})$ and $\bar{R}(\mathbf{x})$.

Effect of block length. Figure 2 plots the codeword failure rate against the fixed rate R on resampled codewords for $n \in \{256, 512, 2048, 4096, 16384\}$. Each curve has the waterfall shape typical of LDPC codes, with FER falling steeply over a narrow range of rates. As n grows the waterfall moves toward $\hat{C}_Q = 0.934$ bits/character, as Theorem 1 predicts: the per-symbol column log loss $\ell_{i,n}/n$ concentrates around its mean (on text8 its standard deviation falls from 0.106 to 0.027 bits

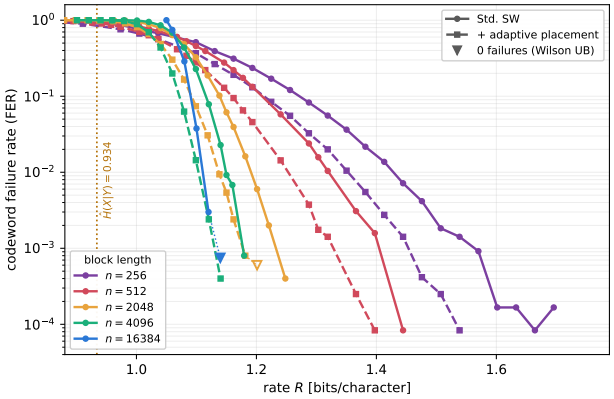


Figure 2: Codeword failure rate versus syndrome rate, standard SW (solid) against adaptive placement (dashed), one color per block length. Triangles are grid points with no failure, plotted at their 95% upper bound.

Table 2: Empirical lossless rate R_{LL} over all corpus codewords, in bits per coded symbol, where a symbol is one byte (a character of text or a sub-pixel of Kodak); the other sources are discussed in Section 6. "adaptive rate" and "both" are three-seed means including signaling and one round of feedback, assuming rate-compatible syndromes (Appendix F), with the predictor trained by cross-validation over sets (Appendix J). Last column: (adaptive placement) − (both), in parentheses as a percentage of adaptive placement. ‡Includes eleven codewords sent uncompressed (Section 6.2).

source	$\hat{C}_Q$	n	Std. SW	adaptive rate	adaptive placement	both	adaptive rate gain
text8	0.934	256	1.73	1.603	1.60	**1.466**	0.13 (8%)
		512	1.50	1.435	1.41	**1.330**	0.08 (6%)
		2048	1.24	1.205	1.20	**1.157**	0.04 (4%)
		4096	1.20	1.163	1.14	**1.110**	0.03 (3%)
		16384	1.13	—	1.08	—	—
Gutenberg	0.969	256	1.75	1.581	1.65	**1.498**	0.15 (9%)
		4096	1.26	1.190	1.22	**1.155**	0.07 (5%)
Python	0.594	256	1.50	**1.338**	1.45	1.350	0.10 (7%)
		4096	0.84	0.788	0.80	**0.765**	0.04 (4%)
Kodak	2.587	256	8.00‡	4.252‡	5.95	**4.033**	1.92 (32%)
		4096	3.12	3.053	3.04	**2.983**	0.06 (2%)

between $n = 256$ and 4096; Appendix C), so the rate margin needed to cover the hardest codewords shrinks.

Empirical lossless rate. For each block length we take all corpus codewords and find the lowest grid rate at which every one of them decodes, so that $\hat{\mathbf{X}} = \mathbf{X}$ on the corpus. We call this rate the *empirical lossless rate* R_{LL}. R_{LL} is selected after decoding the whole corpus, so an encoder cannot choose it in advance. Section 5 gives rates that an encoder can compute.

In the standard SW column of Table 2, R_{LL} falls from 1.73 bits/character at $n = 256$ to 1.13 bits/character at $n = 16384$, and at $n = 4096$ it is 1.20 bits/character, configuration (ii) of Table 1. The other columns are discussed in Section 5.

Failure mechanism at finite block length. A natural hypothesis is that a codeword fails when its mean log loss $\ell_{i,n}/n$ exceeds the rate. The failures we observe are not consistent with it: in a fixed-rate experiment at $n = 4096$ and $R = 1.35$, the two failures had mean log loss 1.015 and 1.009 bits/character, while 110 non-failing codewords had a higher mean log loss than either. The failures are better explained by confidently mispredicted symbols, i.e., positions where the model assigns high probability to a wrong token. BP corrects such a symbol in isolation, but a codeword fails when several of them fall on a trapping set of the LDPC code (Richardson, 2003), so failure

depends on where these symbols sit in the parity-check graph and not only on the mean log loss. Consistent with this, the same codeword can pass under one draw of $\mathbf{H}$ and fail under another.

5 ENCODER-SIDE DECISIONS FROM A WEAK PREDICTOR

Section 4 shows that a fixed-rate codeword fails when several tokens that the decoder's model predicts poorly fall on the same checks of the code graph. The encoder can avoid such failures in two ways: it can raise the rate of codewords that contain many poorly predicted tokens, or it can place the tokens on the code graph so that poorly predicted ones do not share checks; it can also do both. Either decision requires the encoder to know which tokens are likely to be poorly predicted. In this section we give the encoder a weak language model to estimate this, and study the two decisions: the rate of each codeword (Section 5.1) and the placement of its tokens on the code graph (Section 5.2).

The weak model, written $\tilde{Q}$ with prediction vectors $\tilde{\mathbf{p}}_i^{(r)}$, is BLT-7B with its context truncated to $M_{\mathrm{e}} = 16$ characters. Its log loss on the corpus is 1.642 bits/character, better than `xz` (2.135) but worse than `paq8h` (1.40). The truncation stands in for a cheaper model: it lowers the quality of the encoder's predictions without changing the model. Since both decisions use only per-symbol functionals of $\tilde{\mathbf{p}}_i^{(r)}$, such as its entropy and the log loss of the true token, a small or quantized model with full context and a similar log loss could supply comparable inputs at a fraction of the compute.

5.1 CHOOSING THE RATE

Codewords differ in how many syndrome symbols they need, so a single fixed rate over-pays for most of them. We first measure this variation, then predict it with the weak model.

The critical rate. We find $\bar{R}(\mathbf{x})$ for each codeword by decoding it at every grid rate, on samples of 2,500 to 12,000 resampled codewords (Section 4). How much $\bar{R}(\mathbf{x})$ varies across codewords limits what per-codeword rates can save, and the variation shrinks with block length: the standard deviation of $\bar{R}(\mathbf{x})$ is 0.137, 0.093, 0.047 and 0.033 bits/character at $n = 256$, 512, 2048 and 4096. Short blocks therefore have the most to gain, and they also sit furthest above $\hat{C}_Q$. A genie that knew $\bar{R}(\mathbf{x})$ would send each codeword at that rate, plus $\log_2(G)/n$ bits of signaling to name the grid rate; this is the *genie-aided rate* of Table 3.

Rate prediction. The encoder scores each stream by two quantities it can compute from $\tilde{Q}$ and the token it is sending: the log loss $\tilde{\ell}_i^{(r)} = -\log_2 \tilde{p}_i^{(r)}(x_i^{(r)})$ and the miss margin $w_i^{(r)} = \max_a \tilde{p}_i^{(r)}(a) - \tilde{p}_i^{(r)}(x_i^{(r)})$, which is large exactly when a symbol is confidently mispredicted. The n pairs are reduced to 31 permutation-invariant aggregates, mostly of the upper tail. Since decoding on the grid only places $R^*(\mathbf{x})$ between the grid rate below $\bar{R}(\mathbf{x})$ and $\bar{R}(\mathbf{x})$, a small multilayer perceptron is trained on these intervals to predict an estimate $\mu(\mathbf{x})$ of $R^*(\mathbf{x})$ and its uncertainty $\sigma(\mathbf{x})$. Each codeword is sent at $\mu(\mathbf{x})$ plus a margin proportional to $\sigma(\mathbf{x})$, rounded up to the grid, with one factor α^* for all codewords chosen on a validation fold to meet the failure target $\mathrm{FER} \le 10^{-3}$ (Appendix E).

Result. The "adaptive rate" column of Table 3 gives the result. Against a fixed rate calibrated to the same target on the same folds, per-codeword allocation saves 0.068 (17%), 0.085 (28%), 0.030 (22%) and 0.026 bits/character (29%), the percentages relative to the saving of the genie-aided rate. The shortfall is a reliability margin set by the encoder's residual uncertainty about $R^*(\mathbf{x})$, and at the short block lengths, where the learning curve has converged, a better predictor does not remove it: across five model families on identical samples and folds, including a transformer that reads the raw prediction vectors, the best alternative improves on this configuration by at most 0.020 bits/character and captures no more than 37% of the genie-aided gain (Appendix G). The mechanism of Section 4 explains why: the features summarize per-symbol marginals, so they cannot determine where errors fall on the code graph. Section 5.2 therefore changes the placement instead of predicting its effect.

Table 3: Rates in bits/character on resampled codewords of text8 at FER $\leq 10^{-3}$, without feedback: three-seed means on held-out test folds. Per-codeword rates include the $\log_2(G)/n$ signaling overhead.

n	Std. SW	adaptive rate	adaptive placement	both	genie-aided rate
256	1.5695	1.5019	1.5066	**1.3724**	1.1811
512	1.4538	1.3684	1.3753	**1.2705**	1.1498
2048	1.2412	1.2116	1.1706	**1.1572**	1.1081
4096	1.1870	1.1609	1.1347	**1.1044**	1.0964

5.2 Choosing the placement

At finite block length the encoder also chooses which of the n parallel streams is written into which column of $\mathbf{H}$, and whether a codeword fails depends on this assignment as well as on the code. Moving the hard symbols onto the safest positions raises FER tenfold, because those positions are adjacent (Appendix H); the failure mechanism instead suggests spreading the hard symbols apart.

Two rankings. The position ranking orders the variable nodes greedily: each new position is the node whose two checks are least loaded by the positions already chosen, and ties go to the node whose shortest cycle is longest, since short cycles seed trapping sets ((12) in Appendix D). Every prefix of the order is therefore spread across the checks. The order is computed once per code $\mathbf{H}$ and never reads a symbol.

The stream ranking uses the entropy of the encoder model's prediction vector for stream r at column i,

$$h_i^{(r)} \;=\; -\sum_{a\in\mathcal{X}} \tilde{p}_i^{(r)}(a)\log_2 \tilde{p}_i^{(r)}(a), \qquad 1 \le r \le n, \tag{9}$$

a functional of the prediction vector alone, available before the realized token is known. The rankings are matched by placing the hardest stream at v_1, the next at v_2, and so on; the parity-check matrix is unchanged.

Signaling. The permutation is never transmitted. The stream ranking depends only on the prediction vectors that $\tilde{Q}$ computes from the M_{e} symbols preceding column i, so as long as every earlier column has decoded correctly, both terminals obtain the same ranking. Only the ordering of the entropies must agree, with ties broken by stream index, which is weaker than the bit-identical probabilities of arithmetic coding. A disagreement is caught by a per-column error-detection tag, and the decoder must compute $\tilde{Q}$ in addition to its own model.

Result. Placement leaves the rate unchanged. In Figure 2 the dashed curve lies to the left of the solid one at every block length, and in Table 2 the empirical lossless rate falls at every block length, by 0.04 to 0.13 bits/character, most at the short block lengths, without feedback. At fixed rate, placement lowers the failure rate by a factor of 34 to 48, and a fresh draw of $\mathbf{H}$ at $n = 512$ still gives a ninefold reduction (Appendix H).

Combining the two decisions. The two decisions act on different parts of the problem, so their savings can add. On the paired samples, adaptive placement lowers the mean of $\bar{R}(\mathbf{x})$ by 0.041 to 0.061 bits/character while its standard deviation falls by only 9 to 12%, so most of the variation that the rate predictor uses remains. Table 3 confirms the consequence: at every block length the combination is better than either decision alone, and at three of the four block lengths it exceeds the sum of the two individual savings, by 0.067 bits/character at $n = 256$.

6 Other corpora and a second modality

The code, decoder and encoder-side decisions were developed on text8. This section tests whether they carry over, first to two other text corpora with the same decoder model (Section 6.1) and then to images, where the decoder's model is adapted to pixels (Section 6.2). Section 6.3 explains why the value of adaptive rate differs across sources.

6.1 SAME MODALITY, NEW CORPORA

With the same decoder model we repeat the pipeline on 25 Project Gutenberg books ($\hat{C}_Q = 0.969$ bits/character) and on the CPython 3.11 source tree ($\hat{C}_Q = 0.594$ bits/character), cut into 500-token streams as before (Appendix J). Table 2 shows the same pattern as on text8: adaptive rate on top of adaptive placement saves 0.04 to 0.07 bits/character at $n = 4096$ and 0.10 to 0.15 bits/character at $n = 256$ on both corpora. On Python at $n = 256$ the predictor is weak and adaptive rate alone is better than the combination by 0.012 bits/character.

6.2 A SECOND MODALITY: PIXELS

The results so far have been on text, but the encoder only sees bytes. The scheme therefore applies to any source written as bytes, provided the decoder has a model that predicts over the same byte alphabet. Chen et al. (2024) show that a language model fine-tuned on images compresses them well when pixels are tokenized appropriately. We fine-tune BLT-7B, which reads bytes directly, to obtain a byte-level prediction model for images. It is trained on Kodak images 1 to 12 and tested on images 13 to 24, where it reaches a test log loss of 2.467 bits per sub-pixel. LUMI (Tian et al., 2026), trained and tested on the same split, reports a log loss of 2.853. The model does not, however, generalize beyond Kodak: on DIV2K images it improves on PNG by only 8% (Appendix I).

In general the streams of a codeword may come from different images or from the same one. In our experiments each image is flattened patch by patch, following Tian et al. (2026) and Chen et al. (2024), and cut into 500-byte streams, so the streams come from different patches, often of the same image. The streams cover 89% of the test-image bytes, and Appendix I describes the dataset construction.

With the dataset and the model fixed, the experiments follow those on text8. The encoder's weak predictor is the same model with a 16-byte context, and the decoder uses a 128-byte context, at which its log loss on the test images is $\hat{C}_Q = 2.587$ bits per sub-pixel. The LDPC codes keep the variable degree $d_v = 2$ found on text8 by the profile search, which adaptive placement also requires. The rate predictor is retrained on Kodak, because the text8 predictor does not transfer (Appendix J).

The Kodak rows of Table 2 give the result. At $n = 4096$ the scheme reaches empirical lossless rates of 3.12 with no model at the encoder, 3.04 with adaptive placement and 2.98 with both. At $n = 256$ standard SW has no lossless fixed rate (eleven codewords are sent raw); placement restores one at 5.95, and the per-codeword rate on top of it reaches 4.03, a saving of 1.92 bits per sub-pixel or 32% of the adaptive-placement rate, where the three text sources save 7 to 9%. Taken in the other order, adaptive rate alone reaches 4.25 and placement on top of it saves a further 0.22.

6.3 THE VALUE OF ADAPTIVE RATE AND THE TAIL OF THE LOG LOSS

A single global rate must cover the hardest codeword in the corpus, whereas per-codeword rates cover each codeword separately. The saving from adaptive rate therefore grows with the excess log loss of the hardest codeword under the decoder's model, $\Delta_{\max} := \max_i \ell_{i,n}/n - \hat{C}_Q$, with the maximum over all corpus codewords. Figure 4 (Appendix C) shows the distributions and the saving against $\Delta_{\max}$. The spread falls as $1/\sqrt{n}$ on the text sources, and roughly so on Kodak, but the tails differ far more: at $n = 256$ the hardest text8 codeword is 0.42 bits above $\hat{C}_Q$ and the hardest Kodak codeword 3.35 bits above, while at $n = 4096$ both fall, to 0.10 and 0.24. Part of the Kodak tail is the adapter's overconfidence (Appendix I).

7 CONCLUSION

We have asked how much of LLMZip's compression is retained when the encoder runs no language model. By the Slepian–Wolf theorem none need be lost as the block length grows, and at $n = 4096$ an LDPC coset code over $\mathrm{GF}(263)$ decodes our text8 corpus losslessly with no model at the encoder. A deliberately weak predictor at the encoder supports two further decisions, the rate of each codeword and the assignment of its symbols to the code graph, and together they lower the empirical lossless rate at every block length. The same scheme reaches lossless rates on Gutenberg, Python source and Kodak pixels with only the side-information model and the rate predictor changed.

REPRODUCIBILITY STATEMENT

Appendix D specifies the codes and the decoder, Appendix E the rate predictor, Appendix F the feedback repair, Appendix I the Kodak streams and adapter, and Appendix J the other corpora and the cross-validation protocol. All coding results are computed from precomputed prediction vectors of the decoder and encoder models, so they can be reproduced without running the language models. Codeword positions and code construction use fixed seeds, with a second construction seed for the fresh-draw control of Appendix H. A codeword counts as decoded only if the decoded vector equals the source, so a wrong codeword that satisfies the syndrome counts as a failure. The closed-loop decode of each corpus reproduces the reported run exactly: BP is deterministic, and its predictions from recovered contexts equal those from the source contexts as long as no earlier codeword has failed. Code will be released upon publication.

AI USE STATEMENT

In this work, we used generative AI tools for writing and refactoring the experiment drivers, label-assembly and analysis code, for the scripts that produce the figures from result files, and for drafting and editing the text. We have not used generative AI tools for research ideation, experimental design, or the interpretation of results, and the remaining categories requiring disclosure are not applicable to this work. Additionally, we used generative AI tools for editorial review of complete drafts. We have reviewed all AI-assisted work. AI-generated code was checked by re-deriving every number reported in the paper from the logged per-codeword result files, and each claim in the text was verified against those files by the authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Kapil Bhattad and Krishna R. Narayanan. A decision feedback based scheme for Slepian–Wolf coding of sources with hidden Markov correlation. *IEEE Communications Letters*, 10(5):378–380, 2006.

Kecheng Chen, Pingping Zhang, Hui Liu, Jie Liu, Yibing Liu, Jiaxin Huang, Shiqi Wang, Hong Yan, and Haoliang Li. Large language models for lossless image compression: Next-pixel prediction in language space is all you need, 2024. arXiv:2411.12448.

Imre Csiszár. Linear codes for sources and source networks: Error exponents, universal coding. *IEEE Transactions on Information Theory*, 28(4):585–592, 1982.

Matthew C. Davey and David J. C. MacKay. Low-density parity check codes over GF(q). *IEEE Communications Letters*, 2(6):165–167, 1998.

Grégoire Delétang, Anian Ruoss, Paul-Ambroise Duquenne, Elliot Catt, Tim Genewein, Christopher Mattern, Jordi Grau-Moya, Li Kevin Wenliang, Matthew Aitchison, Laurent Orseau, Marcus Hutter, and Joel Veness. Language modeling is compression. In *International Conference on Learning Representations (ICLR)*, 2024.

Javier Garcia-Frias and Ying Zhao. Compression of correlated binary sources using turbo codes. *IEEE Communications Letters*, 5(10):417–419, 2001.

Xiao-Yu Hu, Evangelos Eleftheriou, and Dieter-Michael Arnold. Regular and irregular progressive edge-growth Tanner graphs. *IEEE Transactions on Information Theory*, 51(1):386–398, 2005.

Angelos D. Liveris, Zixiang Xiong, and Costas N. Georghiades. Compression of binary sources with side information at the decoder using LDPC codes. In *IEEE Communications Letters*, 2002.

Matt Mahoney. About the test data, 2006. `http://mattmahoney.net/dc/textdata.html`.

Matt Mahoney. Text8 corpus: cleaned wikipedia dump. *`http://mattmahoney.net/dc/textdata.html`*, 2011.

Quinn McNemar. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12(2):153–157, 1947.

Nitish Mital, Ezgi Ozyilkan, Ali Garjani, and Deniz Gunduz. Neural distributed image compression using common information. *IEEE Data Compression Conference (DCC)*, 2022.

Shigeki Miyake and Fumio Kanaya. Coding theorems on correlated general sources. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E78-A(9): 1063–1070, 1995.

Artidoro Pagnoni, Ram Pasunuru, Pedro Rodriguez, John Nguyen, Benjamin Muller, Margaret Li, Chunting Zhou, Lili Yu, Jason Weston, Luke Zettlemoyer, Gargi Ghosh, Mike Lewis, Ari Holtzman, and Srinivasan Iyer. Byte latent transformer: Patches scale better than tokens. *arXiv preprint arXiv:2412.09871*, 2024.

S. Sandeep Pradhan and Kannan Ramchandran. Distributed source coding using syndromes (DISCUS): design and construction. *IEEE Transactions on Information Theory*, 49(3):626–643, 2003.

Tom Richardson. Error floors of LDPC codes. *Proc. 41st Annual Allerton Conf. on Communication, Control, and Computing*, 2003.

Chris Xing Tian, Chengkai Wu, Ziyu Wang, Rongqun Lin, Kecheng Chen, Xiandong Meng, Haoliang Li, Shiqi Wang, and Siwei Ma. LUMI: Tokenizer-agnostic LLM-based lossless image compression, 2026. arXiv:2607.08221.

Chandra Shekhara Kaushik Valmeekam, Krishna Narayanan, Dileep Kalathil, Jean-Francois Chamberland, and Srinivas Shakkottai. LLMZip: Lossless text compression using large language models. *arXiv preprint arXiv:2306.04050*, 2023.

David Varodayan, Anne Aaron, and Bernd Girod. Rate-adaptive codes for distributed source coding. *Signal Processing*, 86(11):3123–3130, 2006.

A Proof of Theorem 1

Fix a column i and abbreviate $\mathbf{H} = \mathbf{H}_{i,n}$, $m = m_{i,n}$, and $Q = Q_{i,n}$. Condition on a source column $\mathbf{x}$ and its true contexts $\mathbf{y}$. When $Q(\mathbf{x} \mid \mathbf{y}) > 0$, any distinct candidate that can be selected in preference to $\mathbf{x}$ by (4) belongs to

$$\mathcal{A}(\mathbf{x}, \mathbf{y}) := \{\mathbf{x}' \in \mathcal{X}^n \setminus \{\mathbf{x}\} : Q(\mathbf{x}' \mid \mathbf{y}) \geq Q(\mathbf{x} \mid \mathbf{y})\}.$$

Since $Q(\cdot \mid \mathbf{y})$ is normalized, $|\mathcal{A}(\mathbf{x}, \mathbf{y})| \leq 1/Q(\mathbf{x} \mid \mathbf{y})$. This counting step uses only the decoder model, and makes no assumption about the true conditional distribution of the column.

For any $\mathbf{x}' \neq \mathbf{x}$, the embedded difference $\mathbf{x}' - \mathbf{x}$ is nonzero in $\mathrm{GF}(q)^n$. Its inner product with a uniformly random row of $\mathbf{H}$ is uniform in $\mathrm{GF}(q)$. Independence of the m rows therefore gives

$$\Pr_{\mathbf{H}}\{\mathbf{H}\mathbf{x}' = \mathbf{H}\mathbf{x}\} = q^{-m}.$$

The source and the model are fixed independently of the matrix draw, so the candidate set is also fixed under this conditioning. A union bound over its members yields

$$\Pr_{\mathbf{H}}\{\mathcal{D}_{i,n}(\mathbf{H}\mathbf{x}, \mathbf{y}) \neq \mathbf{x}\} \leq \min\left\{1, \frac{q^{-m}}{Q(\mathbf{x} \mid \mathbf{y})}\right\}.$$

For a source realization $(\mathbf{x}, \mathbf{y})$, define

$$e(\mathbf{x}, \mathbf{y}) := \Pr_{\mathbf{H}}\{\mathcal{D}_{i,n}(\mathbf{H}\mathbf{x}, \mathbf{y}) \neq \mathbf{x}\}.$$

For any $\delta > 0$, let

$$\mathcal{G} := \{\ell_{i,n} \leq nR_{i,n} - n\delta\}.$$

Since $m \log_2 q = nR_{i,n}$, the preceding bound gives $e(\mathbf{X}_i, \mathbf{Y}_i) \leq 2^{-n\delta}$ on $\mathcal{G}$. On $\mathcal{G}^c$, we use $e(\mathbf{X}_i, \mathbf{Y}_i) \leq 1$, which also covers realizations with $Q(\mathbf{X}_i \mid \mathbf{Y}_i) = 0$.

Independence of the source and coding matrix allows us to average in either order. Writing $P(\mathbf{x}, \mathbf{y})$ for the true joint source law, we obtain

$$\begin{aligned}\mathbb{E}_{\mathbf{H}}[P_{e,i,n}(\mathbf{H})] &= \sum_{\mathbf{x},\mathbf{y}} P(\mathbf{x},\mathbf{y})\, e(\mathbf{x},\mathbf{y}) \\ &= \mathbb{E}_P[e(\mathbf{X}_i,\mathbf{Y}_i)\mathbf{1}_{\mathcal{G}}] + \mathbb{E}_P[e(\mathbf{X}_i,\mathbf{Y}_i)\mathbf{1}_{\mathcal{G}^c}] \\ &\le 2^{-n\delta} \Pr_P(\mathcal{G}) + \Pr_P(\mathcal{G}^c) \\ &\le \Pr_P\{\ell_{i,n} > nR_{i,n} - n\delta\} + 2^{-n\delta},\end{aligned}$$

which proves (6). □

The sequential bound. For fixed matrices $\mathcal{H}$, let F_i be the event that the sequential decoder's first error occurs in column i, and let A_i be the event that $\mathcal{D}_{i,n}(\mathbf{H}_{i,n}\mathbf{X}_i, \mathbf{Y}_i) \neq \mathbf{X}_i$ when given the true contexts. Correct initialization and the absence of earlier errors imply $F_i \subseteq A_i$. Hence $P_e(\mathcal{H}) \le \sum_i P_{e,i,n}(\mathbf{H}_{i,n})$. Taking expectations and applying Theorem 1 with margin δ_i to each summand proves (8). Neither independence between columns nor independence between their matrices is needed. In particular, the same matrix can be reused whenever the syndrome dimensions agree. Averaging the total error bound also gives a deterministic collection of matrices with no larger document error, for each fixed source law and decoder model.

A variance condition for error decay. Suppose $\ell_{i,n}$ has a finite second moment and put $\mu_{i,n} := \mathbb{E}[\ell_{i,n}]/n$. If $R_{i,n} \ge \mu_{i,n} + 2\delta$ for some $\delta > 0$, applying (6) and Chebyshev's inequality gives

$$\mathbb{E}_{\mathbf{H}_{i,n}}[P_{e,i,n}(\mathbf{H}_{i,n})] \le \frac{\mathrm{Var}(\ell_{i,n})}{n^2\delta^2} + 2^{-n\delta}. \tag{10}$$

Thus $\mathrm{Var}(\ell_{i,n}) = o(n^2)$ suffices for vanishing column error, and $\mathrm{Var}(\ell_{i,n}) \le nV$ gives the explicit bound $V/(n\delta^2) + 2^{-n\delta}$. Writing $U_{i,r} := -\log_2 Q_i^{(r)}(X_i^{(r)} \mid Y_i^{(r)})$, a sufficient condition for the latter is

$$\sup_{1\le r\le n} \sum_{s=1}^{n} |\operatorname{Cov}(U_{i,r}, U_{i,s})| \le V$$

uniformly in n. This condition permits dependent, nonidentically distributed log losses. An exponential bound on the upper tail of $\ell_{i,n}$ instead gives exponential decay through (6).

B Asymptotic Comparison with LLMZip

This appendix shows that the scheme's rate approaches LLMZip's as the block length and the stream length grow. For fixed M, consider $N_c = n(M + L)$ source symbols, and let $B_{\text{init}}^{\text{LLMZip}}$ and B_{init} denote the bits required by LLMZip and our scheme, respectively, to encode the first M columns (nM symbols). If initialization rates are bounded, both costs are $O(nM)$. Using the same model and contexts, LLMZip's total length is approximately $B_{\text{init}}^{\text{LLMZip}} + \sum_{i=M+1}^{M+L} \ell_{i,n}$ bits. For each fixed L, suppose $\ell_{i,n}/n$ converges in probability to a finite constant $C_{i,L}$ at every coded position, and assume $\overline{C}_L := L^{-1}\sum_{i=M+1}^{M+L} C_{i,L} \to C_Q$ as $L \to \infty$. Assigning rates $C_{i,L} + 2\delta$ for fixed $\delta > 0$, rounded up to whole syndrome symbols, then yields deterministic codes with $P_e \to 0$ as $n \to \infty$ by (8). Their rate, excluding other protocol overhead, satisfies

$$\frac{K}{N_c} \le \frac{B_{\text{init}}}{n(M+L)} + \frac{L}{M+L}(\overline{C}_L + 2\delta) + O(1/n).$$

Consequently, for any $\eta, \epsilon > 0$, a rate at most $C_Q + \eta$ with $P_e \le \epsilon$ is attainable by first choosing a sufficiently large, finite L to amortize initialization and make $\overline{C}_L$ close to C_Q, and then choosing δ sufficiently small and n sufficiently large to control the decoding error. The resulting rate approaches LLMZip's limiting model log loss C_Q, provided additional overhead is $o(N_c)$. If a common rate is used, $R > \max_i C_{i,L}$ also ensures vanishing error, but for a higher fixed cost per token.

C EMPIRICAL CONCENTRATION OF COLUMN LOG LOSS

Theorem 1 needs the column log loss to concentrate around its mean; we check this empirically. We examine the normalized column log loss (also referred to as surprisal) $A_{i,n} := \ell_{i,n}/n$ under the decoder's $M = 128$ model, using all corpus codewords. Figure 3 shows the distribution of $A_{i,n}$ across columns for different block lengths n.

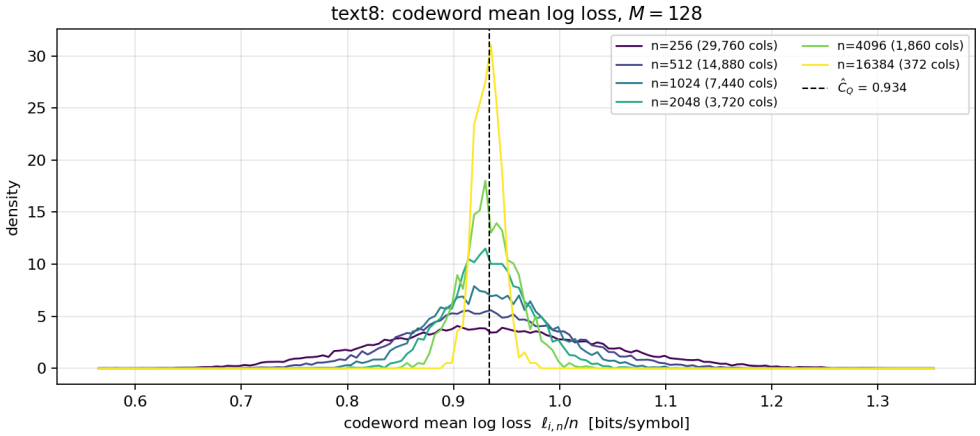


Figure 3: Empirical distributions of normalized column log loss $A_{i,n} = \ell_{i,n}/n$ on text8, with $M = 128$. The dashed line denotes the empirical model cross-entropy, 0.934 bits per symbol. The legend gives the number of column codewords at each length.

On text8, increasing n from 256 to 4096 reduces the empirical standard deviation from 0.106 to 0.027 bits per symbol. Across the five lengths in this range, $n\widehat{\mathrm{Var}}(A_{i,n})$ stays between approximately 2.80 and 2.99, consistent with variance decreasing as $1/n$. The empirical 99.9th percentile decreases from 1.280 to 1.023 bits per symbol between $n = 256$ and 4096. Over the same lengths, standard deviations decrease from 0.104 to 0.025 on Gutenberg, 0.101 to 0.022 on Python, and 0.343 to 0.063 on Kodak (bits per sub-pixel for Kodak). All four sources exhibit narrowing distributions, although the variance scaling is less uniform on Kodak. The mean remains unchanged because the same symbols are regrouped; the evidence for concentration is the shrinking spread and upper tail.

If $\mathrm{Var}(A_{i,n}) \le V/n$ holds under the source law for each fixed position i, then $\mathrm{Var}(\ell_{i,n}) \le nV$. Theorem 1 and (10) bound column error, averaged over the coding matrix, by $V/(n\delta^2) + 2^{-n\delta}$ whenever $R_{i,n} \ge \mathbb{E}[A_{i,n}] + 2\delta$. If the means also converge to C_i, the spectral threshold in (7) equals C_i, so any fixed rate above C_i is achievable with vanishing column error. These finite, pooled measurements support this sufficient condition but do not establish it asymptotically for every position, nor guarantee the same behavior for LDPC decoding with BP.

D STREAMS, CODES, DECODER AND POSITION ORDER

This appendix specifies how the streams are formed, the codes and the decoder used in all experiments, and the position order used by adaptive placement.

Streams. Each corpus is first cut into 512 segments. From each segment we read chunks of 1000 bytes at a stride of 872, so consecutive chunks overlap by 128 bytes, and split each chunk into two streams of 500 bytes, each with $M = 128$ bytes of initial context and $L = 372$ coded bytes. The initial context of a chunk's first stream is then the tail of the previous chunk, which the decoder has already recovered, so only the second stream's initial context is sent uncoded. Text8 uses 20 chunks per segment, which gives 20,480 streams over about 9.0 million distinct tokens, and Gutenberg, Python and Kodak use 28 (Appendices I and J). Streams are assigned to sets in segment order. Byte value b enters BLT as token $b + 4$, above its four special tokens, and every position is predicted from exactly the preceding 128 bytes of its stream at the decoder, or 16 at the encoder.

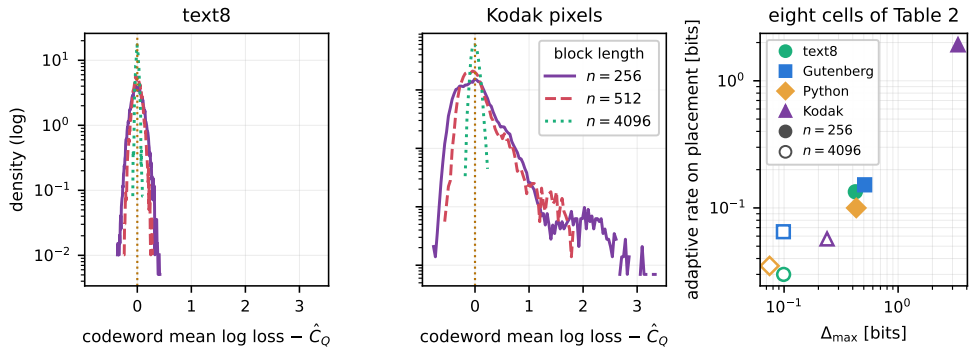


Figure 4: Left, center: distribution of the codeword mean log loss $\ell_{i,n}/n$ under the decoder's side information over all corpus codewords, centered on each source's empirical model cross-entropy $\hat{C}_Q$ (dotted), density on a log scale. Right: adaptive rate on top of adaptive placement against $\Delta_{\max}$ of Section 6.3, for the eight source and block-length pairs of Table 2 at $n = 256$ and 4096.

Degree distributions. The profile search of Section 3 returns one profile per block length, whereas the rate sweeps need a code at every grid rate. We therefore fix the variable degree at $d_v = 2$, so that the variable-node degree distribution, from the node perspective, is x^2, and let the rate set the check side. Counting edges from both sides gives $2n = \bar{d}_c\, m$, where $\bar{d}_c$ is the average check degree, so the rate $R = m \log_2 q/n$ fixes $\bar{d}_c = 2 \log_2 q/R \approx 16.1/R$. The check-node degree distribution, from the node perspective, is

$$(1-\theta)\, x^d + \theta\, x^{d+1}, \qquad d = \lfloor \bar{d}_c \rfloor, \;\; \theta = \bar{d}_c - d, \tag{11}$$

so every check node has one of the two consecutive degrees that bracket $\bar{d}_c$. The average $\bar{d}_c$ is about 9 to 15 at the text8 and Gutenberg rates of Table 2, up to 21 on Python, beyond the search's $d_c \le 20$, and 2.7 to 5.4 on Kodak. We did not rerun the search at each rate or at $n = 256$, so at rates far from those the search selected these profiles are not optimized.

Construction and decoding. Edges are placed by progressive edge growth (Hu et al., 2005), which connects each variable node to the check nodes farthest from it in the graph built so far, avoiding short cycles. Each nonzero entry of $\mathbf{H}$ is drawn uniformly from $\mathrm{GF}(263) \setminus \{0\}$. All experiments use one construction seed, except the fresh-draw control of Appendix H. The decoder is sum-product BP over $\mathrm{GF}(263)$ with FFT-based check-node updates, initialized from the prediction vectors. Each column stops as soon as its syndrome is satisfied, or after 40 iterations.

The position order for adaptive placement. Adaptive placement (Section 5.2) writes the stream with the highest entropy into position v_1, the next into v_2, and so on. The order $v_1, \ldots, v_n$ depends only on $\mathbf{H}$ and is defined as follows. Label the variable nodes $1, \ldots, n$, and write $c_1(v), c_2(v)$ for the two checks of node v and $g(v)$ for the length of the shortest cycle through v. The positions are chosen one at a time. Once $v_1, \ldots, v_{k-1}$ are fixed, let $\mathcal{V}_k$ be the remaining nodes and $\kappa_{k-1}(c)$ the number of $v_1, \ldots, v_{k-1}$ connected to check c, with $\kappa_0 \equiv 0$. Writing κ for κ_{k-1}, the k-th position is

$$\begin{aligned}
\mathcal{T}_1 &= \arg\min_{v \in \mathcal{V}_k} \max\{\kappa(c_1(v)), \kappa(c_2(v))\}, \\
\mathcal{T}_2 &= \arg\min_{v \in \mathcal{T}_1} \big(\kappa(c_1(v)) + \kappa(c_2(v))\big), \\
\mathcal{T}_3 &= \arg\max_{v \in \mathcal{T}_2} g(v), \\
v_k &= \min \mathcal{T}_3, \qquad k = 1, \ldots, n,
\end{aligned} \tag{12}$$

where each $\arg\min$ and $\arg\max$ returns the set of all minimizers or maximizers, so each line only breaks the ties left by the line above it. The first two lines choose the node whose checks are least loaded, so every prefix of the order is spread across the checks. The third prefers, among those, the node whose shortest cycle is longest, since short cycles seed trapping sets. The last picks the smallest label, so the order is unique and both terminals obtain the same one.

E THE RATE PREDICTOR

This appendix describes the rate predictor of Section 5.1: its input features, its training loss, and how its output sets each codeword's rate.

Features. The features are built from two per-symbol quantities, both computable at the encoder from the encoder model $\tilde{Q}$ ($M_{\mathrm{e}} = 16$) and the token being sent. For stream r of a codeword, with prediction vector $\tilde{\mathbf{p}}_i^{(r)}$ and true token $x_i^{(r)}$, they are the log loss $\tilde{\ell}_i^{(r)} = -\log_2 \tilde{p}_i^{(r)}(x_i^{(r)})$ in bits, and the *miss margin* $w_i^{(r)} = \max_a \tilde{p}_i^{(r)}(a) - \tilde{p}_i^{(r)}(x_i^{(r)})$ in $[0, 1]$, which is near zero when the model is right or merely uncertain, and near one exactly when it is confident and wrong. Three further per-symbol quantities are recorded and enter only through their means: the entropy of the prediction vector, and the largest and second-largest predicted probabilities.

The n streams of a codeword are then reduced to 31 aggregates, chosen to be permutation invariant so that the identical feature matrix serves standard SW and adaptive placement:

- means of $\tilde{\ell}$, w, the entropy, the top-1 and top-2 probabilities, and the fraction of streams with $w > 0$, whose most probable token is wrong (6);
- maxima of $\tilde{\ell}$ and w (2);
- the 50th, 90th and 99th percentiles of $\tilde{\ell}$ and of w (6);
- means of the top 10, 25, 100 and 500 values of $\tilde{\ell}$ and of w (8);
- counts of streams with $\tilde{\ell}$ above 6, 8 and 10 bits, and with w above 0.3, 0.5 and 0.7 (6);
- soft maxima $\beta^{-1} \log \sum_r e^{\beta \tilde{\ell}^{(r)}}$ at $\beta = 0.1, 0.3, 1$ (3).

The aggregates emphasize the upper tail, following the failure mechanism of Section 4: a codeword fails through a cluster of confidently mispredicted symbols rather than through a high mean log loss. The features use only $\tilde{Q}$ and the transmitted tokens; no quantity from the decoder's model or from decoding enters.

The interval-censored likelihood. Decoding a sample on a rate grid does not reveal the critical rate $R^*(\mathbf{x})$, only brackets it. A codeword that fails at grid rate r_{lo} and decodes at the next rate r_{hi} contributes the observation $R^*(\mathbf{x}) \in (r_{\mathrm{lo}}, r_{\mathrm{hi}}]$, with $r_{\mathrm{hi}} = \bar{R}(\mathbf{x})$, and the grid spacing bounds how tightly. We therefore do not regress on a point label. The network maps the aggregates to an estimate $\mu(\mathbf{x})$ of $R^*(\mathbf{x})$ and an uncertainty $\sigma(\mathbf{x})$, and training maximizes the probability the induced logistic law places on the observed bracket,

$$\mathcal{L}(\mathbf{x}) \;=\; F\!\left(\frac{r_{\mathrm{hi}} - \mu(\mathbf{x})}{\sigma(\mathbf{x})}\right) - F\!\left(\frac{r_{\mathrm{lo}} - \mu(\mathbf{x})}{\sigma(\mathbf{x})}\right), \qquad F(z) = \frac{1}{1 + e^{-z}}, \tag{13}$$

minimizing $-\sum_{\mathbf{x}} \log \mathcal{L}(\mathbf{x})$. This form also handles one-sided observations: a codeword that decodes even at the lowest grid rate sets $r_{\mathrm{lo}} = -\infty$ and contributes $\log F((r_{\mathrm{hi}} - \mu(\mathbf{x}))/\sigma(\mathbf{x}))$, and one that fails at every rate sets $r_{\mathrm{hi}} = +\infty$ symmetrically.

Each codeword is sent at the quantile $\mu(\mathbf{x}) + \sigma(\mathbf{x}) \log \frac{\alpha}{1-\alpha}$, rounded up to the grid; sending at $\mu(\mathbf{x})$ would fail half the codewords. The uncertainty $\sigma(\mathbf{x})$ sets the margin: under one global quantile level α^*, codewords with small $\sigma(\mathbf{x})$ are sent close to $\mu(\mathbf{x})$ and codewords with large $\sigma(\mathbf{x})$ further above it. A point regression would provide no such margin.

F FEEDBACK FOR PER-CODEWORD RATES

The learned columns of Table 2 send each codeword at a predicted rate $\hat{r}(\mathbf{x})$. A codeword sent below its grid critical rate $\bar{R}(\mathbf{x})$ fails, and it is decodable only after the decoder reports the failure and the encoder sends more syndrome symbols. This appendix describes how these resends are counted in Table 2, what they would cost with the codes we use, and the cost of meeting the failure target $\mathrm{FER} \leq 10^{-3}$ of Table 3 on the test fold. The fixed-rate configurations and adaptive placement need no feedback.

How a resend is counted. The numbers in Table 2 assume a rate-compatible code (Varodayan et al., 2006), in which a resend adds only the syndrome symbols needed to reach $\bar{R}(\mathbf{x})$, so a codeword costs $\max(\hat{r}(\mathbf{x}), \bar{R}(\mathbf{x}))$ in total and every corpus codeword decodes. However, our experiments do not use rate-compatible codes. Each grid rate has its own code, built independently (Appendix D), and we leave rate-compatible LDPC codes over $\mathrm{GF}(263)$ to future work. With our codes a failed codeword would be resent from scratch at $\bar{R}(\mathbf{x})$, costing $\hat{r}(\mathbf{x}) + \bar{R}(\mathbf{x})$, which raises the learned columns of Table 2 by at most 0.016 bits/character on text and 0.038 bits per sub-pixel on Kodak.

The failure target $\mathrm{FER} \leq 10^{-3}$ of Table 3. Table 3 uses no feedback. Its learned columns meet $\mathrm{FER} \leq 10^{-3}$ on the validation fold, and four of the eight exceed it slightly on the held-out test fold. Raising the rates of the codewords that fail, cheapest first, until the test fold meets the target would add at most 0.0002 bits/character (Table 4).

Table 4: Cost of enforcing $\mathrm{FER} \leq 10^{-3}$ on the held-out test fold for the learned columns of Table 3, three-seed means. "Enforced" is the rate after raising the failing codewords, cheapest first, until the test fold meets the target.

n	configuration	Table 3 [bits/character]	test FER	enforced [bits/character]
256	adaptive rate	1.5019	0.12%	1.5019
256	both	1.3724	0.08%	1.3725
512	adaptive rate	1.3684	0.06%	1.3685
512	both	1.2705	0.14%	1.2706
2048	adaptive rate	1.2116	0.03%	1.2116
2048	both	1.1572	0.03%	1.1572
4096	adaptive rate	1.1609	0.20%	1.1610
4096	both	1.1044	0.13%	1.1046

G WHY ADAPTIVE RATE FALLS SHORT OF THE GENIE-AIDED RATE

This appendix asks why adaptive rate recovers only 17 to 29% of the saving of the genie-aided rate (Section 5.1). A larger predictor does not close the gap, and neither does more training data, so the gap comes from how predictable the critical rate is from features of the weak encoder model. All comparisons use the samples of Section 5.1, split into training, validation and test folds by set, so that codewords sharing streams never appear in both the training and the test data. We examine three other possible causes below.

- **The rate-allocation rule.** Replacing the predictor's estimate $\mu(\mathbf{x})$ with the measured $\bar{R}(\mathbf{x})$, under the same margin rule, recovers exactly the genie-aided rate. The gap therefore comes from the predictor's errors in estimating $R^*(\mathbf{x})$, not from how the margin is set.
- **The model class.** Table 5 compares five predictors, up to a transformer that reads the raw prediction vector of every symbol, the most detailed input available at the encoder. The best improves on the deployed perceptron by at most 0.020 bits/character, and none recovers more than 37% of the saving of the genie-aided rate.
- **The amount of data.** Growing the training set from 25% to 100% of the available codewords lowers the test error of the predicted rate only from 0.109 to 0.105 bits/character (RMS) at $n = 256$ and from 0.079 to 0.073 at $n = 512$, close to the training error, so more data would not help there. At $n = 4096$ the same increase lowers the test error from 0.069 to 0.032, so more data might help there.

H ADDITIONAL EXPERIMENTS ON ADAPTIVE PLACEMENT

This appendix supports the placement results of Section 5.2 with experiments at fixed rate. It shows that the reduction in failures is statistically reliable, that it holds for a second parity-check matrix, how much better stream scoring could add, and that two simpler placement rules do not work. The position order itself is defined in Appendix D. These experiments decode fixed samples of $10{,}000$ codewords at a few rates, rather than the rate-swept samples of Figure 2.

Table 5: Five R^* predictor families on identical samples and folds (FER $\leq 10^{-3}$, three-seed mean, bits/character). "fixed" is the single-rate baseline and "genie-aided" sends each codeword at $\bar{R}(\mathbf{x})$. The interval-censored perceptron is the configuration deployed in Section 5.1.

n	fixed	censored MLP	XGBoost AFT	boosted rate scan	transformer	transformer + $\tilde{\mathbf{p}}$	genie-aided rate
256	1.5695	1.5019	1.4862	**1.4853**	1.5060	1.5027	1.1811
512	1.4538	1.3684	1.3826	**1.3595**	1.3705	1.3615	1.1498
2048	1.2412	1.2116	1.2197	1.2012	1.1983	**1.1916**	1.1081
4096	1.1870	1.1609	1.1795	1.1604	1.1577	**1.1569**	1.0964

Paired comparison. Each experiment decodes the same codewords at one rate twice, once under standard SW and once under adaptive placement. Codewords that decode both times, or fail both times, carry no information about the difference. What matters is how many codewords adaptive placement repairs, i.e., fails under standard SW but decodes under placement, and how many it breaks. If placement were no better than standard SW (as-is placement), each of these codewords would be as likely to be repaired as broken, like the flip of a fair coin. McNemar's exact test (McNemar, 1947) gives the probability that such coin flips alone would produce a split at least as uneven as the one observed.

At $n = 4096$ and $R = 1.12$, in the waterfall region, FER falls from 8.95% to 0.26%, with 869 codewords repaired and none broken. At $n = 2048$ and $R = 1.18$ it falls from 1.94% to 0.04%, with 191 repaired and one broken, and at $n = 4096$ and $R = 1.15$ from 1.32% to zero, with 132 repaired and none broken. In all three the McNemar p-value is effectively zero. Placement therefore lowers the failure rate by a factor of 34 to 48 and almost never breaks a codeword that decodes without it.

A second parity-check matrix. All other experiments use one matrix $\mathbf{H}$ per rate, drawn with a fixed seed. To rule out that the placement gain is specific to that draw, we repeated the experiment with a matrix from a different seed at $n = 512$ and $R = 1.30$, where FER falls from 1.71% to 0.18%.

Better stream scoring. A *decoder-scored placement* ranks the streams by the decoder's $M = 128$ prediction vectors, which the encoder cannot compute, and so bounds what better encoder-side scoring could add. It matches the encoder-computable placement at $n = 4096$ and $R = 1.15$, and lowers FER from 0.26% to 0.03% at $n = 4096$ and $R = 1.12$, so better scoring helps mainly in the waterfall region.

Two simpler placement rules. Both were measured on 3,000 codewords at $n = 512$ and $R = 1.30$, where standard SW gives FER $= 1.60\%$. Sorting the hardest symbols onto the least vulnerable positions, those whose shortest cycle is longest, raises FER to 17.5%, 10.9 times the baseline, because those positions are individually safe but adjacent to one another. A uniformly random permutation gives 1.73%, within the sampling error of the baseline.

I Details of the Kodak experiment

This appendix gives the details of the Kodak experiment of Section 6.2. It describes how the images become streams, how the decoder's model is trained and how far it generalizes, and how its predictions behave at $n = 256$.

Stream construction. Each Kodak image (768×512, RGB) is cut into non-overlapping 16×16 patches, taken left to right and top to bottom, and each patch is flattened row by row with the three channel values of a pixel adjacent, as in Tian et al. (2026) and Chen et al. (2024). A patch is then 768 bytes, one byte per sub-pixel. The twelve test images are concatenated in order, cut into 512 segments of 27,648 bytes, 36 patches each, and read into 28,672 streams as described in Appendix D. The 28 chunks cover the first 24,544 bytes of each segment, and the remaining 3,104 bytes are not used. Because the context slides across patch boundaries, a prediction can draw on the previous patch as well as its own, and because streams are assigned to sets in segment order, the

streams of a set come from nearby patches, mostly of the same image. Eleven of the 512 segments straddle two images, so about 128 positions in each are predicted from the previous image's last patch, and the eleven newline separators between images are not pixels; these segments are included in every number reported.

Decoder model and its training. The decoder's model is BLT-7B (Pagnoni et al., 2024) with its patcher fixed to one byte per BLT patch, and a rank-64 LoRA adapter ($\alpha_{\mathrm{LoRA}} = 128$) on the query, key, value and output projections of all three transformer stacks, as in Chen et al. (2024). Training uses random 1024-byte crops of the twelve training images, 75% without a start token and 25% starting at an image-patch boundary with one, for 20,000 steps of 8 crops at learning rate 10^{-4} with cosine decay (164 million bytes, 19.5 hours on one A100). The checkpoint is chosen by held-out loss at positions with at least 128 bytes of context, the regime the decoder uses, and is reached at step 17,500.

Scope of the adapter. The adapter is trained and tested within one collection, twelve film scans seen for the equivalent of 11.6 epochs, with a final training loss of 1.84 bits per sub-pixel against 2.47 on the test images. Table 6 scores it on 48 DIV2K images it never saw, at native 2K resolution and resized to Kodak's width, with PNG as a model-free reference. It improves on PNG by 45% on Kodak but by only 8% on DIV2K at both resolutions, so most of what it learned is specific to Kodak, and the loss comes from content rather than resolution. For comparison, the DIV2K-trained adapter of Chen et al. (2024) reaches 2.83 on Kodak. Section 6.2 uses only the adapter's prediction vectors on the Kodak test images.

Table 6: Bits per sub-pixel of the pixel adapter outside its training collection. Kodak test: images 13 to 24. DIV2K: 48 random training images of DIV2K at 2K resolution, and the same images resized to 768 pixels wide. Adapter columns are the stream regime with at least 128 bytes of context; 32 random 1024-byte crops per image.

image set	PNG	BLT-7B, no adapter	adapter	adapter / PNG
Kodak test (12)	4.54	8.18	2.49	0.55
DIV2K, 2K (48)	3.99	8.52	3.68	0.92
DIV2K, 768 px wide (48)	4.49	9.05	4.15	0.92

Predictions at $n = 256$. The adapter is mildly overconfident. Over all codewords at $n = 256$ the mean log loss is 2.587 bits against a mean predicted entropy of 2.442, and on the 0.1% of codewords with the highest log loss the gap grows to 1.85 bits. The rate grid at $n = 256$ runs from 2.25 to 3.95 in steps of 0.05, and was extended to 8.0 because eleven codewords under standard SW decode at no rate. BP does not converge on them because each contains one to twenty confidently mispredicted symbols, at 12 to 22 bits of log loss, not because their mean log loss is high, and they are sent uncompressed. Adaptive placement separates those symbols on the code graph, and under it all 41,664 codewords decode at 5.95. Placement is encoder-computable on pixels as on text, since the stream entropies of (9) come from the same adapter at 16 bytes of context.

J TRAINING AND TRANSFER OF THE RATE PREDICTOR

The rate predictor of Section 5.1 is trained on the codewords of the corpus it is applied to. This appendix describes the two additional text corpora, how the predictor is trained for the learned columns of Table 2, and what happens when the predictor trained on text8 is applied to the other corpora without retraining. For the predictor, Gutenberg is a mild shift from text8, Python a larger one, and Kodak an extreme one.

Corpora. Gutenberg: 25 public-domain books, headers and licenses stripped, folded to ASCII, and joined at word boundaries into 512 segments of about 25,000 bytes. Code: the Python files of the CPython 3.11 source tree with the test directories excluded, newline-aligned into 512 segments, of which only the first 28 chunks are used. Appendix D describes how the streams are read.

Training on each corpus. For the learned columns of Table 2, the predictor is trained on the corpus it is applied to by cross-validation over sets. The sets of corpus codewords are split into k folds, with $k = 5$ to 7 at $n = 4096$ (one set per fold) and $k = 10$ at $n = 256$. For each fold, the predictor is trained on the other sets, with 20% of them held out to choose α^* and the fixed-rate reference, and is then applied to the held-out fold, so every codeword is predicted by a model that did not see it. Results are averaged over three seeds, with grid steps of 0.02 at $n = 4096$ and 0.05 at $n = 256$. On Python at $n = 256$ the predictor is weak, and its 0.10 bits/character saving arises because a few hard codewords set the fixed rate, not because its per-codeword predictions are accurate.

Measuring the shift. To measure how far a corpus is from text8 as the predictor sees it, we take, for each of the 31 features, the distance of the corpus's mean from the text8 mean in text8 standard deviations, and average the absolute values over the features. Table 7 reports this shift. It is small for Gutenberg, larger for Python and very large for Kodak.

Applying the text8 predictor without retraining. The three text8 predictors of Table 3, one per seed, are applied to the other corpora in two ways. *Zero-shot* keeps their weights, feature scaling and α^*. *Recalibrated* keeps the weights and scaling but chooses α^* again on part of the target corpus and applies it to the rest, rotating so that every codeword is scored once. At $n = 256$, $\alpha^* = 0.999$ gives a wide margin, and the predictor transfers to both text corpora with failure rates at the smallest level we can measure. At $n = 4096$ the margin is narrow ($\alpha^* = 0.94$ for adaptive rate alone and 0.91 with adaptive placement), and the feature shift raises the zero-shot failure rate on Gutenberg to 7.3%; choosing α^* again on Gutenberg brings it back to 0.04%. On Python the text8 predictor asks for 1.43 bits/character where 0.80 suffices, since it both ranks the codewords poorly ($\rho = 0.24$) and misjudges their scale, and only retraining corrects this. On Kodak every prediction lies above the rate grid.

Table 7: The rate predictor trained on text8, applied without retraining to each corpus with adaptive placement, in bits per coded symbol. Failure rates are before feedback. Shift is the feature shift defined above, and ρ is the Spearman rank correlation between predicted and measured $\bar{R}(\mathbf{x})$. The lossless column charges each codeword $\max(\text{predicted}, \bar{R}(\mathbf{x}))$ at the zero-shot operating point (Appendix F).

target	n	shift	fixed	zero-shot (FER)	recalibrated (FER)	lossless	ρ
text8	256	0.01	1.60	1.489 (10^{-5})	1.497 (2×10^{-5})	1.489	0.67
Gutenberg	256	0.67	1.65	1.508 (3×10^{-5})	1.560 (10^{-5})	1.508	0.68
Python	256	1.11	1.45	1.496 (0)	1.474 (3×10^{-5})	1.496	0.26
text8	4096	0.02	1.14	1.106 (0.22%)	1.124 (0.05%)	1.107	0.63
Gutenberg	4096	2.26	1.22	1.121 (7.3%)	1.188 (0.04%)	1.126	0.45
Python	4096	4.22	0.80	1.426 (0)	1.421 (0)	1.426	0.24
Kodak	4096	9×10^4	3.04	every prediction above the grid			0.22