ZIP2ZIP++: TOWARDS INFERENCE-TIME TOKEN COMPRESSION WITH MINIMAL QUALITY LOSS

Anonymous authors
Paper under double-blind review

ABSTRACT

Dynamic vocabularies built on-the-fly via Lempel–Ziv–Welch (LZW) compression let a language model represent the same text in fewer tokens, but prior work shows that this efficiency comes at a cost to reasoning and perplexity. We show that this quality gap is not a fundamental price of compression but can be substantially closed through targeted improvements to the compression layer and LZW codec, without modifying the underlying pretrained language model. Our analysis reveals four key issues: (1) LZW's compression algorithm creates an implicit conflict between the input embedding and the output embedding — a tension that a single shared embedding cannot resolve; (2) naively applying RoPE to LZW-compressed sequences ignores that merged tokens can span multiple tokens; (3) merging certain tokens (notably digits) disproportionately hurts downstream accuracy; and (4) dynamic vocabularies introduce an inherent segmentation ambiguity that confounds standard evaluation. We address these through decoupled input/output dynamic embedding layer, size-aware RoPE positioning, and a modified LZW codec supporting selective merges and multi-view segmentations. Across four models, our recipe improves perplexity on every corpus we evaluated as well as mathematical reasoning accuracy, closing roughly half the quality gap between zip2zip and the uncompressed baseline at a similar compression ratio on general text. This represents a significant step toward lossless adaptive-vocabulary compression for language models.

1 INTRODUCTION

The vocabulary of a language model determines the granularity at which text is represented, directly governing both computational efficiency and representational capacity. A more compact tokenization—one that represents the same text in fewer tokens—yields immediate practical benefits: lower inference cost (and, by extension, lower end-user billing in API-based services), reduced KV-cache memory footprint, and a larger effective context window within the same fixed token budget. Modern large language models universally adopt fixed vocabularies derived from statistical compression algorithms—most commonly Byte Pair Encoding (BPE) (Sennrich et al., 2016) or its variants (Kudo & Richardson, 2018; Kumar & Thawani, 2022; Liu et al., 2025)—trained once on large web corpora and frozen thereafter. While this approach is simple and effective on average, it is inherently suboptimal for any particular input: domain-specific terminology, low-resource languages, and structured data (e.g., code, mathematical expressions) are often over-segmented, producing unnecessarily long sequences and wasting compute (Rust et al., 2021; Ahia et al., 2023; Foroutan et al., 2026). Every excess token directly translates to unnecessary autoregressive steps, increased latency, and wasted context capacity. Prior attempts to mitigate this using vocabulary expansion (Wang et al., 2019), domain-adaptive retraining (Zhao et al., 2024), or efficient language-specific vocabulary expansion (Kim et al., 2024)—require per-domain intervention and do not generalize to new domains.

Dynamic vocabularies, which adapt tokenization to the input at inference time offer a principled alternative. Rather than committing to a single fixed segmentation, such methods construct input-specific tokens on the fly, achieving shorter sequences without retraining. One line of work removes the fixed vocabulary entirely by operating at the byte level (Yu et al., 2023; Slagle, 2024; Pagnoni et al., 2025; Hwang et al., 2025; Feher et al., 2025), learning to dynamically group bytes into variable-length patches through the model architecture itself. Another line of work (Liu et al., 2024b; Geng et al.,

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.

2025) builds on top of the existing BPE token vocabulary, constructing higher-order tokens on the fly from observed token sequences. Geng et al. (2025) recently repurposed the Lempel–Ziv–Welch (LZW) streaming compression codec (Welch, 1984) for language model inference, dynamically building a **hypertoken** vocabulary on the fly. Unlike BPE, which learns a fixed merge table offline from a static corpus, LZW expands its dictionary online from the input token stream, reducing token usage by up to 40%. However, a language model's vocabulary is tightly coupled with its architecture through the embedding and output projection layers, so any vocabulary change risks degrading model behavior. Geng et al. (2025) mitigate this via lightweight LoRA finetuning (Hu et al., 2022), but at a significant cost: perplexity increases substantially and downstream performance degrades sharply on some tasks.

In this work, we identify the key sources of quality loss in `zip2zip` and address them with four targeted modifications: (1) **Decoupled Residual Hypertoken Embeddings** with identity initialization (Sections 3.1 and 3.2); (2) **Size-Aware RoPE**, which positions a hypertoken by the span of base tokens it covers (Section 3.3); (3) **Fault-Tolerant LZW**, a codec that stays robust when the compressor is an imperfect neural model and lets fragile tokens be exempted from merging (Section 4); and (4) **Multi-View Perplexity**, which marginalizes over every segmentation that decodes to the target rather than only the canonical one (Section 4.1). Together, these substantially narrow the gap to uncompressed baselines.

2 Background: Zip2Zip

Tokenizer: LZW-based dynamic vocabulary. Rather than relying solely on a fixed base vocabulary $\mathcal{V}$ of size $|\mathcal{V}|$, `zip2zip` augments it with a dynamic *hyper-vocabulary* $\mathcal{V}_h$ constructed online via the Lempel–Ziv–Welch (LZW) compression algorithm. Given an input sequence of base tokens $\mathbf{x} = (x_1, x_2, \ldots, x_T)$ where each $x_i \in \mathcal{V}$, LZW incrementally identifies recurring subsequences $x_{1:K} := x_1 \ldots x_K$ with $K \leq M$, and merges them into *hypertokens* $z \in \mathcal{V}_h$, where M is the maximum merge size controlling the granularity of compression. We refer to $x_1, \ldots, x_K$ as the *constituent* base tokens of z, and to K as its *span length*. LZW operates in a streaming fashion: a hypertoken created at step t is immediately available for reuse at step $t+1$. The resulting compressed sequence $\mathbf{z} = (z_1, z_2, \ldots, z_{T'})$, where each $z_i \in \mathcal{V} \cup \mathcal{V}_h$, serves as the model's input and prediction target. At any point during decoding, the model operates over the combined vocabulary $\mathcal{V} \cup \mathcal{V}_h$, where $\mathcal{V}_h$ grows dynamically as new patterns are encountered.

Architecture: Dynamic Embedding. Since hypertokens are not part of the original vocabulary $\mathcal{V}$, they lack entries in the static embedding matrix $\mathbf{W}_{\text{in}} \in \mathbb{R}^{|\mathcal{V}| \times d}$ and the unembedding matrix $\mathbf{W}_{\text{out}} \in \mathbb{R}^{|\mathcal{V}| \times d}$. `zip2zip` introduces a lightweight *hyper-encoder*—a small transformer f_ϕ—that computes hypertoken embeddings on-the-fly from their constituent base token embeddings:

$$v(z) = f_\phi\big(\mathbf{W}[x_1],\ \mathbf{W}[x_2],\ \ldots,\ \mathbf{W}[x_K]\big), \tag{1}$$

where $v(z) \in \mathbb{R}^d$ is fed into the model in the same way as the base token embeddings. The output side is handled symmetrically: a hyper-encoder (optionally tied with the input one) maps rows of $\mathbf{W}_{\text{out}}$ to an unembedding vector $v_{\text{out}}(z)$, enabling the model to produce logits over the full union $\mathcal{V} \cup \mathcal{V}_h$ at each decoding step: the logit for emitting z at position j is $\mathbf{h}_j^\top v_{\text{out}}(z)$, where $\mathbf{h}_j \in \mathbb{R}^d$ is the decoder's hidden state at that position.

Training: Efficient Uptraining. An existing pretrained LLM can be uptrained to `zip2zip` via continued pretraining on LZW-compressed sequences using the standard causal language modeling objective over the compressed token space $\mathcal{V} \cup \mathcal{V}_h$. With parameter-efficient finetuning (LoRA; Hu et al., 2022), only the hyper-encoder f_ϕ and low-rank adapters $\Delta\theta$ are trained, requiring as few as 100M tokens.

3 Hypertoken Encoder Architecture

3.1 Prefix-Aligned Geometry of Hypertoken Embeddings

The key insight is that the LZW dictionary is *prefix-closed*: a hypertoken z with constituents $x_{1:K}$ enters $\mathcal{V}_h$ only after $x_{1:K-1}$ has already been registered, so $\mathcal{V}_h$ is precisely the node set of a trie over

base-token strings. This is the maximal violation of the prefix-free condition—every internal node is a prefix of some other entry—and hence $\mathcal{V} \cup \mathcal{V}_h$ is not a uniquely decipherable code: the map from segmentations to base-token strings is many-to-one (§4). Standard LZW appears deterministic only because its encoder imposes a greedy longest-match parsing rule; once the encoder is a sampling language model, that tie-break disappears and hypertokens sharing a prefix genuinely compete at the same decoding position, as illustrated in Figure 1(c). For the output hyper-encoder this means $v_{\text{out}}(x_{1:K})$ and $v_{\text{out}}(x_{1:K-1})$ are scored against one hidden state, which favours a strongly *prefix-aligned* geometry on f_ϕ^{out}. The input hyper-encoder f_ϕ^{in}, in contrast, faces no such structural constraint. Its role is to produce an embedding $v(z)$ that best serves to predict the next token—which, as we show empirically, leads to a *balanced* geometry that weighs every constituent of $x_{1:K}$ comparably. These are opposing objectives—one geometry is prefix-dominated, the other treats all positions alike—and we hypothesise that one shared encoder cannot serve both well.

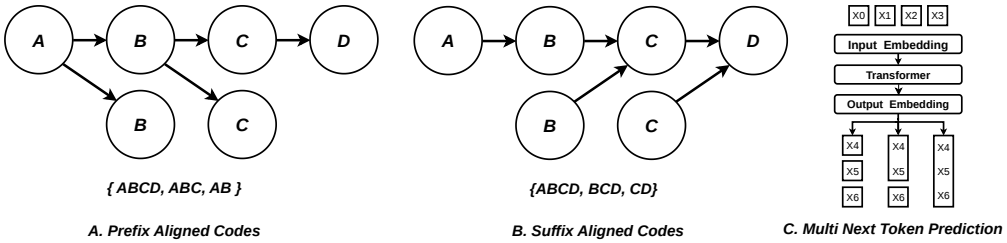


Figure 1: **Why LZW induces prefix-aligned output codes. (a)** The LZW dictionary is prefix-closed: an entry is registered only after its own prefix, so the entries available at a given position form a chain under the prefix order ($\{AB, ABC, ABCD\}$), all sharing the same first constituent. **(b)** The mirror image—a suffix-closed dictionary $\{CD, BCD, ABCD\}$—never arises under LZW. **(c)** Consequence for the language model: at a single decoding position the output hyper-encoder must score every member of such a chain against the same hidden state, which pushes their unembedding vectors toward one another and imposes a prefix-aligned geometry on f_ϕ^{out}.

Decoupling Input and Output Hypertoken Embeddings. In conventional transformer language models, tying the input embedding matrix $\mathbf{W}_{\text{in}}$ and the output unembedding matrix $\mathbf{W}_{\text{out}}$ is standard practice (Press & Wolf, 2017; Inan et al., 2017), particularly for smaller models. This reduces parameters without degrading performance, as both matrices operate over the same fixed vocabulary $\mathcal{V}$ and each token serves a single, static role. For `zip2zip` one might similarly wish to tie f_ϕ^{in} and f_ϕ^{out}. Due to the conflict above, however, a shared encoder would have to serve both geometries at once and ends up serving neither. We therefore untie the two hyper-encoders, at the cost of one additional lightweight encoder.

Experiments. We verify the above hypothesis by probing the learned geometry of the untied hyper-encoders (f_ϕ^{in} and f_ϕ^{out}) after training. ***Substitution probe.*** Take a hypertoken with constituents ABC and perturb exactly one end: replacing the first constituent gives KBC, replacing the last gives ABJ, where J and K are drawn uniformly from $\mathcal{V}$. We compute the input and output embeddings of the original and perturbed hypertokens, and measure the cosine similarity between them. We then obtain the ratio $r_K = \cos_{ABC,ABJ} \,/\, \cos_{ABC,KBC}$, which quantifies positional asymmetry: $r_K > 1$ indicates **prefix-aligned** representations, $r_K < 1$ **suffix-aligned**, and $r_K \approx 1$ a **balanced** representation that weighs both ends equally.

Observations. Three findings emerge from the substitution probe. **(1) The output encoder is prefix-aligned:** replacing the first constituent moves the embedding far more than replacing the last. **(2) The input encoder is balanced:** its substitution effects are similar at both ends. **(3) Neither pattern is present in the base BPE embeddings:** running the same experiments on the base BPE embeddings shows no comparable asymmetry, indicating that these patterns are learned from LZW compressed sequences rather than being general properties of BPE embeddings. As a second check we vary the *extent* of a span rather than perturbing it in place: given a span $ABCD$, we embed every contiguous sub-span and compare the nested prefix chain ($AB \subset ABC \subset ABCD$) against the sub-spans that drop A (we refer to this as *Sequential Probe*). The resulting similarity structure

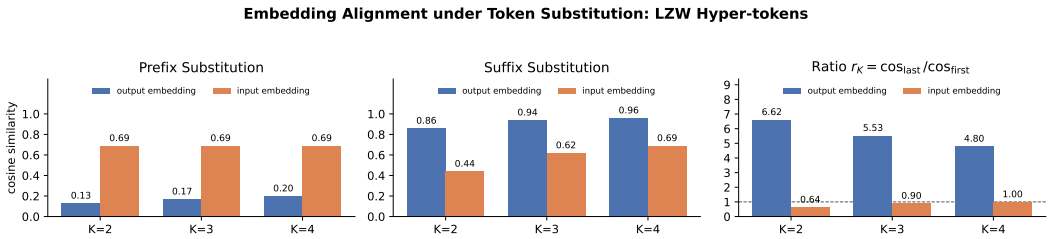
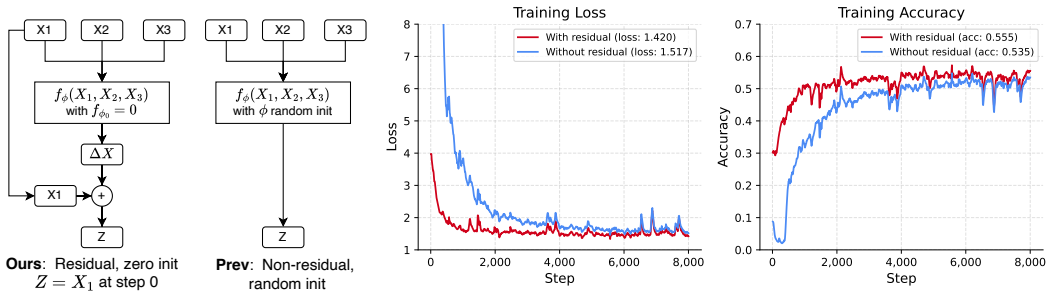


Figure 2: **Substitution probe on LZW hypertokens** (zip2zip++, Phi-3.5-mini). Bars report the mean cosine *similarity* between a hypertoken embedding and the same embedding after replacing its first (*Prefix Substitution*) or last (*Suffix Substitution*) constituent. The ratio $r_K = \cos_{\text{last}} / \cos_{\text{first}}$ (right) quantifies positional asymmetry. The output hyper-encoder is strongly prefix-dominated at every span length, whereas the input hyper-encoder is balanced. The corresponding BPE control, which shows no comparable asymmetry, is deferred to Figure 7.

(Figure 9) further supports the conclusion above—the output embedding clusters by shared prefix, whereas the input embedding tracks total constituent overlap. Details are deferred to Section B.

3.2 Residual Hyper-Encoder with Identity Initialization



(a) (b)

Figure 3: **Identity initialization of the hyper-encoder. (a)** With the residual, the encoder only predicts a correction ΔX on top of e_{x_1} and is zero-initialized ($f_{\phi_0} = 0$), so a hypertoken embedding starts out exactly at e_{x_1}. **(b)** Training dynamics: identity initialization removes the early loss shock and reaches both a better final loss and a better accuracy.

Problem: Hypertoken Embedding Scale Mismatch. In the original `zip2zip`, a hypertoken embedding is read straight off a randomly initialized encoder, $v(z) = f_\phi(e_{x_1}, \ldots, e_{x_K})$, where $e_{x_k} := \mathbf{W}_{\text{in}}[x_k]$. The scale mismatch this creates is severe. The encoder terminates in a LayerNorm, which rescales every coordinate to unit variance. Its output vectors therefore have norm $\approx\sqrt{d} \approx 55$, roughly $54\times$ the norm of a base token embedding. Every hypertoken therefore enters the decoder as a large out-of-distribution vector, and the resulting optimization shock is severe: step-10 loss of 70–100, off the scale of Figure 3b.

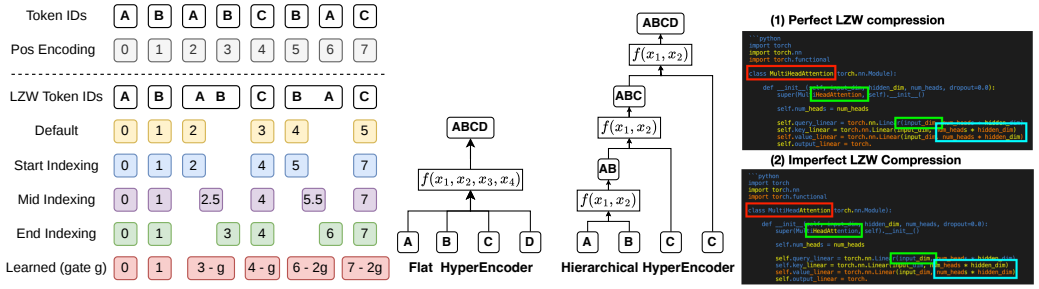
Residual Hyper-Encoder with Identity Initialization. We instead give the encoder a residual connection onto the embedding of the *first* constituent base token (Figure 3a):

$$v(z) = e_{x_1} + f_\phi(e_{x_1}, \ldots, e_{x_K}), \qquad f_{\phi_0} = 0, \tag{2}$$

where f_ϕ denotes the encoder function. We enforce $f_{\phi_0} = 0$ by setting the scale γ and bias β of the encoder's final LayerNorm to zero. This guarantees $v(z) = e_{x_1}$ exactly at the first step, so every hypertoken enters the decoder as a genuine pretrained token embedding and the encoder only has to learn how to *move away* from it. This is the trick LoRA uses—zero initializing B in $\Delta W = BA$ so that adaptation departs from the pretrained function rather than from noise—applied to the embedding table instead of the weights. As shown in Figure 3, this simple change has a dramatic effect on

training dynamics. It removes the shock above (step-10 loss 3.97, gradient norm ≈50) and yields a better final loss and accuracy.

3.3 SIZE-AWARE ROPE



(a) Size-aware RoPE (b) Hierarchical hyper-encoder (c) Imperfect compression

Figure 4: **(a)** Position assignment for a compressed sequence (Section 3.3). The base tokens $ABABCBAC$ occupy positions 0–7; LZW merges them into A, B, AB, C, BA, C. *Default* numbers the compressed tokens consecutively; *start*, *mid* and *end* indexing take the first, mean and last base position of each span and leave the other base tokens in place. The *learned* rule interpolates between end indexing and the default index with a gate g on the low-frequency rotary pairs ($g = 0$ is end indexing, $g = 1$ the default). **(b)** Two hyper-encoder variants (Section 3.4). *Flat* encodes all constituents tokens in a single pass and is tied to the training-time merge size; *hierarchical* folds them pairwise with a shared composition function. **(c)** Perfect versus imperfect LZW compression of the same code snippet (Section 4). Coloured boxes mark different hypertokens.

Merging tokens shortens the sequence, which raises the question of what RoPE position a hypertoken should receive. In the original implementation, Geng et al. (2025) simply number the compressed tokens $0, 1, 2, \ldots$, treating them the same as base tokens. One consequence is that the RoPE coordinates of every subsequent token are shifted by $K - 1$ for each merged span of length K, as shown in Figure 4a. Here we instead assign a hypertoken a position that accounts for its size, which we call *size-aware RoPE*. Since a hypertoken z with constituents $x_{1:K}$ occupies the span of base tokens $x_1, \ldots, x_K$, a natural choice is to reuse one of those base positions. We consider three variants—*start*, *mid* and *end* indexing—which take the first, mean and last base position of the span. As Figure 4a shows, the positions of the remaining base tokens are then unaffected by the merge: their absolute coordinates, and the relative distances between them, are the same as in the uncompressed sequence. This should reduce the distortion that compression introduces into the RoPE geometry. In addition, we also consider a *learned* rule that interpolates between the end-indexed position and the default compressed index, $p = p_{\text{end}} + g\,(p_{\text{comp}} - p_{\text{end}})$, with one gate g per layer and rotary frequency pair on the low-frequency pairs, initialised at zero (end indexing) and learned during finetuning.

Table 4 supports this: adding end indexing to the digit-protected, untied model raises GSM8K by 2.7 points at unchanged perplexity. The learned rule matches its perplexity but not its reasoning, so we keep the parameter-free end indexing (Section D.8).

3.4 HIERARCHICAL HYPER ENCODER

The hyper-encoder described so far is *flat*: it takes all K constituents at once, $f_\phi(e_{x_1}, \ldots, e_{x_K})$, padding shorter spans up to the maximum merge size M (Figure 4b, left). This gives direct access to the whole constituent sequence, but ties the encoder's learned positional structure to the M used during training, so a larger merge size at evaluation time is out of distribution.

We therefore also consider a *hierarchical* hyper-encoder, which builds the same embedding by repeatedly applying f_ϕ to *two* arguments (Figure 4b, right): for constituents $[A, B, C, D]$ it first composes A and B into $[A, B]$, then $[A, B]$ and C into $[A, B, C]$, and finally $[A, B, C]$ and D into $[A, B, C, D]$, each step reusing the same weights. Because the pairwise composer is shared across all depths, increasing the evaluation-time merge size adds recursion steps but no new parameters and no length-specific embeddings—the property we test in Section D.5.

4 A FAULT-TOLERANT LZW CODEC

The original zip2zip (Geng et al., 2025) adapted the classical LZW codec into token space, but the algorithm is the same. Yet, Language Models (LMs), which generate tokens autoregressively, have some distinct properties that make the original LZW algorithm (designed for text compression) brittle.

Multi-View Segmentation Problem. A fixed BPE vocabulary already admits many tokenizations of the same text and merely picks one by merge order, as shown in Figure 5, so scoring only that one understates a model's likelihood (Cao & Rimell, 2021)—though for BPE the correction is small, under 0.5% of log-likelihood in most cases (Chirkova et al., 2023). In `zip2zip` the LZW-induced vocabulary makes this *tokenization ambiguity* central: a single string can be segmented into hypertokens in many valid ways, all of which decompress to the same output (see Figure 4c for an example). We refer to the set of equivalent segmentations as the *multi-view* of a given string. Deterministic LZW hides this multiplicity behind a greedy longest-match rule, which selects one *canonical* segmentation; a language model trained in compressed space has no such guarantee. During training, the model is exposed to the canonical LZW segmentation, but at inference time it may emit any valid alternative. While the generated text remains correct upon decompression, this distributional spread poses a challenge for evaluation: standard metrics that condition on a single canonical segmentation will underestimate the model's true performance.

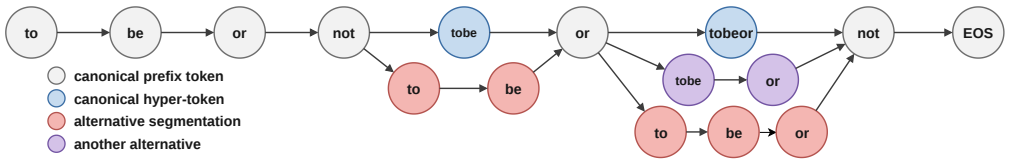


Figure 5: **Segmentation tree for multi-view scoring.** Segmentation tree for the text `to be or not to be or to be or not`. The canonical chain (grey and blue) carries the greedy LZW segmentation; all views of a position decode to the same base-token string.

Recovering from missed merges. Standard LZW decompression assumes a perfectly valid compressed sequence—a property that holds for deterministic compressors but breaks when the compressor is an autoregressive neural model, as demonstrated in Figure 4c. Any under-merging violates the one-to-one correspondence between received codes and novel dictionary entries, causing codebook desynchronization and cascading errors. The main change FT-LZW makes to the codec is to replace the codec's single unconditional insertion with a *guarded merge loop* (Algorithm 1). Upon decoding code c_t to sequence ω, the decoder walks along ω, extending the buffer W one symbol at a time: if $W \cdot y$ is already in the dictionary—the signature of a merge the model failed to make—the symbol is absorbed into W and the next symbol of ω is tried; an entry is inserted only when a genuinely novel combination is reached. On perfectly compressed input the guard never fires and FT-LZW recovers exactly the dictionary of standard LZW; on imperfect input it degrades gracefully to emitting base tokens rather than producing garbage.

Protecting fragile tokens. Standard LZW codecs blindly merge every pair of tokens they encounter, regardless of semantics. For LMs, however, certain tokens should not be merged. This applies to at least two cases: (1) special tokens used to delimit system, user, and assistant messages or signal EOS, and (2) tokens where merging is known to hurt performance, such as digits (Singh & Strouse, 2024). To handle this, we add a short-circuit to the codec's compression loop: whenever a token from the protected set is encountered, we emit it directly as a base token and skip the merge step.

The full codec—both the batch decoder and the online variant used during generation—is given in Section E, and we release it as an open-source Rust library.[1]

[1] URL anonymized for review.

4.1 MULTI-VIEW PERPLEXITY

To account for this ambiguity, we define a *multi-view* likelihood that aggregates over equivalent complete segmentations at the probability level. Concretely, for each target hypertoken z_i, let $\mathcal{S}(z_i)$ denote the set of all valid segmentations of $\mathrm{expand}(z_i)$—i.e., token sequences that decode to the same base-token string. The quantity sums the probability mass assigned to all equivalent segmentations:

$$p_{\text{multi-view}}(z_i \mid z_{<i}) = \sum_{s \in \mathcal{S}(z_i)} p(s \mid z_{<i}) = \sum_{s \in \mathcal{S}(z_i)} \prod_{j=1}^{|s|} p(s_j \mid z_{<i}, s_{<j}). \tag{3}$$

Intuitively, this captures the total probability the model assigns to *any* valid way of producing the same output text, rather than penalizing it for choosing a non-canonical but equally correct segmentation. For base tokens, $\mathcal{S}(z_i)$ reduces to the singleton $\{z_i\}$, and the multi-view probability coincides with the standard one. The multi-view perplexity replaces the token probability with this aggregate over the N canonical tokens (byte-level scores divide the same sum by the byte count of x):

$$\mathrm{PPL}_{\text{multi-view}}(x) = \exp\left(-\frac{1}{N}\sum_{i=1}^{N} \log\, p_{\text{multi-view}}(z_i \mid z_{<i})\right). \tag{4}$$

For a hypertoken expanding to n base tokens, the number of valid segmentations $|\mathcal{S}(z_i)|$ grows as $\mathcal{O}(2^n)$, but the merge cap M bounds it by 2^{M-1}, which stays small in practice. Using sequence packing and a tree-structured attention mask, we evaluate Eq. (3) exactly, given the canonical prefix, in a single additional forward pass at only $1.6\times$ the cost of standard perplexity (details in Section F).

5 EXPERIMENTS

Baselines. We compare three recipes that share one finetuning configuration. `Uncompressed` is the uncompressed control: the same base model, data, LoRA configuration and update budget, with compression disabled, so it isolates what our data mixture does on its own. `zip2zip` is the original recipe of Geng et al. (2025). `zip2zip++` adds the components of this paper on top of it: protected digits (Section 4), untied hyper-encoders (Section 3.1), size-aware RoPE (Section 3.3), the identity-initialised residual hyper-encoder (Section 3.2).

Models and data. We evaluate on four instruction-tuned decoders: Phi-3.5-mini-instruct (3.8B) and Phi-3-medium-4k-instruct (14B) in the main text, and Llama-3.2-1B/3B-Instruct in Table 7. All four are LoRA finetunes for 8,000 steps on the 1B-finetuning data released in Geng et al. (2025). The maximum merge size is $M{=}4$ throughout. Full hyperparameters are in Section C.

Metrics. We report three families. *Accuracy* on six two-shot multiple-choice benchmarks and, after finetuning, on GSM8K, MATH-500, HumanEval, IFEval and RULER; per-benchmark details are in Section C.1. *Byte-level perplexity* on four corpora, in two variants: *strict*, which scores only the canonical LZW segmentation of each target, and *multi-view* ($\star$), which sums the probability of every valid segmentation (Section 4.1); byte normalisation makes the two vocabularies comparable.

Table 1: **Main results.** Two-shot accuracy on six benchmarks and byte-level perplexity (↓) on four corpora with a 1024-token context; details in Section C.1. **Uncompressed** is the same base model finetuned on uncompressed sequences under the same conditions as in Section C. Bold marks the better of the two compressed models in each column. Results for Llama-3.2-1B/3B-Instruct are in Table 7.

Model	Method	ARC-c	ARC-e	HS	OBQA	PIQA	WG	Wiki	Pile	mC4	dC4
Phi-3.5-4B	Uncompressed	0.59	0.84	0.74	0.48	0.82	0.75	1.58	1.67	1.77	1.82
	zip2zip	0.54	0.81	0.71	0.46	**0.80**	**0.75**	1.70	1.82	1.86	1.92
	zip2zip++	**0.57**	**0.83**	**0.72**	**0.47**	**0.80**	0.74	**1.66**	**1.78**	**1.83**	**1.89**
Phi-3-14B	Uncompressed	0.65	0.88	0.81	0.51	0.83	0.79	1.43	1.61	1.71	1.76
	zip2zip	0.61	0.85	0.78	**0.51**	0.81	0.77	1.58	1.76	1.80	1.85
	zip2zip++	**0.62**	**0.86**	**0.79**	0.50	**0.82**	**0.78**	**1.53**	**1.72**	**1.77**	**1.82**

Table 2: Perplexity evaluation on four corpora using a 1024-token context window. ⋆ denotes the exact multi-view variants of Section 4.1, which sum the probability of every valid segmentation of the target. The uncompressed reference has a unique tokenization, so its multi-view scores coincide with its strict ones. PPL is per whitespace-delimited word, BPB per byte.

		Wiki				Pile				mC4				dC4			
Model	**Training**	PPL	PPL⋆	BPB	BPB⋆	PPL	PPL⋆	BPB	BPB⋆	PPL	PPL⋆	BPB	BPB⋆	PPL	PPL⋆	BPB	BPB⋆
Phi-3.5-4B	Uncompressed	11.5	11.5	0.66	0.66	31.5	31.5	0.74	0.74	30.4	30.4	0.82	0.82	35.2	35.2	0.87	0.87
	zip2zip	16.8	14.8	0.76	0.73	57.9	49.5	0.87	0.83	41.0	37.3	0.89	0.87	47.5	43.0	0.94	0.91
	zip2zip++	**14.9**	**13.6**	**0.73**	**0.70**	**49.3**	**43.4**	**0.83**	**0.81**	**37.8**	**35.0**	**0.87**	**0.85**	**44.0**	**40.7**	**0.92**	**0.90**
Phi-3-14B	Uncompressed	6.8	6.8	0.52	0.52	25.0	25.0	0.69	0.69	24.9	24.9	0.77	0.77	28.6	28.6	0.81	0.81
	zip2zip	11.6	9.9	0.66	0.62	45.4	38.7	0.82	0.78	33.6	30.5	0.84	0.82	38.5	34.9	0.89	0.86
	zip2zip++	**9.7**	**8.8**	**0.61**	**0.59**	**39.0**	**34.7**	**0.78**	**0.76**	**30.7**	**28.6**	**0.82**	**0.81**	**35.5**	**33.0**	**0.87**	**0.85**

Evaluation I: Perplexity and Few-Shot Accuracy. Table 1 reports six two-shot benchmarks together with strict byte-level perplexity, which scores only the canonical LZW segmentation of each target; Table 2 adds the multi-view score. **(1) `zip2zip++` improves on `zip2zip` across the board.** Perplexity is lower on all four corpora at both scales, under strict and multi-view scoring alike, and accuracy matches or beats `zip2zip` in most tasks. **(2) The gap to uncompressed is roughly halved, not closed.** Of the strict gap `zip2zip` leaves, the recipe closes about a third and multi-view scoring a further sixth to a third (a scoring artefact, not a modelling failure); the rest is unattributed, as the score conditions on the canonical prefix (Section F).

Table 3: **Post-SFT evaluation** on generation tasks and long context. All three recipes share the same data stream and update budget and differ only in the training representation and its window filter (Section D.7); per-benchmark details are in Section C.1 and strict-format variants in Section C.2. Bold marks the better of the two compressed models in each column; **Uncompressed** is the reference.

		GSM8K	**MATH-500**	**HumanEval**	**IFEval**	**RULER**			
Scale	**Model**	acc	acc	pass@1	prompt acc	4K	8K	16K	32K
Phi-3.5-4B	Uncompressed	0.75	0.36	0.58	0.38	86.4	81.5	78.1	73.7
	zip2zip	0.36	0.15	0.26	**0.32**	46.6	28.1	21.0	15.2
	zip2zip++	**0.68**	**0.24**	**0.32**	**0.32**	**70.7**	**60.6**	**53.2**	**52.8**
Phi-3-14B	Uncompressed	0.70	0.39	0.57	0.49	81.6	47.9	40.6	32.2
	zip2zip	0.51	0.22	**0.36**	**0.43**	52.5	23.2	14.0	11.6
	zip2zip++	**0.69**	**0.30**	0.31	0.40	**69.2**	**30.3**	**24.0**	**17.1**

Evaluation II: Downstream Tasks. Table 3 evaluates on tasks that require generation and reasoning. **(1) Mathematical reasoning improves the most.** GSM8K rises from 0.36 to 0.68 at 3.8B and from 0.51 to 0.69 at 14B, the latter within a point of `Uncompressed` (0.70). MATH-500 improves from 0.15 to 0.24 and from 0.22 to 0.30, but stays well below `Uncompressed` (0.36 and 0.39). **(2) Code generation and instruction following are not improved.** At 3.8B HumanEval moves from 0.26 to 0.32 and IFEval is unchanged at 0.32; at 14B `zip2zip++` is *worse* than `zip2zip` on both.

Evaluation III: Long-Context Understanding. The RULER columns of Table 3 probe retrieval at 4K–32K context. **(1) `zip2zip++` recovers a large share of what compression costs.** At 3.8B it raises 4K accuracy from 46.6 to 70.7 and 32K from 15.2 to 52.8, against 86.4 and 73.7 for `Uncompressed`; the same ordering holds at 14B. **(2) The absolute numbers are limited by the training length, not by compression alone.** None of the three recipes received long-context training: they were finetuned with 2,048-token sequences and evaluated with inference-time LongRoPE adaptation (Section C.1).

Evaluation IV: Ablation Study. Table 4 adds one component at a time, in the order the recipe was developed. **(1) Protecting digits dominates.** It alone moves GSM8K from 0.36 to 0.61; the three architectural changes add the remaining 7 points, with size-aware RoPE and the identity-initialised residual contributing $+2.7$ and $+2.8$. Untying the hyper-encoders acts mainly on the other metrics ($+1.6$ points of mean multiple-choice accuracy, -0.025 WikiText byte-perplexity). **(2) The end-to-**

Table 4: **Ablation study on Phi-3.5-mini.** Rows are cumulative, in the order in which the recipe was developed; the last row of the ladder is the zip2zip++ model of Table 1. Two-shot accuracy and byte-level perplexity ↓, as in Table 1. The continued-pretraining row is a single run after 200B additional tokens of compressed pretraining, not compute-matched to the others. Per-component gains and paired significance tests are in Section D.3. Bold marks the best value along the ladder; the rows below the rule are references.

Configuration	ARC-c	ARC-e	HS	OBQA	PIQA	WG	GSM8K	Wiki	Pile	mC4	dC4
zip2zip	0.54	0.81	0.71	0.46	**0.80**	**0.75**	0.36	1.70	1.82	1.86	1.92
+ Digit protection	0.54	0.81	0.70	**0.47**	0.79	0.71	0.61	1.71	1.83	1.87	1.93
+ Untied hyper-encoders	0.56	0.82	**0.72**	**0.47**	0.79	0.74	0.62	1.69	1.81	1.85	1.91
+ Size-aware RoPE	0.55	0.82	0.71	0.46	**0.80**	0.74	0.65	1.69	1.81	1.85	1.91
+ Identity-init residual	**0.57**	**0.83**	**0.72**	**0.47**	**0.80**	0.74	**0.68**	**1.66**	**1.78**	**1.83**	**1.89**
+ Continued Pretraining	0.55	0.82	0.72	0.46	0.80	0.75	0.64	1.63	1.74	1.82	1.88

end effect is significant; the individual rungs are less so. Paired significance tests against both `zip2zip` and `Uncompressed`, together with the seed spread, are reported in Section D.3. **(3) More compressed pretraining improves perplexity but not reasoning.** A further 200B tokens of compressed pretraining gives the best perplexity in the table (1.63 on WikiText) while GSM8K drops from 0.68 to 0.64—one more case where perplexity and reasoning move apart.

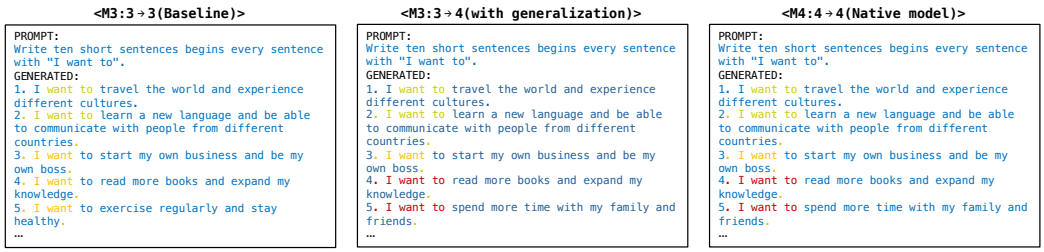


Figure 6: **Hypertoken length generalization examples.** Left: M_3 evaluated with its training-time maximum merge size (3→3, baseline). Center: the same M_3 model evaluated with the unseen merge size 4 (3→4, length generalization). Right: M_4, trained and evaluated with maximum merge size 4 (4→4, native reference). **Blue**: base tokens. **Yellow**: hypertokens (composed of 2 base tokens). **Orange**: hypertokens (composed of 3 base tokens). **Red**: hypertokens (composed of 4 base tokens).

Evaluation V: Hypertoken Length Generalization. Because the hierarchical encoder applies one shared pairwise composer at every recursion depth (Section 3.4), it can compose hypertokens longer than any it saw in training without adding parameters. We train with maximum merge size 3 and raise the cap to 4 at inference time. Figure 6 shows that the model does form and use the unseen length-4 merges, and quantitatively it tracks a model trained natively at size 4 closely: on WikiText the byte-perplexity gap is under 0.1%, and it stays within a few percent even on repeated-window text engineered to make length-4 merges frequent. Extrapolating further, to merge sizes $5, \ldots, 8$, the largest jump is at the training boundary and performance plateaus after it. Full numbers and the stress-test construction are in Section D.5.

6 Conclusion

In this work we investigate the performance loss incurred when a pretrained language model is upgraded from its static BPE vocabulary to an LZW-based dynamic one. We find that a series of architectural and algorithmic design choices in prior work are suboptimal and we propose an effective alternative for each. The resulting recipe strongly outperforms the original `zip2zip` on every perplexity corpus and at both model scales, and roughly halves the distance to an uncompressed model. Although the gap is not yet closed, this is a substantial step towards making adaptive tokenization quality-neutral.

AI USE STATEMENT

We used generative AI tools, both agentic assistants and code completion, for (1) programming, (2) paper writing and mock reviewing, (3) brainstorming and scientific discussion, and (4) literature review. We did not use them to run experiments or to produce any reported number. We have reviewed all AI-assisted work and take responsibility for the final content of this paper.

ETHICS STATEMENT

This work studies the token vocabulary of language models and introduces no new data collection, human subjects or model capabilities beyond those of the publicly released checkpoints it builds on. The datasets used are standard public corpora and benchmarks. One consequence of the method is worth stating: a shorter token sequence lowers inference cost, which makes the underlying models cheaper to deploy for both beneficial and harmful uses alike.

REPRODUCIBILITY STATEMENT

The finetuning configuration, data mixture and evaluation details needed to reproduce every number in this paper are given in Section C, with per-benchmark settings in Section C.1, the GSM8K answer-extraction rule in Section C.2, and the fixed perplexity subsets in Section C.3. Section D reports the full per-component results, the seed spread and the branches that did not work. Our Rust implementation of Fault-Tolerant LZW, the training code and the model checkpoints are publicly available.[2]

REFERENCES

Orevaoghene Ahia, Sachin Kumar, Hila Gonen, Jungo Kasai, David Mortensen, Noah Smith, and Yulia Tsvetkov. Do all languages cost the same? tokenization in the era of commercial language models. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 9904–9923, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.614. URL https://aclanthology.org/2023.emnlp-main.614/.

Kris Cao and Laura Rimell. You should evaluate your language model on marginal likelihood over tokenisations. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (eds.), *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 2104–2114, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.161. URL https://aclanthology.org/2021.emnlp-main.161/.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling, 2023. URL https://arxiv.org/abs/2302.01318.

Alexis Chevalier, Alexander Wettig, Anirudh Ajith, and Danqi Chen. Adapting language models to compress contexts. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 3829–3846, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.232. URL https://aclanthology.org/2023.emnlp-main.232/.

Nadezhda Chirkova, Germán Kruszewski, Jos Rozen, and Marc Dymetman. Should you marginalize over possible tokenizations? In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 1–12, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-short.1. URL https://aclanthology.org/2023.acl-short.1/.

[2]URL anonymized for review.

Darius Feher, Ivan Vulić, and Benjamin Minixhofer. Retrofitting large language models with dynamic tokenization. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 29866–29883, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.1444. URL https://aclanthology.org/2025.acl-long.1444/.

William Fleshman and Benjamin Van Durme. Toucan: Token-aware character level language modeling, 2023. URL https://arxiv.org/abs/2311.08620.

Negar Foroutan, Clara Meister, Debjit Paul, Joel Niklaus, Sina Ahmadi, Antoine Bosselut, and Rico Sennrich. Parity-aware byte-pair encoding: Improving cross-lingual fairness in tokenization. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7514–7538, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.342. URL https://aclanthology.org/2026.acl-long.342/.

Tao Ge, Hu Jing, Lei Wang, Xun Wang, Si-Qing Chen, and Furu Wei. In-context autoencoder for context compression in a large language model. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=uREj4ZuGJE.

Leonidas Gee, Leonardo Rigutini, Marco Ernandes, and Andrea Zugarini. Multi-word tokenization for sequence compression. In Mingxuan Wang and Imed Zitouni (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pp. 612–621, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-industry.58. URL https://aclanthology.org/2023.emnlp-industry.58/.

Saibo Geng, Nathan Ranchin, Yunzhen Yao, Maxime Peyrard, Chris Wendler, Michael Gastpar, and Robert West. zip2zip: Inference-time adaptive tokenization via online compression. In *Advances in Neural Information Processing Systems*, 2025. URL https://arxiv.org/abs/2506.01084.

Zébulon Goriely, Suchir Salhan, Pietro Lesci, Julius Cheng, and Paula Buttery. Bytespan: Information-driven subword tokenisation, 2025. URL https://arxiv.org/abs/2506.18639.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Sukjun Hwang, Brandon Wang, and Albert Gu. Dynamic chunking for end-to-end hierarchical sequence modeling, 2025. URL https://arxiv.org/abs/2507.07955.

Hakan Inan, Khashayar Khosravi, and Richard Socher. Tying word vectors and word classifiers: A loss framework for language modeling. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=r1aPbsFle.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. LLMLingua: Compressing prompts for accelerated inference of large language models. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 13358–13376, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.825. URL https://aclanthology.org/2023.emnlp-main.825.

Seungduk Kim, Seungtaek Choi, and Myeongho Jeong. Efficient and effective vocabulary expansion towards multilingual large language models, 2024. URL https://arxiv.org/abs/2402.14714.

Taku Kudo. Subword regularization: Improving neural network translation models with multiple subword candidates. In Iryna Gurevych and Yusuke Miyao (eds.), *Proceedings of the 56th*

Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp. 66–75, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1007. URL https://aclanthology.org/P18-1007/.

Taku Kudo and John Richardson. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 66–71, Brussels, Belgium, 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-2012. URL https://aclanthology.org/D18-2012/.

Dipesh Kumar and Avijit Thawani. BPE beyond word boundary: How NOT to use multi word expressions in neural machine translation. In Shabnam Tafreshi, João Sedoc, Anna Rogers, Aleksandr Drozd, Anna Rumshisky, and Arjun Akula (eds.), *Proceedings of the Third Workshop on Insights from Negative Results in NLP*, pp. 172–179, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.insights-1.24. URL https://aclanthology.org/2022.insights-1.24/.

Brian Lester, Jaehoon Lee, Alexander A Alemi, Jeffrey Pennington, Adam Roberts, Jascha Sohl-Dickstein, and Noah Constant. Training LLMs over neurally compressed text. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=pRvhMSV48t. Featured Certification.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding, 2023. URL https://arxiv.org/abs/2211.17192.

Yucheng Li, Bo Dong, Frank Guerin, and Chenghua Lin. Compressing context to enhance inference efficiency of large language models. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 6342–6353, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.391. URL https://aclanthology.org/2023.emnlp-main.391/.

Alisa Liu, Jonathan Hayase, Valentin Hofmann, Sewoong Oh, Noah A Smith, and Yejin Choi. SuperBPE: Space travel for language models. In *Second Conference on Language Modeling*, 2025. URL https://arxiv.org/abs/2503.13423.

Chengyuan Liu, Shihang Wang, Lizhi Qing, Kun Kuang, Yangyang Kang, Changlong Sun, and Fei Wu. Gold panning in vocabulary: An adaptive method for vocabulary expansion of domain-specific LLMs. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 7442–7459, Miami, Florida, USA, November 2024a. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.424. URL https://aclanthology.org/2024.emnlp-main.424/.

Siyang Liu, Naihao Deng, Sahand Sabour, Yilin Jia, Minlie Huang, and Rada Mihalcea. Task-adaptive tokenization: Enhancing long-form text generation efficacy in mental health and beyond. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 15264–15281, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.944. URL https://aclanthology.org/2023.emnlp-main.944/.

Yanting Liu, Tao Ji, Changzhi Sun, Yuanbin Wu, and Xiaoling Wang. Generation with dynamic vocabulary. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 18931–18948, Miami, Florida, USA, November 2024b. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.1053. URL https://aclanthology.org/2024.emnlp-main.1053/.

Benjamin Minixhofer, Edoardo M. Ponti, and Ivan Vulić. Zero-shot tokenizer transfer. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 46791–46818. Curran Associates, Inc., 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/532ce4fcf853023c4cf2ac38cbc5d002-Paper-Conference.pdf.

Jesse Mu, Xiang Li, and Noah Goodman. Learning to compress prompts with gist tokens. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 19327–19352. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/3d77c6dcc7f143aa2154e7f4d5e22d68-Paper-Conference.pdf.

Piotr Nawrot, Szymon Tworkowski, Michał Tyrolski, Lukasz Kaiser, Yuhuai Wu, Christian Szegedy, and Henryk Michalewski. Hierarchical transformers are more efficient language models. In Marine Carpuat, Marie-Catherine de Marneffe, and Ivan Vladimir Meza Ruiz (eds.), *Findings of the Association for Computational Linguistics: NAACL 2022*, pp. 1559–1571, Seattle, United States, July 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.findings-naacl.117. URL https://aclanthology.org/2022.findings-naacl.117/.

Piotr Nawrot, Jan Chorowski, Adrian Lancucki, and Edoardo Maria Ponti. Efficient transformers with dynamic token pooling. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6403–6417, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.353. URL https://aclanthology.org/2023.acl-long.353/.

Rodrigo Nogueira, Zhiying Jiang, and Jimmy Lin. Investigating the limitations of transformers with simple arithmetic tasks. In *ICLR 2021 Workshop on the Role of Mathematical Reasoning in General Artificial Intelligence*, 2021. URL https://arxiv.org/abs/2102.13019.

Artidoro Pagnoni, Ramakanth Pasunuru, Pedro Rodriguez, John Nguyen, Benjamin Muller, Margaret Li, Chunting Zhou, Lili Yu, Jason E Weston, Luke Zettlemoyer, Gargi Ghosh, Mike Lewis, Ari Holtzman, and Srini Iyer. Byte latent transformer: Patches scale better than tokens. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9238–9258, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.453. URL https://aclanthology.org/2025.acl-long.453/.

Ofir Press and Lior Wolf. Using the output embedding to improve language models. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pp. 157–163, Valencia, Spain, 2017. Association for Computational Linguistics. URL https://aclanthology.org/E17-2025/.

Ivan Provilkov, Dmitrii Emelianenko, and Elena Voita. BPE-dropout: Simple and effective subword regularization. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault (eds.), *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 1882–1892, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.170. URL https://aclanthology.org/2020.acl-main.170/.

Phillip Rust, Jonas Pfeiffer, Ivan Vulić, Sebastian Ruder, and Iryna Gurevych. How good is your tokenizer? on the monolingual performance of multilingual language models. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 3118–3135, Online, 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.243. URL https://aclanthology.org/2021.acl-long.243/.

Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In Katrin Erk and Noah A. Smith (eds.), *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1715–1725, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1162. URL https://aclanthology.org/P16-1162/.

Aaditya K. Singh and DJ Strouse. Tokenization counts: the impact of tokenization on arithmetic in frontier llms, 2024. URL https://arxiv.org/abs/2402.14903.

Kevin Slagle. Spacebyte: Towards deleting tokenization from large language modeling. In *Proceedings of the 38th Conference on Neural Information Processing Systems (NeurIPS*

2024), 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/e1f418450107c4a0ddc16d008d131573-Paper-Conference.pdf.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Bo Wen, and Yunfeng Liu. RoFormer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024. doi: 10.1016/j.neucom.2023.127063. URL https://doi.org/10.1016/j.neucom.2023.127063.

Avijit Thawani, Saurabh Ghanekar, Xiaoyuan Zhu, and Jay Pujara. Learn your tokens: Word-pooled tokenization for language modeling. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 9883–9893, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-emnlp.662. URL https://aclanthology.org/2023.findings-emnlp.662/.

Hai Wang, Dian Yu, Kai Sun, Jianshu Chen, and Dong Yu. Improving pre-trained multilingual model with vocabulary expansion. In Mohit Bansal and Aline Villavicencio (eds.), *Proceedings of the 23rd Conference on Computational Natural Language Learning (CoNLL)*, pp. 316–327, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/K19-1030. URL https://aclanthology.org/K19-1030/.

Shumin Wang, Yuexiang Xie, Bolin Ding, Jinyang Gao, and Yanyong Zhang. Language adaptation of large language models: An empirical study on LLaMA2. In Owen Rambow, Leo Wanner, Marianna Apidianaki, Hend Al-Khalifa, Barbara Di Eugenio, and Steven Schockaert (eds.), *Proceedings of the 31st International Conference on Computational Linguistics*, pp. 7195–7208, Abu Dhabi, UAE, January 2025. Association for Computational Linguistics. URL https://aclanthology.org/2025.coling-main.480/.

Welch. A technique for high-performance data compression. *Computer*, 17(6):8–19, 1984. doi: 10.1109/MC.1984.1659158.

David Wingate, Mohammad Shoeybi, and Taylor Sorensen. Prompt compression and contrastive conditioning for controllability and toxicity reduction in language models. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2022*, pp. 5621–5634, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.findings-emnlp.412. URL https://aclanthology.org/2022.findings-emnlp.412/.

Lili Yu, Daniel Simig, Colin Flaherty, Armen Aghajanyan, Luke Zettlemoyer, and Mike Lewis. Megabyte: Predicting million-byte sequences with multiscale transformers. In *Proceedings of the 37th Conference on Neural Information Processing Systems (NeurIPS 2023)*, 2023. URL https://neurips.cc/virtual/2023/poster/71722.

Jun Zhao, Zhihao Zhang, Luhui Gao, Qi Zhang, Tao Gui, and Xuanjing Huang. Llama beyond english: An empirical study on language capability transfer, 2024. URL https://arxiv.org/abs/2401.01055.

A Related Work

Adapting a fixed vocabulary. The most direct response to over-segmentation is to change the vocabulary offline. One line adds tokens for a target language or domain and continues pretraining: Chinese (Zhao et al., 2024), Korean (Kim et al., 2024), and specialised domains such as mental health (Liu et al., 2023; 2024a), with follow-up work studying which tokens are worth adding (Wang et al., 2025; 2019). A second line improves the vocabulary-learning algorithm itself, relaxing BPE's word-boundary constraint (Gee et al., 2023; Liu et al., 2025) or choosing merges by an information criterion (Goriely et al., 2025), which yields more compact tokenization than BPE (Sennrich et al., 2016; Kudo & Richardson, 2018; Kumar & Thawani, 2022). Both lines produce a vocabulary that is fixed once training ends, so they require per-domain intervention and cannot exploit repetition inside a particular input.

Compressing the input. Prompt-compression methods shorten the context by discarding or summarising it: gist tokens (Mu et al., 2023), selective context (Li et al., 2023), LLMLingua (Jiang et al., 2023), summary vectors (Chevalier et al., 2023), in-context autoencoders (Ge et al., 2024) and soft-prompt compression (Wingate et al., 2022). These achieve high compression ratios precisely because they are *lossy*, and they compress only the input: generated tokens—which dominate latency at low batch size—are unaffected. Closest in spirit to compressed language modelling is Lester et al. (2024), who train language models directly on text compressed with arithmetic coding, and find that the resulting codes are hard to model. LZW differs in that its codes stay aligned with the original token boundaries, which is what allows an existing model to be adapted to them rather than retrained from scratch.

Dynamic and byte-level architectures. A parallel line removes the fixed vocabulary from the architecture instead of the tokenizer, learning where to segment as part of the model: Hourglass and dynamic pooling (Nawrot et al., 2022; 2023), MegaByte (Yu et al., 2023), SpaceByte (Slagle, 2024), BLT (Pagnoni et al., 2025), H-Net (Hwang et al., 2025), Toucan (Fleshman & Durme, 2023), Learn-Your-Token (Thawani et al., 2023), ZeTT (Minixhofer et al., 2024) and retrofitted dynamic tokenization (Feher et al., 2025). These require training the segmentation policy jointly with the model, and offer no lossless codec that maps back to a canonical token stream. Dynamic Vocab (Liu et al., 2024b) shares our motivation most closely—it too grows the vocabulary during generation—and differs in how entries are created and how the model is trained. `zip2zip` (Geng et al., 2025) is the setting we build on: it keeps the base BPE vocabulary, adds LZW-derived hypertokens on top of it, and adapts a pretrained model with LoRA (Hu et al., 2022).

The interfaces a compressed stream disturbs. Our three architectural findings each touch a well-studied component. Tying the input and output embedding matrices is standard for small models (Press & Wolf, 2017; Inan et al., 2017); we show it fails for a prefix-closed dynamic vocabulary, where the two matrices are pushed toward incompatible geometries. Rotary position embeddings (Su et al., 2024) assume one position per token, which a variable-span hypertoken violates. And the fragility of digit tokenization for arithmetic is documented (Nogueira et al., 2021; Singh & Strouse, 2024); we find it is the single largest source of degradation under compression, and that exempting digits from merging recovers most of it.

Tokenization ambiguity and marginal likelihood. That a text has many valid tokenizations, and that scoring only the canonical one understates a model's likelihood, is known for static vocabularies. Cao & Rimell (2021) argue that language models should therefore be evaluated on the marginal likelihood over tokenizations, while subword regularization (Kudo, 2018) and BPE-dropout (Provilkov et al., 2020) exploit the same multiplicity on the training side, sampling among segmentations for robustness. For a fixed BPE vocabulary, however, the effect turns out to be small: Chirkova et al. (2023) estimate the marginal probability by importance sampling and find the log-likelihood gap is under 0.5% in most cases. Our setting is qualitatively different. A prefix-closed LZW dictionary is the maximal violation of the prefix-free condition, so the number of valid segmentations grows exponentially in the span length instead of comprising a few alternatives, and the model is free to emit any of them. Our multi-view perplexity (Section 4.1) marginalises over that set exactly, in one packed forward pass rather than by sampling, and finds a gap of a wholly different order—about a third of the apparent perplexity deficit of a compressed model (29–36% of the strict deficit of `zip2zip++` against its uncompressed control, Section D.4) is a scoring artefact rather than a modelling failure.

Other routes to fewer forward passes. Speculative decoding (Leviathan et al., 2023; Chen et al., 2023) also emits several tokens per target-model pass, but leaves the vocabulary and the output distribution untouched: it trades extra compute for latency and is exact by construction. Adaptive tokenization is complementary—it shortens the sequence itself, which reduces KV-cache and context consumption as well as decoding steps, at the cost of having to preserve model quality across a changed vocabulary.

B Embeddings Probes

Substitution Details. We sample the top 5,000 hypertokens (by corpus frequency) for each span length $K \in \{2, 3, 4\}$ from WikiText-2, excluding spans containing special or digit tokens. As a

control, we run identical diagnostics directly on the raw embedding matrices $\mathbf{W}_{\text{in}}$ and $\mathbf{W}_{\text{out}}$ (no encoder) to verify that any observed structure is created by the hyper-encoders rather than inherited from the pretrained tables. Figure 7 reports this control. Unlike the LZW hypertoken embeddings of Figure 2, the raw BPE tables are close to positionally symmetric: r_K stays within 0.71–1.33 for both matrices at every K, against 4.8–6.6 for the output hyper-encoder. The mild residual drift below 1 at larger K reflects nested real words sharing more suffix than prefix mass, not a compositional strategy. The prefix-aligned and balanced geometries reported in Section 3.1 are therefore properties the hyper-encoders acquire during training, not artifacts of the pretrained embedding space.

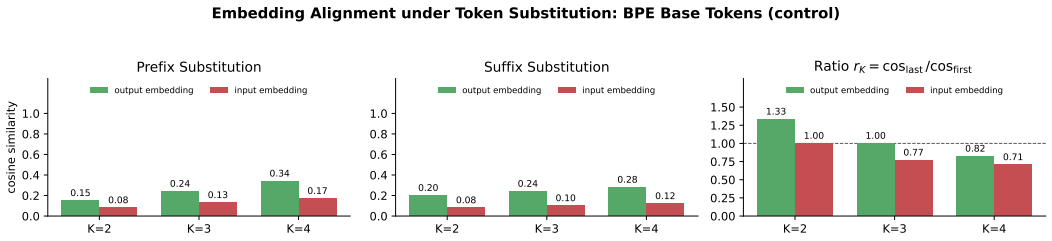


Figure 7: **BPE base-table control for the substitution probe** (zip2zip++, Phi-3.5-mini). The identical probe applied directly to the pretrained $\mathbf{W}_{\text{in}}$ and $\mathbf{W}_{\text{out}}$, with no hyper-encoder in the path. Panels and metrics match Figure 2. Both tables stay within $r_K \in [0.71, 1.33]$ at every K, i.e. no asymmetry comparable to the 4.8–6.6 of the output hyper-encoder.

Sequence probe. We complement the aggregate substitution probe with an illustrative span-level analysis. For the 4-token span *"It is a dog"*—written $ABCD$ below—we embed every contiguous sub-span of length ≥ 2 and compute all pairwise cosine similarities (Figure 9). This adjudicates whether similarity follows a shared prefix or broader constituent overlap: a prefix-aligned encoder should cluster the nested chain $AB \subset ABC \subset ABCD$ tightly regardless of what follows, whereas a balanced encoder should instead track how many constituents two spans have in common.

Both predictions hold. In the output embedding the prefix chain forms a block at $\cos \geq 0.88$, while every pair spanning the two blocks falls to 0.21–0.44. In the input embedding there is no such block structure but a smooth gradient: $ABCD$ sits closer to BCD (0.71), with which it shares three constituents, than to its own prefix AB (0.48), with which it shares two.

C Experimental Setup

Table 5 lists the configuration, shared by every run in this paper and by the uncompressed control. The control sets the codebook size to zero, so compression becomes the exact identity, the hyper-encoder path is not executed, and the objective reduces to ordinary next-token prediction. It is *update-matched* but not base-token- or trainable-parameter-matched: at equal decoder positions the control sees ≈262M base tokens against ≈375M for the compressed recipes, which additionally train two hyper-encoders. We report compression as $C = N_{\text{base}}/N_{\text{compressed}} \geq 1$.

Implementation. All three are implemented behind one interface in a single training and evaluation stack, so they share the data loader, the LoRA setup and the evaluation harness; any difference between them is attributable to the training representation and to the window filter it implies (Section D.7), not to the pipeline.

C.1 Benchmarks and evaluation details

Multiple-choice tasks. ARC-Challenge, ARC-Easy, HellaSwag, OpenBookQA, PIQA and WinoGrande are evaluated two-shot through the chat template, with the few-shot examples supplied as previous turns. We report length-normalised accuracy for all of them except WinoGrande, where normalisation does not apply. Bootstrapped standard errors are at most 0.022 on OpenBookQA and below 0.015 on the other five tasks.

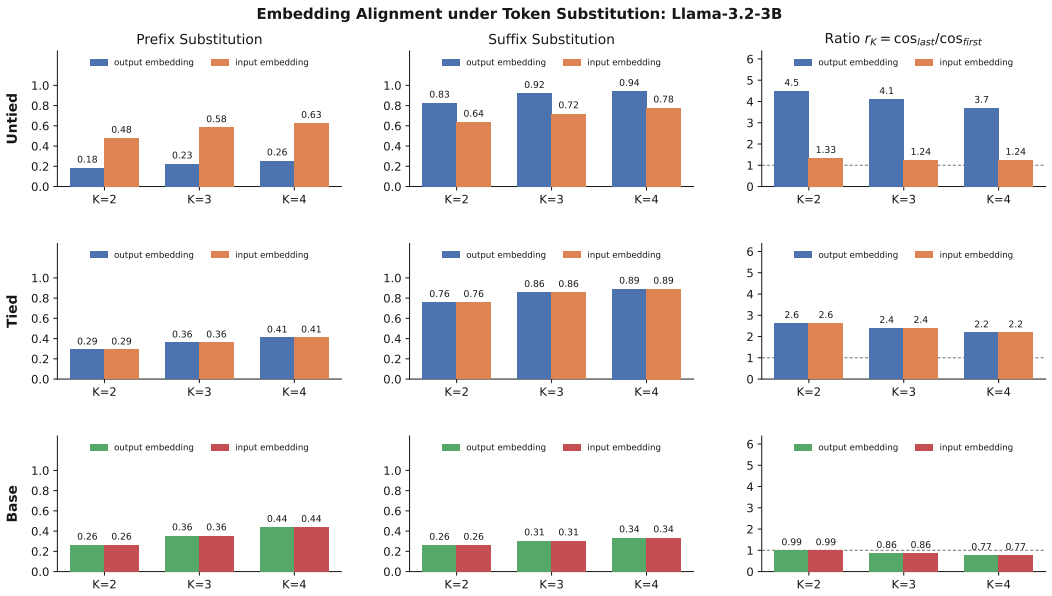


Figure 8: **Substitution probe on Llama-3.2-3B hypertokens.** The untied output hyper-encoder is strongly prefix-aligned, whereas the untied input hyper-encoder exhibits substantially weaker positional asymmetry. When the two hyper-encoders are tied, their input and output ratios coincide by construction and consistently fall between the corresponding untied output and input ratios. The base-token embeddings remain close to or below $r_K = 1$, showing no comparable prefix-aligned asymmetry.

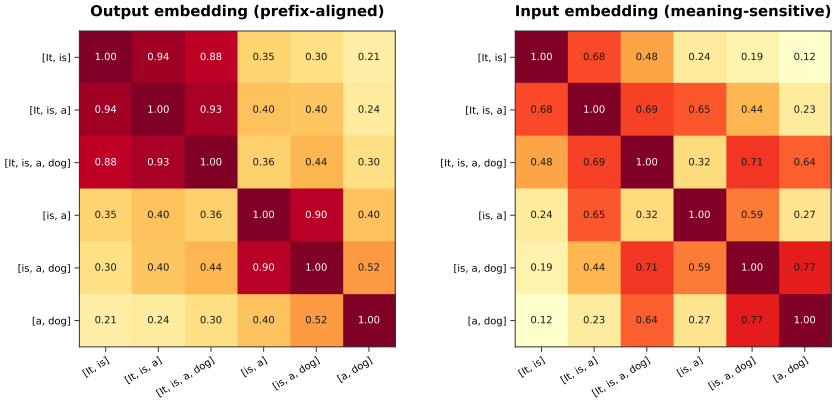


Figure 9: **Sequence probe: pairwise similarity of contiguous sub-spans** (zip2zip++, Phi-3.5-mini). Cosine similarity between the embeddings of all contiguous sub-spans of length ≥ 2 of *"It is a dog"*. *Left:* the output hyper-encoder shows clear block structure—spans sharing a prefix cluster at $\cos \geq 0.88$ while cross-block pairs fall to 0.21–0.44. *Right:* the input hyper-encoder shows no such blocks but a smooth gradient in the number of shared constituents, consistent with the balanced geometry of Section 3.1.

Generation tasks. GSM8K is two-shot, capped at 256 emitted tokens (a hypertoken counts once, so a compressed model may decode up to C_{gen} times more base tokens within the cap), with answers extracted as in Section C.2. MATH-500 uses the four-shot Minerva prompt and answers are checked symbolically with `math_verify`. HumanEval is zero-shot greedy pass@1, with generation stopped at the closing code fence. IFEval is zero-shot prompt-level strict accuracy.

Table 5: Finetuning configuration, shared across all runs and the control.

Base models	Phi-3.5-mini-instruct (3.8B), Phi-3-medium-4k-instruct (14B), Llama-3.2-1B-Instruct, Llama-3.2-3B-Instruct
Adapter	LoRA rank 32, $\alpha = 32$, all attention and MLP projections; decoder otherwise frozen
Optimizer	AdamW, $\beta = (0.9, 0.999)$, weight decay 0.01
Learning rate	cosine $3 \times 10^{-4} \to 10^{-5}$, 800 warm-up steps
Steps	8,000, at 32,768 decoder positions per step
Window	2,048 tokens
Seed	42
Max merge size M	4
Codebook	4,096 entries; a 2,048-token window creates at most 2,048
Protected set	special and added tokens, plus every vocabulary piece that is a run of ASCII digits (10 pieces for Phi, 1,110 for Llama-3.2, whose pre-tokenizer splits numbers into groups of up to three digits)
Data	$\sim$1B-token instruction mixture (multi-turn chat, competition mathematics with worked solutions, web and code text), re-tokenized per model: 997M tokens (Phi), 846M (Llama), eight shards. Chat documents use the model's own template, loss on assistant turns only.
Evaluation	two-shot through the chat template, lm-evaluation-harness 0.4.12, seed 1234, length-normalized accuracy where applicable; byte-level perplexity over disjoint windows of 1,024 base tokens, each compressed independently with a fresh codebook
Hardware	4$\times$H100 80GB; 1.5 h (Llama-1B), 2.5 h (Llama-3B, Phi-3.5-mini), 6.5 h (Phi-3-medium), plus evaluation
Hyper-encoders	two flat pre-norm transformer layers over the $\leq M$ constituent embeddings with a learned slot embedding; width = decoder width, 32 heads (40 for Phi-3-medium), $4\times$ feed-forward; non-causal, mean-pooled, added to the first constituent's embedding

Long context. RULER is evaluated at 4K, 8K, 16K and 32K context on a fixed subset of 200 examples per task. Because the post-SFT checkpoints were trained with 2,048-token sequences and received no long-context training, RULER uses inference-time LongRoPE adaptation with the official 128K configuration of the corresponding Phi model family; no LongRoPE finetuning was performed.

C.2 Answer extraction for GSM8K and MATH-500

Strict-format metrics require the answer template verbatim (GSM8K `####`, the Minerva *Final Answer* sentence). Compressed models rarely reproduce these templates even when their answers are correct: on Phi-3.5-mini, GSM8K strict-match is .37/.04/.22 and MATH-500 exact-match .12/.01/.01 for Uncompressed/`zip2zip`/`zip2zip++`, against .75/.36/.68 and .36/.15/.24 under flexible extraction. We therefore report flexible metrics throughout and treat the strict gap as a formatting effect. Template adherence is itself a facet of instruction following, and the IFEval drop of the compressed models (Table 3) points the same way; the flexible rule isolates arithmetic correctness from this effect rather than denying it.

The flexible rule has a symmetric failure mode: the uncompressed 14B baseline answers correctly and then, instead of stopping, continues with an unrelated new problem in the style of its finetuning data. This happens in 275 of 1,319 generations and defeats a plain last-number heuristic, which scores that model .49 where the strict rule gives .66. Every GSM8K number in the main text and in Tables 6, 8 and 12 therefore uses one rule: the answer is the number following "####" when the model emits that template, and the last number of the generation otherwise. Relative to plain flexible extraction this moves the 14B baseline from .49 to .70 and no other row by more than half a point. The evaluation-only diagnostics of Table 9 use plain flexible extraction, as their generations were not stored; on the checkpoints where both rules are available they differ by at most 0.5 points.

Table 6: GSM8K under the two extraction rules along the ablation ladder of Table 4. The flexible column uses the combined rule defined above. Strict matching demands the #### template verbatim; it moves by 14 points between the first two rungs, whose combined-rule scores differ by 2.7, which is why we report the combined rule throughout.

Recipe	GSM8K flexible	GSM8K strict
Protected digits, untied encoders	.6232	.066
+ end-indexed positions	.6497	.210
+ identity init.	.6778	.215
Uncompressed reference	.7468	.372

C.3 Perplexity corpora and the fixed subsets

WikiText is the full WikiText-2 (raw) test set (62 documents, the lm-evaluation-harness `wikitext` task); Pile is the validation split of The Pile; mC4 and dC4 are the `en` and `en.noblocklist` validation splits of C4, the latter without bad-word filtering. The Pile, mC4 and dC4 validation sets are too large to score for every checkpoint at 1,024-token context, so all rows use the same fixed subset of 1,000 documents per corpus, drawn once with seed 1234 and pinned by content hash; scores are reproducible to 10^{-4} (bf16 jitter $\approx 10^{-5}$). On the four Phi-3.5-mini rows the subset values differ from the full-corpus values by at most 0.04 byte-perplexity and never change the ordering between rows.

D More Experimental Results

Every table in this section uses the setup of Section C. Where a compressed-trained checkpoint is evaluated twice, once with compression enabled and once disabled, we decompose the gap to the uncompressed control as

$$S_{\mathrm{ctrl}} - S_{\mathrm{comp}} = \underbrace{S_{\mathrm{ctrl}} - S_{\mathrm{base}}}_{\text{weight-adaptation cost}} + \underbrace{S_{\mathrm{base}} - S_{\mathrm{comp}}}_{\text{representation cost}}, \tag{5}$$

where S_{base} is the score of the compressed-trained weights with compression disabled. This separates damage stored in the adapted weights from damage caused by presenting those weights with a compressed sequence.

D.1 Main Results, All Models

Table 7 extends Table 1 to all four base models and adds the **Original** row: the released instruct checkpoint with its own tokenizer, evaluated without any finetuning. Comparing **Original** with **Uncompressed** isolates what our finetuning mixture does on its own, before compression enters: on the multiple-choice tasks it helps Phi-3.5-mini and Llama-3.2-1B slightly, is roughly neutral on Phi-3-medium, and costs Llama-3.2-3B a few points on ARC-Challenge; on GSM8K it costs every model 3–14 points (Table 8), most on Phi-3-medium. The compressed rows should therefore be read against **Uncompressed** rather than against the released model.

Table 8: GSM8K accuracy (two-shot, combined extraction rule of Section C.2) and generation compression C_{gen} (base tokens per emitted token over the generated answers) for all four models. **Original** uses flexible extraction; the two rules give identical scores on the released Llama models. Bold marks the better of the two compressed models.

	GSM8K accuracy↑				C_{gen} ↑	
	Original	Uncompressed	zip2zip	zip2zip++	zip2zip	zip2zip++
Phi-3.5-mini	.78	.75	.36	**.68**	1.53	1.25
Phi-3-medium	.84	.70	.51	**.69**	1.52	1.27
Llama-3.2-1B-Inst	.42	.34	.20	**.24**	1.37	1.26
Llama-3.2-3B-Inst	.76	.64	.41	**.53**	1.37	1.29

Table 7: **Main results, all four base models.** Columns and conventions as in Table 1. **Original** is the released instruct checkpoint with no finetuning; **Uncompressed** is the same model finetuned on uncompressed sequences. Bold marks the better of the two compressed models in each column.

Model	Method	ARC-c	ARC-e	HS	OBQA	PIQA	WG	Wiki	Pile	mC4	dC4
Phi-3.5-4B	Original	0.56	0.79	0.71	0.48	0.78	0.73	1.58	1.68	1.79	1.84
	Uncompressed	0.59	0.84	0.74	0.48	0.82	0.75	1.58	1.67	1.77	1.82
	zip2zip	0.54	0.81	0.71	0.46	**0.80**	**0.75**	1.70	1.82	1.86	1.92
	zip2zip++	**0.57**	**0.83**	**0.72**	**0.47**	**0.80**	0.74	**1.66**	**1.78**	**1.83**	**1.89**
Phi-3-14B	Original	0.64	0.86	0.82	0.57	0.83	0.79	1.43	1.62	1.72	1.77
	Uncompressed	0.65	0.88	0.81	0.51	0.83	0.79	1.43	1.61	1.71	1.76
	zip2zip	0.61	0.85	0.78	**0.51**	0.81	0.77	1.58	1.76	1.80	1.85
	zip2zip++	**0.62**	**0.86**	**0.79**	0.50	**0.82**	**0.78**	**1.53**	**1.72**	**1.77**	**1.82**
Llama-3.2-1B-Inst	Original	0.39	0.63	0.58	0.35	0.74	0.58	1.74	1.79	1.88	1.94
	Uncompressed	0.39	0.66	0.59	0.35	0.75	0.60	1.71	1.77	1.84	1.89
	zip2zip	**0.38**	**0.64**	**0.58**	**0.37**	**0.74**	**0.61**	1.78	1.87	1.90	1.96
	zip2zip++	0.36	**0.64**	**0.58**	0.37	**0.74**	**0.61**	**1.77**	**1.85**	**1.89**	**1.95**
Llama-3.2-3B-Inst	Original	0.51	0.77	0.69	0.43	0.77	0.65	1.65	1.71	1.78	1.84
	Uncompressed	0.47	0.77	0.69	0.43	0.78	0.68	1.63	1.68	1.74	1.80
	zip2zip	**0.48**	**0.76**	0.67	**0.43**	**0.76**	**0.68**	1.72	1.81	1.83	1.89
	zip2zip++	**0.48**	**0.76**	**0.68**	**0.43**	**0.76**	**0.68**	**1.68**	**1.75**	**1.79**	**1.84**

D.2 Protected Tokens

FT-LZW's protected-token set doubles as a diagnostic: by exempting one class of tokens at a time we can ask *which* tokens compression actually harms. Table 9 does this for digits, operators and LaTeX control tokens. Digits dominate, and adding the other two classes buys nothing further, so effective protection has to be targeted rather than broad: compressibility is not semantic substitutability.

Achieved compression. The compression `zip2zip++` actually realises is uneven across workloads: it represents WikiText in 14% fewer tokens ($1.17\times$) and its own GSM8K generations in 20% fewer ($1.25\times$), while the short two-shot multiple-choice prompts shrink by only 7%—these tasks are both where compression helps least and where the accuracy differences between recipes are smallest. Part of the gap to the unprotected recipe, which compresses WikiText by $1.21\times$, is the price of protecting digits, quantified in the last three columns of Table 9. The same pattern holds on GSM8K generations for every model (Table 8, last two columns): `zip2zip` compresses answers by 1.37–$1.53\times$ and `zip2zip++` by 1.25–$1.29\times$; the Llama models, whose BPE vocabulary already merges up to three digits, gain the least from LZW on arithmetic text.

Table 9: **Token protection with FT-LZW.** Rows sharing a checkpoint differ only in the evaluation representation, so the block structure instantiates Equation (5): protecting digits at evaluation alone recovers 10.2 of the 12.7-point representation penalty, and protecting them during training as well repairs both terms, a 25.4-point gain over the unprotected baseline. All compressed rows in this paper use the FT-LZW codec and differ only in its protected-token set. The cost is localized to math-heavy generation; general-text and multiple-choice compression are barely affected. Protecting operators and LaTeX control tokens in addition yields $+0.5$ points, confirming that protection should target empirically fragile atoms rather than be applied broadly.

	GSM8K↑	Compression C ↑		
Protection strategy	flexible	WikiText	MC input	Math decode
No protection	.356	1.209	1.079	1.53
same weights, digits protected at eval	.458	1.169	1.077	1.30
same weights, uncompressed eval	.483	1.000	1.000	1.00
Digits protected (train + eval)	.610	1.169	1.077	1.28
same weights, uncompressed eval	.679	1.000	1.000	1.00
+ operators and LaTeX (eval only)	.615	1.166	1.077	1.23
Uncompressed control	.742	1.000	1.000	1.00

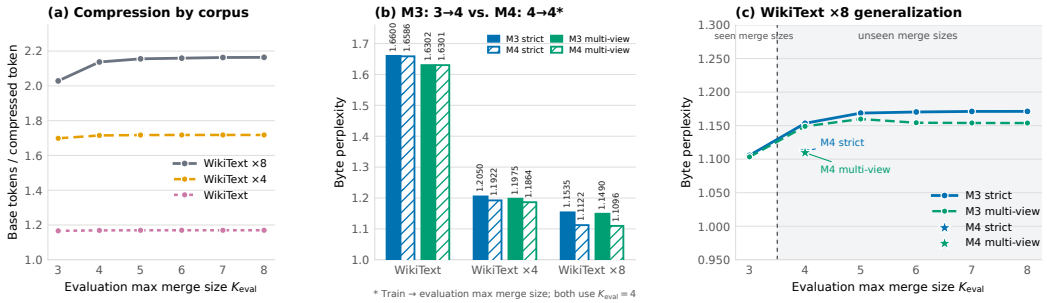


Figure 10: **Hierarchical merge-size generalization. (a)** Compression as the evaluation-time maximum merge size increases; WikiText ×4 and ×8 provide progressively stronger transfer exposure. **(b)** Strict and exact multi-view byte perplexity for transferred $M_3 : 3{\to}4$ and native $M_4 : 4{\to}4$. **(c)** M_3, trained with $K = 3$, evaluated at $K_{\text{eval}} = 3, \ldots, 8$ on WikiText ×8, with native M_4 at $K_{\text{eval}} = 4$ for reference.

D.3 Per-Component Gains and Significance

Along the ladder of Table 4, digit protection accounts for most of the GSM8K recovery (+25.4 points), and end-indexed positions and the zero-initialized residual for most of the rest (+2.7 and +2.8 points). Untying the hyper-encoders acts mainly on the other metrics (+1.6 points of mean multiple-choice accuracy, −0.025 WikiText byte-perplexity, on unrounded scores). Under an exact paired McNemar test on the 1,319 GSM8K problems, `zip2zip++` is 32.2 points above the unmodified `zip2zip` recipe ($p = 1 \times 10^{-83}$) and 6.9 points below the update-matched `Uncompressed` reference ($p = 3 \times 10^{-7}$): both the improvement and the residual gap are real rather than noise. A second training seed of the final recipe lands 1.8 points below the first ($p = 0.14$), the same order as every individual step after digit protection. We therefore read the ladder as a cumulative effect that is significant end to end, rather than as a sequence of individually significant steps. On the Llama models the same test gives `zip2zip++` vs. `zip2zip` +4.7 points (p=1.5×10^{-4}) at 1B and +12.0 (p=1.3×10^{-17}) at 3B, and −9.3 / −11.6 points vs. `Uncompressed` (p<10^{-11}).

D.4 Multi-View Scoring

Section 4.1 defines multi-view perplexity as the exact sum over all complete segmentations. Some of our earlier evaluations instead used a *first-token bound*, which credits the base token x_1 plus every available hypertoken whose expansion is a nonempty prefix of the remaining target:

$$p^*_{\text{ft}}(x_{1:n} \mid \mathbf{z}_{<i}) = p(x_1 \mid \mathbf{z}_{<i}) + \sum_{z:\, \text{expand}(z) \preceq x_{1:n}} p(z \mid \mathbf{z}_{<i}). \tag{6}$$

Since the strict target is included, $p^*_{\text{ft}} \geq p_{\text{multi-view}} \geq p_{\text{strict}}$. We report both because the difference is large: exact marginalisation removes 29–36% of the canonical penalty, whereas the bound appears to remove 74–97%. Segmentation ambiguity is therefore a real but minority component of the perplexity deficit, and we no longer use the bound as evidence.

Table 10: Byte-perplexity of `zip2zip++` (Phi-3.5-mini) against the update-matched control, on WikiText and the fixed 1,000-document Pile/mC4/dC4 subsets of Section C.3.

Corpus	Control	Strict	Exact multi-view	First-token bound
WikiText	1.5777	1.6573	1.6290	1.5882
Pile	1.6678	1.7819	1.7488	1.6972
mC4	1.7660	1.8311	1.8084	1.7696
dC4	1.8221	1.8913	1.8668	1.8240

D.5 MERGE-SIZE TRANSFER

Natural text rarely exercises a larger merge-size cap: raising it from 3 to 4 changes compression by 0.018% on our default evaluation and 0.22% on WikiText. We therefore stress-test on repeated-window WikiText, where length-4 spans cover 6.54% or 29.90% of scored base tokens; absolute perplexities on these synthetic corpora are not comparable to natural WikiText. Under exact multi-view scoring the transfer penalty is only 12% and 5% smaller than the strict penalty, so the cross-size cost is a genuine limitation of the size-3 encoder rather than a canonical-segmentation artefact — the first-token bound would have hidden most of it.

Table 11: Merge-size transfer. $a \to b$ denotes training with maximum merge size a and evaluating with size b. Penalties are for $3 \to 4$ relative to the native $4 \to 4$ model. Single runs.

	Length-4	Strict byte-PPL			Penalty of $3 \to 4$		
Corpus	coverage	$3 \to 3$	$3 \to 4$	$4 \to 4$	Strict	Exact MV	First-token
Repeat-4	6.54%	1.1890	1.2050	1.1922	1.07%	.94%	.25%
Repeat-8	29.90%	1.1057	1.1535	1.1122	3.71%	3.54%	.33%

D.6 DATA SCALE

A raw-text pretraining stage before SFT improves language modeling but degrades instruction-tuned reasoning, which longer SFT then partially restores while surrendering most of the perplexity advantage (Table 12). Extending SFT alone from 8k to 32k steps, for the digit-protected untied model (third rung of Table 4, before end-indexed positions and zero-initialisation), raises GSM8K from .623 to .672 and improves WikiText byte-perplexity from 1.686 to 1.656—roughly what the full recipe reaches at 8k steps with a quarter of the compute— but the control was not extended, so this is not an equal-compute result. The pretraining mixture, used at 20B and 200B tokens (50% FineWeb-Edu, 20% Stack, 10% FineMath, 20% FineWeb2-HQ), compresses by only $1.3100\times$ despite its code component, while the templated SFT mixture reaches $1.4504\times$: repeated dialogue templates and mathematical markup matter more than semantic domain for online LZW compressibility.

Table 12: A raw-text pretraining stage before the standard SFT mixture. Single runs, without a matched long-run uncompressed control. The 200B row is the "Continued Pretraining" row of Table 4.

Training path	GSM8K	Wiki byte-PPL
zip2zip++, direct SFT, 8k steps	.6778	1.6574
20B raw + SFT, 8k	.6088	1.6377
20B raw + SFT, 16k	.6255	1.6390
20B raw + SFT, 48k	.6391	1.6554
200B raw + SFT, 8k	.6414	1.6331

D.7 SEED VARIANCE AND COMPARABILITY

Seed variance. Two runs of the final recipe give .6778 and .6596 GSM8K accuracy (combined rule), disagreeing on 240 of 1319 items (132 vs. 108, p=0.14), while their multiple-choice averages differ by .12 points and their final losses by 10^{-4}. Strict accuracy moves in the opposite direction ($+4.09$ points for the seed with lower flexible accuracy), confirming that strict matching entangles reasoning with output format. Single-seed GSM8K differences below roughly two points should therefore not be read as effects.

Comparability. Several factors limit which numbers may be compared directly. The control is update-matched rather than base-token- or parameter-matched. Every run uses the same data loader, which discards the $\approx 6.5\%$ of training windows that compress by more than $2\times$; the uncompressed control, whose ratio is 1, keeps every window. Training the final recipe on the complete stream instead lowers GSM8K by 8.2 points (.677→.595, paired McNemar p=10^{-8}) and worsens perplexity

and generation compression, so the filter is kept and reported here as a data choice rather than a bug fix. A control trained on the same retained windows was not run, so the filter is a confound shared by every compressed-versus-uncompressed comparison; and because the retained set depends on the codec configuration (protected digits lower the ratio), the rungs of Table 4 need not train on identical windows. The released Llama models are scored through their own chat template, which inserts the evaluation date into the system prompt; re-running GSM8K on a later date moved their scores by 1.3–1.7 points, whereas our finetuned models pin the date and re-run to within 0.9 points. We report achieved compression rather than wall-clock decoding speed; the hyper-encoders and the codec add per-step costs that we have not benchmarked. Finally, perplexity on Pile, mC4 and dC4 should be compared only within a fixed sampling pipeline.

D.8 Negative Results

What the recipe does not fix. The gains are concentrated in perplexity and mathematical reasoning. On HumanEval and IFEval, `zip2zip++` is statistically indistinguishable from the unmodified `zip2zip` recipe at both scales (paired McNemar $p = 0.16$ and $p = 1$ on Phi-3.5-mini, $p = 0.32$ and $p = 0.16$ on Phi-3-medium): compression costs 25–26 points of code generation and 6–9 points of instruction following, and no statistically established part of it is recovered.

Branches that did not help. Table 13 lists the one-variable branches we tried. Several lower the training loss or the canonical perplexity while lowering GSM8K, which is why we report reasoning accuracy, decoded-text probability and achieved compression as separate axes rather than relying on perplexity alone. Two rows need a caveat. By default, compressed position t may predict the base vocabulary and the codebook entries with creation index $k \leq t$ in its window; since LZW creates at most one entry per consumed token, this is a superset of the entries that exist at t, so some later-created entries enter the softmax denominator early, while the target itself always exists. The same mask is used for training and for every reported perplexity, strict and multi-view alike; generation masks to the entries present in the decoder's codebook, which is exact by construction. Because the default set is a superset of the entries the decoder can emit, restricting the softmax to the exact set can only raise the target probability: for the same logits, the reported perplexities of the compressed models are upper bounds on those of the deployed decoder. The online-mask run instead exposes at each position exactly the entries created earlier in its window. It improves every training-side proxy (teacher-forced next-token accuracy $49.41 \rightarrow 50.15\%$, hypertoken accuracy $.420 \rightarrow .504$, generation compression $1.2536 \rightarrow 1.3107\times$) yet worsens six of seven downstream tasks. The two positional branches (two-axis RoPE puts even rotary pairs on base-stream positions and odd pairs on compressed ones; the gated variant learns the mix per layer) were still improving when the learning-rate budget ran out, so they show that neither variant helps at 8k steps, not that compressed coordinates are intrinsically harmful.

Table 13: One-variable branches that did not improve the recipe. ΔGSM is in percentage points relative to each branch's parent (the `zip2zip` baseline, the "+ untied" rung of Table 4, or the final recipe). "Flat" means no practically meaningful change. All are single runs; see Section D.7 for seed spread.

Intervention	Parent	Training / LM proxy	ΔGSM	Takeaway
Reformat math SFT as chat	`zip2zip`	Wiki PPL $1.695 \rightarrow 1.719$	$+2.3$	Narrow style gain, broad MC/PPL trade-off
Freeze decoder LoRA for 200 steps	+ untied	Final loss flat	-6.6	Co-adaptation is preferable here
Hyper-encoder 2→4 layers	+ untied	Loss $1.51 \rightarrow 1.45$; PPL flat	-2.6	Capacity fits training, does not transfer
Exact online codebook mask	final	Loss $1.434 \rightarrow 1.376$	-2.4	Better proxies, worse downstream
25% uncompressed microsteps	final	PPL flat	-3.6	No measurable transfer
Two-axis RoPE (fixed 50/50 split)	final	Wiki PPL flat	-2.4	Mixed coordinates do not help
Gated RoPE (512 low-freq. gates)	final	**Best** Wiki PPL 1.6565	-3.3	Best perplexity, worst reasoning
Tie the two hyper-encoders	final	MC/PPL near flat	-9.1	Role-specific composition matters most
Remove the residual connection	final	Wiki PPL $1.657 \rightarrow 1.705$	-3.9	The pretrained anchor is important
Hierarchical instead of flat encoder	final	Wiki PPL flat; slower	-4.6	No gain at fixed M; kept for length transfer (Section D.5)

E Fault-Tolerant LZW: Details

Where standard LZW breaks. Algorithm 1 isolates the single line that differs between the two decoders. Standard LZW (red) inserts `buffer + z` unconditionally, because a deterministic

encoder guarantees that this combination is always new: every code it emits creates exactly one dictionary entry. A neural compressor gives no such guarantee—when it under-merges, `buffer + z` may already be present, and the unconditional insert then writes a duplicate, desynchronising the decoder's dictionary from the encoder's for the rest of the sequence. FT-LZW (blue) replaces that line with a *guarded merge*: it walks along ω, absorbing symbols whose combination with the buffer is already in the dictionary, and inserts only when it reaches a genuinely novel one. On perfectly compressed input the loop exits on its first iteration and the two decoders agree exactly; on imperfectly compressed input FT-LZW skips the redundant insert instead of corrupting the dictionary. Three further mechanisms—a disabled-token set, unknown-code resolution and a codebook cap—are listed below.

Algorithm 1 LZW decompression: standard (red) vs. FT-LZW (blue).

Initialize dictionary with the base tokens (e.g. `to`, `be`, `or`, `not`) $\rightarrow$ `dict[0..3]`
`next_id` $\leftarrow 4$
Read first code X; **emit** `dict[`X`]`; `buffer` $\leftarrow$ `dict[`X`]`
while more codes **do**
    Read next code Z
    $\omega \leftarrow$ `dict[`Z`]`; **emit** ω

    $z \leftarrow$ first symbol of ω ▷ standard LZW: one unconditional insert
    `dict[next_id]` $\leftarrow$ `buffer` $+ z$; `next_id` $\leftarrow$ `next_id` $+ 1$
    `buffer` $\leftarrow \omega$

    while $\omega \neq \varepsilon$ **do** ▷ FT-LZW: guarded merge
        $z \leftarrow$ first symbol of ω
        if `buffer` $+ z \notin$ `dict` **then**
            `dict[next_id]` $\leftarrow$ `buffer` $+ z$; `next_id` $\leftarrow$ `next_id` $+ 1$
            `buffer` $\leftarrow \omega$; **break** ▷ novel entry
        end if
        `buffer` $\leftarrow$ `buffer` $+ z$; $\omega \leftarrow \omega$ without its first symbol ▷ already known: absorb
    end while
    end while

A worked example. Figure 11 traces the same token stream through both cases. Under canonical compression every code creates exactly one new codebook entry, so the decoder stays in its NORMAL state throughout. Under non-canonical compression the guarded loop of Algorithm 1 takes its other branches: it SKIPs the insertion when the candidate entry already exists, RESUMEs on the following code to create the entry that was deferred, and takes an EARLY EXIT when it reaches a novel combination partway through ω. In every case the emitted base-token stream is identical to the input, which is what keeps decompression lossless even when the model under-merges.

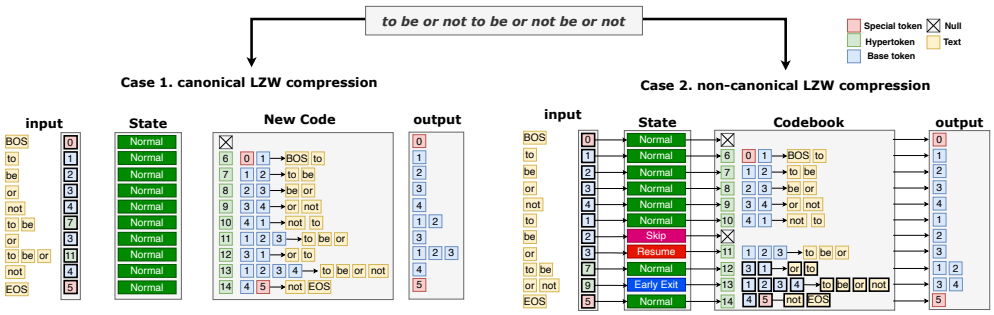


Figure 11: **FT-LZW decoding, step by step.** The same stream *"to be or not to be or to be or not"* under canonical (left) and non-canonical (right) compression. Columns show the input codes, the decoder state, the codebook entry created at that step, and the emitted base tokens. Canonical input keeps the decoder in NORMAL at every step, one code creating one entry. Non-canonical input triggers the guard: SKIP (no entry, the candidate already exists), RESUME (the deferred entry is created on the next code), and EARLY EXIT (a novel combination is reached partway through the expansion). Both cases emit the same base-token sequence.

- **Disabled tokens.** A designated set of token IDs (e.g. `<pad>`, `<eos>`, and the protected tokens of Section 4) is emitted verbatim and resets the buffer, so these tokens are never absorbed into a composite entry.
- **Unknown-code resolution.** When a received code is not yet in the dictionary, two cases arise. If it equals the next free index, the decoder infers $\omega = W \cdot \mathrm{head}(W)$—the classic LZW "cScSc" special case—and inserts and emits it immediately. If instead the buffer has reached the maximum entry length M, the unknown code must denote the buffer itself, force-emitted by the encoder on reaching the merge bound, so $\omega \leftarrow W$.
- **Codebook cap.** Dictionary growth stops once N new entries exist; the buffer resets and decoding continues without further growth, bounding the memory the codebook can consume over a long context.

Online variant for autoregressive generation. During generation the model emits one (hyper-) token at a time, so the dictionary must be updated *after each token* for newly created hypertokens to be available at the next step. The online variant keeps a persistent state (W, D, j) across steps and runs exactly the guarded loop of Algorithm 1 on each generated token; it differs from the batch decoder only in that it receives one code per call and emits nothing, since the model has already produced the token. Applying it token by token to a code sequence therefore yields the same dictionary as decoding that sequence in one pass, which is what keeps the encoder's and decoder's codebooks synchronized even when sampling produces suboptimal compression.

F TREE ATTENTION FOR MULTI-VIEW SCORING

The canonical sequence forms a backbone under ordinary causal attention. The distinct segmentation prefixes of each target are packed immediately after the corresponding root, forming a tree. The tree-attention mask allows each branch node to attend only to the canonical context preceding its target and to its own ancestors; canonical nodes cannot see branches, and sibling or cross-target branches remain isolated. Root probabilities are reused from the canonical forward pass, while the packed pass computes all remaining conditional probabilities. This matches independent prefix scoring up to numerical precision. On WikiText, tree attention is $26.0\times$ faster than the independent-prefix implementation and brings the total evaluation cost to only $1.6\times$ that of standard perplexity.

Every view is scored conditioned on the canonical prefix $z_{<i}$; alternative segmentations of the prefix itself are not enumerated. The score is therefore exact for the target given the canonical context and lies between the strict score and the first-token bound (Section D.4), but it is not the marginal over all segmentations of the whole text. The two coincide only if the model's predictions do not depend on how the context was segmented; in general the multi-view score can lie on either side of that

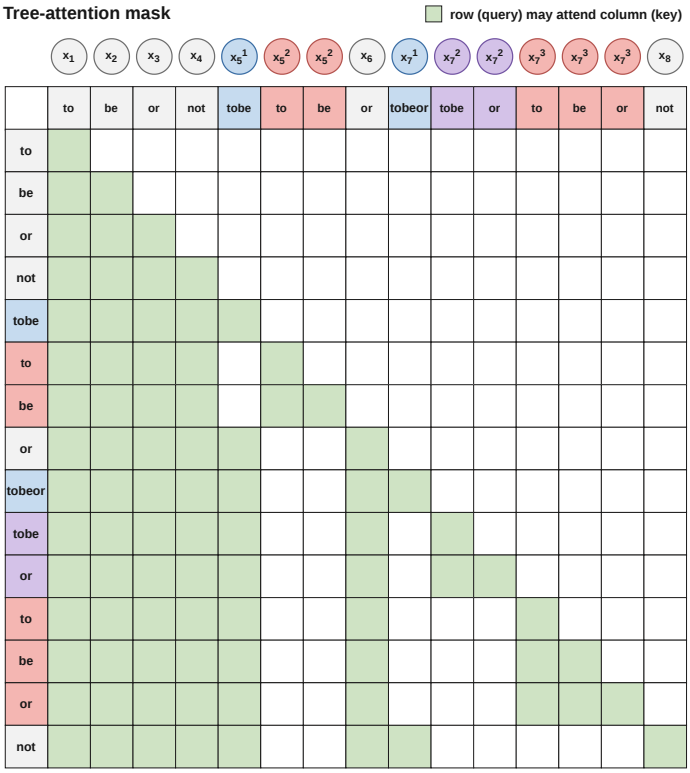


Figure 12: **Tree attention for multi-view scoring.** The corresponding attention mask over the packed sequence. A shaded cell means the row (query) may read the column (key). The canonical backbone is causal; each branch sees only the shared prefix and its own ancestors, so sibling views and views of different targets cannot contaminate one another. All views are scored in one forward pass.

marginal, so the deficit that remains under it is a canonical-context measurement, not a bound on the modelling cost of the compressed interface.