Token Folding: Shortening Language-Model Sequences by Fusing Predictable Tokens into Content Words

Anonymous

Under review

Abstract

When using a standard tokenizer, LLM sequences carry many low-entropy tokens such as function words, punctuation, capitalization, common suffixes, etc, which we call modifier tokens whose identity in context is often low entropy, and that serve to modify the function of the "base" word in the sentence. For example, if "the buildings" was 3 tokens, "[the] [building] [s]", the main word would be [building], while [the] and [s] are modifiers. We introduce *token folding*: a preprocessing step that fuses such modifier tokens into an adjacent content token, so that a single sequence position now represents the "base" word with its modifiers. The folded modifiers are predicted by a small auxiliary head, while the full transformer capacity is only spent on the "base" word prediction. We test folded and unfolded models from 71M to 1B parameters against compute-matched no-fold twins. Folding reduces sequence length by 14.2% for a small but statistically significant cost: +0.0064 bits per character at 356M, with a 95% interval of $[+0.0056, +0.0073]$ over three seeds. The shorter sequence requires 14.2% fewer autoregressive forward passes, 14.2% less KV cache per unit of text, and 26.4% fewer pairwise attention interactions. At 16K context, this translates to 24.6% higher measured prefill throughput, or 17.4% after charging our current unoptimised prompt-folding implementation, with peak memory at 0.87×. The quality cost remains nearly unchanged from the compute-optimal training budget to 2.6× that budget and decreases across our 163M to 1B scale sweep. Predictable surface-form tokens can therefore be moved out of the main autoregressive sequence at a small quality cost, reducing both sequential and attention work at inference.

1 Introduction

Autoregressive language models spend a transformer forward pass on every token, but not every token is equally hard. Function words (*the*, *of*, *to*), punctuation, sentence-initial capitalization, and productive suffixes are near-zero-entropy *in context*: given the surrounding content words, their identity is almost fully determined. Spending full model capacity, and a full sequence position, on each of them is wasteful.

We propose **token folding**. A greedy, left-to-right preprocessing pass fuses each predictable token into an adjacent content token by adding a small *modifier* embedding at that position. The content word "walking" with a preceding "the" and a trailing period becomes a single folded position carrying (base = *walking*, prev-word = *the*, punct = .). The transformer sees the shorter folded sequence; a lightweight per-slot head predicts the next position's modifiers. Generation emits the full unfolded token stream.

We evaluate folding along three distinct axes: matched *compute* (the same number of forward passes or folded positions), matched *text* (the same number of original tokens seen), and downstream quality. The main results are:

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.

- A fold pipeline and factored modifier head, with a fair-BPC accounting that charges the model for both the content prediction and the modifier prediction (Section 3).
- The quality cost is small and measurable at and beyond the compute-optimal budget. At 356M the folded model is +0.0068 BPC from a single-variable, compute-matched no-fold twin while processing 14.2% fewer positions, and the price does not meaningfully grow with training: +0.0068 at the compute-optimal budget and +0.0072 at 2.6× the budget, a difference of 0.3 standard deviations of the seed spread we measure at this size.
- The shorter sequence reduces inference cost, with a small but nonzero quality cost. Prefill throughput improves with context length (transformer-only break-even near 2K, +24.6% at 16K, or +17.4% once the fold's own prompt preprocessing is charged) at 0.87× peak memory, and decoding uses 14.2% fewer autoregressive forward passes. The language-modeling quality cost is small but measurable.

2 Related Work

Shortening the sequence. The idea of spending fewer positions on redundant input is well established. Funnel-Transformer pools the sequence progressively across depth, trading length for width in an encoder (Dai et al., 2020); Hierarchical ("hourglass") transformers apply the same intuition to autoregressive language modelling with a downsample–upsample stack (Nawrot et al., 2022); and dynamic token pooling learns *where* to segment rather than pooling at fixed stride (Nawrot et al., 2023). Token folding differs in *what* it removes and *when*. These methods shorten an already-tokenised sequence *inside* the network, so every position still enters the first layer and the model must learn the reduction. Folding is a preprocessing step: the sequence is shorter before the model sees it, the reduction rule is fixed and interpretable (a function word fuses into the following content word), and the removed token is still *predicted* , by a small auxiliary head rather than by the transformer. Our accounting charges the model for those predictions, which is what makes the comparison to an unfolded baseline fair.

Tokenizer-free and byte-level models. A parallel line removes the subword tokenizer (Sennrich et al., 2016; Kudo and Richardson, 2018) altogether. CANINE (Clark et al., 2022) and Charformer (Tay et al., 2022) operate on characters with learned or gradient-based segmentation; ByT5 (Xue et al., 2022) trains directly on bytes; MEGABYTE (Yu et al., 2023) makes long byte sequences tractable with a multiscale architecture; and the Byte Latent Transformer learns entropy-based patches (Pagnoni et al., 2025). T-FREE instead attacks the embedding table via sparse character-trigram representations (Deiseroth et al., 2024). These works aim to *eliminate* the tokenizer; token folding keeps it and asks a narrower question, given a subword tokenizer, are some of its tokens cheap enough to be predicted by something smaller than the full model? The comparison is not direct: these methods change the segmentation itself, whereas folding operates on the output of a tokenizer. We do not test folding with byte-level or tokenizer-free models.

A second distinction is representational. Folding exposes some surface-form structure, such as capitalization and suffix identity, as explicit modifier variables rather than leaving it implicit in subword identities. This can be viewed as a small morphological inductive bias. We do not isolate a consistent quality gain from that factor in the present experiments.

Factored and parallel prediction. Predicting more than one token per forward pass is the other way to decouple sequence length from compute. Multi-token prediction trains additional output heads to predict several future tokens at once (Gloeckle et al., 2024), and speculative decoding

accelerates inference by drafting with a cheaper model and verifying with the target (Leviathan et al., 2023). Folding is closest in spirit to the former, a small head predicts part of the output while the trunk predicts the rest, but the split is chosen by a linguistic criterion rather than by position, and it changes the training sequence rather than only the decoding procedure. The two mechanisms are complementary in principle: speculative decoding changes the decoding procedure, while folding changes the sequence modeled by the trunk.

Attention sinks. Transformers allocate large attention mass to a few low-information positions, and removing them destabilises the softmax; StreamingLLM shows that retaining such "sink" tokens is necessary for long-context streaming (Xiao et al., 2024), and vision transformers benefit from explicit register tokens serving the same role (Darcet et al., 2024). Because function words and punctuation can receive substantial attention mass, we test a learnable per-head sink logit. It improves the folded and no-fold models by similar amounts, suggesting a generic architectural effect rather than a fold-specific correction.

Compute-optimal training. Because our central claim is about quality per unit compute, the training budget matters. We report results at and around the compute-optimal budget for each model size (Hoffmann et al., 2022) rather than at a single fixed step count, since a fold model and its baseline reach a given step having processed different amounts of text.

3 Method

Fold pipeline. Given a tokenized sequence, the folding pipeline supports four fold types in a fixed order, greedy left-to-right, at most one fusion per slot: (1) *caps-fold*, a capitalized-word bigram whose lowercased form is a single token collapses to that token plus a CAP modifier; (2) *prev-word fold*, a function word fuses into the following content word; (3) *suffix fold*, a standalone suffix fuses into the preceding word; (4) *punct fold*, a period or comma fuses into the preceding word. Fold tables are precomputed per tokenizer and frequency-filtered. Different fold aggressiveness levels (how many function words and punctuation classes are foldable) define the configurations GFV3<GFV6<GFV8<GFV9 studied in the ablation.

Model and head. The backbone is a standard RoPE transformer. At each folded position the model receives the base token embedding plus additive modifier embeddings (one per occupied slot). It predicts, for position i: the next base token (standard LM objective) and the next position's per-slot modifiers via a small head that adds 1.6M parameters to a 356M backbone (+0.46%). Modifier targets are the *next* position's modifiers (what generation needs), not the current position's (which are trivially visible in the input).

Fair BPC. To compare a fold model against a no-fold baseline on the same text we charge both prediction costs and normalize by original characters:

$$\text{fair-BPC} \;=\; \frac{(\ell_{\text{base}} + \ell_{\text{mod}}) \cdot r}{c \cdot \ln 2}, \tag{1}$$

where ℓ_{base} is the cross-entropy on the next base token, ℓ_{mod} is the summed per-slot cross-entropy on the next position's modifiers, $r = T_{\text{folded}}/T_{\text{original}}$ is the compression ratio, and c is characters-per-token (4.73 for the Granite tokenizer). Both the fold and the no-fold model are scored on byte-identical held-out spans through the same code path; the no-fold model is the *same* architecture trained with folding disabled, so the comparison carries no harness offset.

Choice of comparison axis. A folded position contains more original text than an unfolded one, so matched compute and matched text are different comparisons. We use matched compute

throughout: the two arms take the same number of optimizer steps and transformer forward passes. The distinction affects training-time BPC accounting only; inference speed is measured per unit of processed or generated text. Section 4.1 gives the full accounting and the matched-text discussion.

4 Experimental Setup

Evaluation protocol. All held-out BPC uses 500 chunks strided across the held split. A second independent implementation agrees on the same text (+0.0078 versus +0.0072); Section 4.2 shows why contiguous slices can give substantially different gaps.

All models use the Granite tokenizer and a modern decoder-only RoPE architecture (352M parameters: d=1024, 28 layers, GQA 16/4, 2× SwiGLU FFN, tied embeddings); the fold model matches the baseline exactly except for the modifier head (a +1.6M net checkpoint delta). Training uses a shuffled 30B-token corpus. The no-fold baseline (`sdfv8_bl_g4`, step 24000, shuffled, batch 192) reaches held-out BPC 0.7481 on this metric; a fold run is always compared against the shuffle- and step-matched baseline checkpoint. All held-out numbers come from a single fixed-offset evaluation on the second half of the validation set (`eval_compare --split held --n-chunks 500`), so fold and baseline are scored on identical spans.

4.1 Training comparison axis

A folded position packs more original text than an unfolded one. For the reported configuration the train-split compression ratio is 0.805, so matched compute lets the fold arm read about 1.24× as much original text. Matched text would instead give it fewer gradient steps. Compute, text, and learning-rate position therefore cannot all be fixed simultaneously. We use matched compute because transformer forward passes are the scarce resource. This distinction concerns training-time comparison only and does not affect inference measurements per unit of processed or generated text. Using matched text instead would disadvantage folding, increasing the gap.

4.2 Evaluation text selection

The canonical scorer uses 500 chunks strided across the held split. A second independent implementation gives a gap of +0.0078 on those same offsets, compared with +0.0072 from the canonical scorer. The same second implementation gives +0.0176 on a contiguous prefix, whose absolute BPC is also much higher (0.97 versus 0.65). Thus text selection can move the measured fold penalty by more than seed noise. We therefore use strided held-out sampling throughout and do not interpret a single contiguous slice as a corpus-level estimate.

5 Results

Unless stated otherwise, "fold" is GFV3 with capitalization folding and *no* vocabulary retirement: compression r=0.858, i.e. 14.2% fewer transformer positions per unit of text, and no carrier tokens of any kind. Every fold number is paired with a no-fold twin of identical architecture, identical effective batch (192 sequences), identical peak learning rate (5×10^{-4}) and identical single-cosine schedule, differing only in the fold.

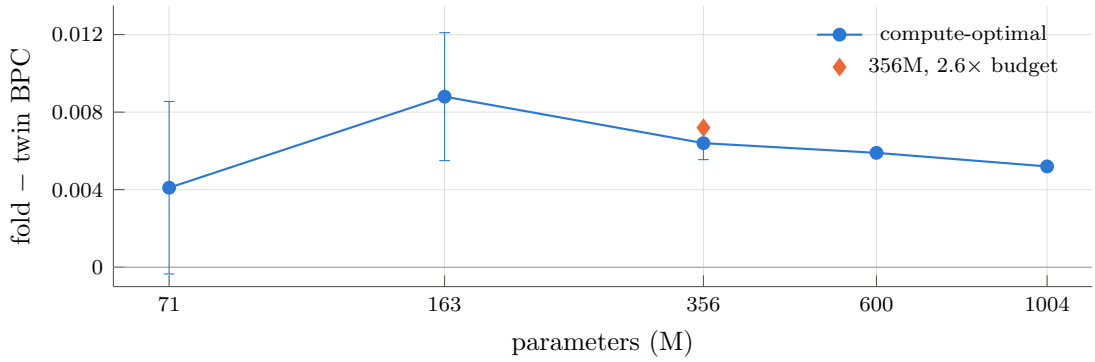

Figure 1: Fold cost across model size for GFV3. Error bars show 95% intervals where available; the diamond is the longer 356M run.

5.1 A fixed price for shorter sequences, at and past compute-optimal

Figure 1 shows the fold cost across model sizes at each size's compute-optimal budget. Table 1 summarizes the scale ladder and the longer 356M run. For the budget comparison we use the matched 356M draw at two completed training budgets, both with a *completed* cosine: the compute-optimal point for that size (Hoffmann et al., 2022), which at this batch size falls at ∼7.1B folded tokens (step ∼36,200), and 2.6× that. The gap is +0.0068 at compute-optimal and +0.0072 at 2.6×, a difference of 0.3 seed standard deviations (Section 6). The price of folding is therefore *flat* over the measured budget range rather than growing. The longer run reaches about 52 folded tokens per parameter, compared with about 20 at the compute-optimal point. We compare budgets only at completed learning-rate schedules. Intermediate checkpoints confound training budget with annealing progress and therefore do not identify how the folding cost changes with budget. The matched 356M endpoints provide the budget comparison used throughout the paper.

5.2 Checks on the measured cost

First, the gap is flat in evaluation context length: scoring the same pair over 640, 768, 896 and 1024 original tokens of context gives +0.0066, +0.0068, +0.0069 and +0.0072, with the two arms' context slopes agreeing to 4% (−0.042 vs −0.044 BPC per thousand tokens). This matters because a folded arm trains on 1024 *folded* positions but is scored on 1024 original tokens folded to ∼878, so it is evaluated below its trained sequence length while the twin is not. Had that asymmetry driven the result, the gap would close as context grows; it does not. The main configuration weights each auxiliary slot by $\lambda/n_{\text{slots}} = 0.075$ during training, while fair-BPC charges the full summed modifier loss. In a single matched-objective run, increasing the training weight to match the reported metric reduces the gap from +0.0068 to **+0.0030**. Base BPC rises slightly (0.6484 → 0.6501), while modifier cross-entropy falls (0.051 → 0.046 per slot). The main experiments use the original weighting; the matched-objective run indicates that the quality cost can be reduced by allocating more training weight to modifier prediction.

5.3 Which folds to apply

Fold aggressiveness has an optimum in our sweep. The pipeline can fold capitalization, function words, suffixes, and punctuation; the configurations below mainly vary the number of foldable function words and punctuation classes. At matched budget and with no retirement, GFV3_DROP (13

Table 1: Main GFV3 results. Gap is fold−twin in BPC. Scale rows are at compute-optimal training; the final row is the longer 356M run.

model	folded tokens	tok/param	gap	uncertainty	draws
71M	1.43B	20.0	+0.0041	95% CI $[-0.0004, +0.0085]$	5
163M	3.26B	20.0	+0.0088	95% CI $[+0.0055, +0.0121]$	3
356M	7.08B	19.9	+0.0064	95% CI $[+0.0056, +0.0073]$	3
600M	12.43B	20.0	+0.0059	two draws: $+0.0051, +0.0066$	2
1004M	19.86B	19.8	+0.0052	single draw	1
356M, 2.60×	18.54B	52.1	+0.0072	single draw	1

Table 2: Single-stream prefill, 356M fold vs. no-fold twin, bf16, one GPU, SDPA, batch 1, 32 row-sets averaged per cell; r is the compression actually achieved on the timed rows. *prefill* excludes the fold's own prompt preprocessing and *all-in* charges it; *peak* is a true peak with no baseline subtracted.

context	r	fold (ms)	+fold-pre	twin (ms)	prefill	all-in	peak
1K	0.860	14.95	1.40	14.36	−4.0%	−12.2%	0.96×
2K	0.858	15.81	2.58	16.14	+2.1%	−12.2%	0.93×
4K	0.857	34.06	5.19	38.44	+12.9%	−2.0%	0.91×
8K	0.858	100.12	10.19	119.99	+19.8%	+8.8%	0.89×
16K	0.857	328.76	20.24	409.79	**+24.6%**	**+17.4%**	0.87×

function words, 2 punctuation classes, r=0.873) and GFV3 (20 and 2, r=0.858) both reach 0.7495–0.7496, while the more aggressive GFV6 (20 function words, 5 punctuation classes, r=0.830) is worse at 0.7551 despite its shorter sequences. Each added fold class lowers the base-prediction cost but raises the modifier-prediction cost, and past a point the second term wins. Source: `logs/honest_nocarrier_g4_24k.tsv`.

5.4 Compute

The modifier head is a +1.6M net checkpoint delta on a 356M backbone (+0.46%), independent of sequence length. Because folding removes positions, FFN cost falls with r and attention cost falls with r^2; at r=0.858 that is 14.2% and 26.4% respectively, so the benefit grows with context length.

We measured single-stream prefill on the same checkpoint pair used for Table 1, at the compression this model actually achieves (Table 2). The gain grows with context, as expected from the linear and quadratic sequence-length terms, and is modest: folding is a small *loss* at 1K, breaks even near 2K, and reaches +24.6% at 16K. Peak memory, measured without subtracting a weights-containing baseline, is 4–13% lower and approaches r only at long context. The measured attention-FLOP ratio is 0.744–0.749 against r^2=0.736, and the KV-cache ratio is 0.857, equal to r to three decimals, both as predicted.

Charging the current pure-Python prompt folder moves break-even from about 2K to between 4K and 8K, and reduces the 16K gain from 24.6% to 17.4%.

5.5 Scale across model sizes

At compute-optimal training, the mean gap is +0.0041 at 71M, +0.0088 at 163M, +0.0064 at 356M, +0.0059 at 600M, and +0.0052 at 1004M. At 71M, the confidence interval includes zero. From 163M onward, the observed gap decreases by 0.0036 across the full 163M–1B span. The individual steps above 356M are smaller than the measured seed variation: the 356→600 and 600→1004 changes are smaller than the 0.0015 spread between the two 600M draws. We do not extrapolate a scaling law from these measurements. Appendix E gives the scale results in more detail.

5.6 Seed variance

We replicated the 71M, 163M, and 356M comparisons with five, three, and three seeds. The mean gaps are +0.0041, +0.0088, and +0.0064; seed standard deviations are 0.0036, 0.0013, and 0.0004. The corresponding 95% intervals are $[-0.0004, +0.0085]$, $[+0.0055, +0.0121]$, and $[+0.0056, +0.0073]$. Thus the 71M interval includes zero, while the 163M and 356M intervals exclude it. We do not extrapolate the shrinking variance beyond 356M because the two 600M draws differ by 0.0015.

5.7 Downstream multiple-choice evaluation

We additionally evaluate the 356M models zero-shot on HellaSwag, ARC-Easy, ARC-Challenge, WinoGrande, and PIQA by ranking candidate completions using character-length-normalized log likelihood (`acc_norm`). Across three independently trained fold/twin pairs, the unweighted mean across tasks differs by −0.0034 accuracy (95% CI $[-0.022, +0.015]$). Weighting tasks by their number of examples gives −0.0044 (95% CI $[-0.0079, -0.0010]$). Individual-task mean differences are small: HellaSwag −0.0052, ARC-Easy −0.0117, ARC-Challenge +0.0022, WinoGrande 0.0000, and PIQA −0.0024. For HellaSwag, the largest task, the three-draw gap is −0.0052 (95% CI $[-0.0115, +0.0012]$); a separate 2.6×-budget matched run gives −0.0082. The external multiple-choice evaluations therefore show a small overall cost. The example-weighted aggregate is statistically significant, while the task-macro average and HellaSwag interval include zero. Appendix A gives the per-task results and evaluation details.

5.8 Width and depth

Holding compute fixed, we fold across a grid of six shapes spanning 71M to 321M parameters, $d \in \{512, 1024\}$ by $L \in \{8, 16, 24\}$, each against its own matched no-fold twin, and each arm trained on its own complete cosine (Table 3). This grid uses the more aggressive GFV6 table (7 slots, r=0.830, so 17.0% shorter rather than 14.2%), which makes it an independent replication rather than a re-reading of the same configuration: different fold table, different held data-dir, different run family.

Folding costs +0.0056 to +0.0121 BPC at every shape, bracketing the +0.0068 we measure for GFV3 at 356M. The price appears across all six configurations. The price varies with *depth* while remaining nearly unchanged with width: at matched depth the two widths agree to within 0.0009, below our seed spread at this size, while going from 8 to 24 layers roughly doubles the cost at both widths. The bookkeeping split suggests why: folding's content-word advantage decays with depth (−0.0814 to −0.0705 at d=512) faster than its modifier bill does (+0.0868 to +0.0833). Deeper models predict content words well without help, which is precisely what folding sells.

The depth trend is distinct from parameter count because these runs hold *steps* fixed rather than tokens-per-parameter, which varies from 84 down to 19 across the grid. If budget drove

Table 3: Fold vs. matched no-fold twin across six shapes at fixed compute (30,518 steps, effective batch 192, 6.0B tokens per cell), each arm on its own *complete* cosine. GFV6 table, r=0.830. The last two columns are the same bookkeeping identity as Table 1 and sum to the gap. Absolute BPCs here are on a different held data-dir from Table 1 and are not comparable across the two; the gaps are. Source: `logs/sizesweep_rescore_result.tsv`.

shape	params	tok/param	twin	fold	gap	base-BPC	modifier
$d512$/8L	71M	84	0.8683	0.8739	+0.0056	−0.0814	+0.0868
$d512$/16L	89M	67	0.8215	0.8304	+0.0089	−0.0753	+0.0833
$d512$/24L	107M	56	0.7969	0.8090	+0.0121	−0.0705	+0.0833
$d1024$/8L	178M	34	0.7746	0.7805	+0.0059	−0.0723	+0.0780
$d1024$/16L	250M	24	0.7327	0.7407	+0.0080	−0.0679	+0.0762
$d1024$/24L	321M	19	0.7093	0.7205	+0.0112	−0.0634	+0.0744

the deficit the cells would order by tokens per parameter; they do not. The 107M cell at 56 tokens/parameter loses more (+0.0121) than the 178M cell at 34 (+0.0059), which is backwards for a budget explanation and exactly right for a depth explanation. We therefore treat this as a fixed-compute shape result rather than a scale result. The increase with depth is a potential concern for extrapolation to substantially deeper deployed models, although the absolute gap remains small even at 24 layers. This architectural sweep uses the more aggressive GFV6 folding configuration, rather than the more stable GFV3 configuration used for our main scale results, and holds training compute rather than tokens per parameter fixed.

6 Limitations

The headline 356M result has three seeds, giving a mean gap of +0.0064 BPC, seed sd 0.0004, and a 95% interval $[+0.0056, +0.0073]$. At 71M the corresponding interval includes zero, while at 163M and 356M it excludes zero; Section 5.6 gives the full seed study. The upper scale rungs have fewer replicates, so only the full 163M–1B decrease is claimed.

Our largest model is 1B parameters, and the longest controlled training comparison is 2.6× the compute-optimal budget. Within the tested range we do not see the folding cost worsen with model scale or additional training, but substantially larger and longer-trained models remain untested. In the GFV6 architectural sweep, the quality gap, while still small, increases with depth. Since frontier models tend to be substantially deeper, this is a potential concern for scaling folding to such models.

Our downstream evaluation is limited to zero-shot multiple-choice benchmarks. We do not test open-ended generation, so whether the small likelihood and multiple-choice differences translate to perceptible differences in generated text remains unknown. All headline language-modeling experiments also use one tokenizer and one web-text corpus; robustness across other corpora, tokenizer families, and languages remains untested. The fold rules are hand-specified and tokenizer-specific rather than learned.

Our primary comparisons match transformer-token compute. Matching raw training text would give the folded model fewer transformer-token updates and therefore increase the measured quality gap.

Finally, inference measurements are single-stream prefill on one GPU and one implementation; below 4K–8K context, the current all-in implementation does not improve throughput. These

limitations bound the claim to a measured quality-for-length tradeoff, not a general improvement in model quality or training efficiency.

7 Future Work

Reducing the quality cost. The current training objective was not optimized to minimize fair-BPC. In the matched-objective experiment of Section 5.2, changing only the auxiliary-loss weighting reduced the folding gap from +0.0068 to +0.0030 BPC. This suggests that the reported quality cost may not be intrinsic to folding. Better loss weighting, auxiliary-head design, and fold selection may reduce it further; whether the gap can be closed remains open.

Factorization as an inductive bias and vocabulary reallocation. Capitalization and suffix folding do more than shorten the sequence: they expose relationships that a flat token vocabulary represents only implicitly. For example, separate tokens for "Houses" and "houses," or for related suffixed forms, need not share parameters, whereas a representation in terms of a common base plus capitalization or suffix modifiers makes this structure explicit. Such sharing could improve sample efficiency, particularly early in training or for rare forms; with sufficient data, however, a standard model may learn similar relationships in its embeddings. The same factorization could also make some vocabulary entries redundant. Retiring surface variants could reduce embedding and output-vocabulary parameters, which may matter especially for smaller models. Alternatively, at fixed vocabulary size, freed entries could be reassigned to useful base or subword units, potentially shortening the remaining base sequence. Our current retirement implementation is not beneficial, so realizing either advantage requires a better retirement mechanism. Future work should test whether joint optimization of folding, retirement, and vocabulary allocation can turn the representational redundancy exposed by folding into parameter savings, additional sequence-length savings, or improved sample efficiency.

8 Conclusion

Token folding removes predictable modifier tokens from the transformer sequence and predicts them with a small auxiliary head. At 356M it shortens sequences by 14.2% for a mean cost of +0.0064 BPC over three seeds, and that cost is flat from the compute-optimal budget to 2.6× it. The trade is not favorable for pretraining compute, but it is useful at inference: at 16K context we measure +24.6% prefill throughput before prompt-folding cost and +17.4% after charging our current implementation, with peak memory at 0.87×. Folding provides a direct way to exchange a small amount of modeling quality for fewer autoregressive positions.

9 AI Tool Use

The original idea was conceived and developed without AI assistance. AI tools were used substantially during implementation and testing, with care taken to verify the reported results. AI assistance was also used in drafting and editing the paper.

References

Zihang Dai, Guokun Lai, Yiming Yang, and Quoc V. Le. Funnel-Transformer: Filtering out Sequential Redundancy for Efficient Language Processing. Advances in Neural Information Processing Systems 33 (NeurIPS), 2020.

Piotr Nawrot, Szymon Tworkowski, Michał Tyrolski, Łukasz Kaiser, Yuhuai Wu, Christian Szegedy, and Henryk Michalewski. Hierarchical Transformers Are More Efficient Language Models. Findings of the Association for Computational Linguistics: NAACL 2022, pages 1559–1571, 2022.

Piotr Nawrot, Jan Chorowski, Adrian Łańcucki, and Edoardo M. Ponti. Efficient Transformers with Dynamic Token Pooling. Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (ACL), pages 6403–6417, 2023.

Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural Machine Translation of Rare Words with Subword Units. Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL), 2016.

Taku Kudo and John Richardson. SentencePiece: A Simple and Language Independent Subword Tokenizer and Detokenizer for Neural Text Processing. Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, 2018.

Jonathan H. Clark, Dan Garrette, Iulia Turc, and John Wieting. CANINE: Pre-training an Efficient Tokenization-Free Encoder for Language Representation. Transactions of the Association for Computational Linguistics, 10:73–91, 2022.

Yi Tay, Vinh Q. Tran, Sebastian Ruder, Jai Gupta, Hyung Won Chung, Dara Bahri, Zhen Qin, Simon Baumgartner, Cong Yu, and Donald Metzler. Charformer: Fast Character Transformers via Gradient-based Subword Tokenization. International Conference on Learning Representations (ICLR), 2022.

Linting Xue et al. ByT5: Towards a Token-Free Future with Pre-trained Byte-to-Byte Models. Transactions of the Association for Computational Linguistics, 10:291–306, 2022.

Lili Yu, Dániel Simig, Colin Flaherty, Armen Aghajanyan, Luke Zettlemoyer, and Mike Lewis. MEGABYTE: Predicting Million-byte Sequences with Multiscale Transformers. Advances in Neural Information Processing Systems 36 (NeurIPS), 2023.

Artidoro Pagnoni et al. Byte Latent Transformer: Patches Scale Better Than Tokens. Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (ACL), pages 9238–9258, 2025.

Björn Deiseroth, Manuel Brack, Patrick Schramowski, Kristian Kersting, and Samuel Weinbach. T-FREE: Subword Tokenizer-Free Generative LLMs via Sparse Representations for Memory-Efficient Embeddings. Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 21829–21851, 2024.

Fabian Gloeckle, Badr Youbi Idrissi, Baptiste Rozière, David Lopez-Paz, and Gabriel Synnaeve. Better & Faster Large Language Models via Multi-token Prediction. Proceedings of the 41st International Conference on Machine Learning (ICML), PMLR 235:15706–15734, 2024.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast Inference from Transformers via Speculative Decoding. Proceedings of the 40th International Conference on Machine Learning (ICML), PMLR 202:19274–19286, 2023.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient Streaming Language Models with Attention Sinks. International Conference on Learning Representations (ICLR), 2024.

Timothée Darcet, Maxime Oquab, Julien Mairal, and Piotr Bojanowski. Vision Transformers Need Registers. International Conference on Learning Representations (ICLR), 2024.

Jordan Hoffmann et al. Training Compute-Optimal Large Language Models. Advances in Neural Information Processing Systems 35 (NeurIPS), 2022.

A Downstream multiple-choice details

Table 4 gives the per-task results underlying Section 5.7. All values are three-draw means from the matched 356M fold/twin pairs. Evaluation is zero-shot: candidate completions are ranked by character-length-normalized log likelihood (`acc_norm`). Character normalization uses the same completion text in both arms and therefore does not mechanically favor the folded model because its token sequence is shorter.

Table 4: Zero-shot multiple-choice accuracy for the matched 356M models, averaged over three independently trained fold/twin pairs. The macro average weights tasks equally; the example-weighted result weights by the number of evaluation examples.

task	n	fold	twin	fold − twin
HellaSwag	10,042	0.353	0.359	−0.0052
ARC-Easy	570	0.526	0.538	−0.0117
ARC-Challenge	299	0.298	0.295	+0.0022
WinoGrande	1,267	0.491	0.491	+0.0000
PIQA	1,838	0.652	0.654	−0.0024
macro average		0.464	0.467	−0.0034
example-weighted				−0.0044

Across the three draws, the macro-average gap is -0.0034 (95% CI $[-0.0223, +0.0155]$), while the example-weighted gap is -0.0044 (95% CI $[-0.0079, -0.0010]$). The difference in statistical significance reflects the aggregation: macro averaging gives each task equal weight, whereas the example-weighted result is dominated more strongly by HellaSwag, which has 10,042 examples. HellaSwag alone has a mean gap of -0.0052 (95% CI $[-0.0115, +0.0012]$).

B External sanity anchor for the baseline

Our central comparison is against our own no-fold twin, which invites the question of whether that twin is strong enough for parity against it to mean anything. We therefore score published base models of comparable size on the *identical* held split, through the *identical* scoring path used for every number in Section 5. We deliberately do not compare against figures reported in other

papers: bits-per-character depends on the tokenizer's characters-per-token and on the corpus, so cross-setup values share no scale.

Table 5: No-fold baselines vs. a published base model, scored on the same held split through the same scoring path. This is an in-domain held BPC on our corpus; it is on a different normalisation from the fair-BPC of Table 1 and the two must not be compared across. Source: `eval/models.yaml` pair `twin_vs_published`.

model	training budget	held BPC
our no-fold twin, 356M	1.10× compute-optimal	1.0223
our no-fold twin, 1B	1.00× compute-optimal	0.9149
published 3B base	(undisclosed, far larger)	0.7869

Our baselines fall on a monotone progression in size and budget, and the published model leads them by 0.128 BPC, a margin consistent with roughly three times the parameters and a much larger token budget. That is the shape one should expect if our baselines are ordinary well-trained models, and not the shape one would see if they were undertrained.

The published models were trained on far more tokens than our compute-optimal-budget twins, so they should score better; this anchor only checks that our baseline lies in a sensible range. All arms in our comparison also share one peak learning rate and one schedule, avoiding advantages from unmatched optimization settings.

C Additional ablations

Vocabulary retirement is net harmful. Retirement removes a capitalized or suffixed token from the output vocabulary and reconstructs it from a base token plus a carrier flag. Its appeal is a smaller softmax. Measured against the same fold with no retirement, at 0.66× compute-optimal: no retirement 0.7551; caps-only retirement 0.7697, i.e. +**0.0146** *worse*. The reason is that the carrier must be conditioned on the base token the model actually emits, and doing so honestly costs an order of magnitude more than the vocabulary saving returns. Directly measuring the saving: the trained model already assigns only 4.2×10^{-4} of its probability mass to the retired rows, so masking them is worth −0.0001 BPC. Sources: `logs/config_decompose_24k.tsv`, `logs/capsfix_penalty_8k.tsv`.

Folding capitalization into a shared form is a null. Since `President` and `president` are the same word, folding the capital into a flag should let the model share statistics. Across five budgets from 0.32× to 1.93× compute-optimal the effect is −0.0047, +0.0021, −0.0009, −0.0006, −0.0027: scattered about zero with a spread larger than any single value. Sources: `logs/unicaps_xy.tsv`, `logs/caps_intrinsic_floor.tsv`.

Attention sinks help, but not because of folding. Folding removes function words and punctuation, which is what attention sinks are made of (Xiao et al., 2024; Darcet et al., 2024), so folded models might be sink-starved. A read-only probe supports the premise: at matched sequence length the folded model places +0.0247 more attention mass on position 0 and has +6.7 points more sink-dominated heads. But adding a learnable per-head sink logit helps the folded and unfolded models by the same amount (−0.0048 vs. −0.0047 at 0.64×; −0.0048 vs. −0.0041 at 1.29×), so it is a generic improvement. Sinks lower both floors together and leave the gap unchanged. Sources: `logs/attn_sink_s0.tsv`, `logs/sink_test_g124m.tsv`.

D Vocabulary expansion and composition

D.1 Folding after vocabulary expansion

A natural question is whether folding remains useful after the tokenizer has already been optimized more aggressively for the training corpus. We therefore continued greedy BPE on top of the existing tokenization, adding 13,312 merges. The additional vocabulary entries reduce held-split sequence length by 14.24%. We then trained a model on a merge-then-fold corpus, applying the same folding mechanism after the additional BPE merges.

The two mechanisms compose. The merge-then-fold model reaches 22.7% shorter sequences than the original tokenizer, compared with 14.24% from the additional merges alone. Relative to the vocabulary-expanded arm, folding therefore removes a further 8.5 percentage points of sequence positions. Its BPC is +0.0060 relative to the vocabulary-expanded arm, close to the +0.0072 folding cost measured with the original tokenizer. Thus the opportunity exploited by folding remains after substantial additional BPE merging.

Why not represent folded combinations as vocabulary entries? A flat vocabulary could in principle encode particular combinations directly, but the number of entries needed for broad coverage can grow multiplicatively: a base token may combine independently with capitalization, a preceding function word, a suffix, and punctuation. Explicitly adding many such combinations would therefore require a large number of additional embedding and output rows, while rare combinations would receive little training data. Folding instead factorizes the representation into a base-token embedding plus small modifier embeddings, sharing each modifier across base tokens and across combinations. Vocabulary expansion and folding therefore address different aspects of the representation and can be applied together.

E The scale claim, stated narrowly

The fully annealed scale ladder gives fold-minus-twin gaps of +0.0088 at 163M, +0.0064 at 356M, +0.0059 at 600M, and +0.0052 at 1B. The full 163M–1B decrease is larger than the measured seed variation, while the individual steps above 356M are smaller than it. We therefore claim only that the observed folding cost decreases across the full scale sweep; we do not infer a particular functional scaling law or distinguish the two largest rungs. The seed study used to calibrate this precision is described in Section 5.6.

The separate six-shape sweep from 71M to 321M at fixed compute (Section 5.8) should not be read as additional points on this scale curve. It uses a more aggressive fold table and varies tokens per parameter from 84 to 19. Its main result is instead that the folding cost depends on depth but shows little dependence on width at matched depth.