zip2zip: Inference-Time Adaptive Vocabularies for Language Models via Token Compression

Anonymous Authors[1]

Abstract

Tokenization efficiency plays a critical role in the performance and cost of large language models (LLMs), yet most models rely on static tokenizers optimized on general-purpose corpora. These tokenizers' fixed vocabularies often fail to adapt to domain- or language-specific inputs, leading to longer token sequences and higher computational costs. We introduce `zip2zip`, a framework that enables LLMs to dynamically adjust the token vocabulary at inference time, allowing for fewer generated tokens and thus faster inference. `zip2zip` consists of three key components: (1) a tokenizer based on Lempel-Ziv-Welch (LZW) compression that incrementally merges co-occurring tokens into reusable *hypertokens* on the fly; (2) an embedding layer that computes embeddings for newly formed hypertokens at runtime; and (3) a causal language modeling variant that trains the model to operate on hypertokenized, compressed sequences. We show that an existing LLM can be zip2zip-fied in 10 GPU-hours via parameter-efficient finetuning. The resulting zip2zip LLMs effectively learn to use hypertokens at inference time, reducing input and output sequence length by 20–60%, with significant improvements in inference latency. Code is available at this anonymous repository.

1. Introduction

Large language models (LLMs) have shown impressive versatility across a broad spectrum of tasks and domains (Brown et al., 2020; Bubeck et al., 2023), including biomedical tests (Nori et al., 2023), mathematical reasoning (Frieder et al., 2023), programming (Jiang et al., 2024), and multiple human languages. A critical underlying component of this flexibility is the tokenizer, which defines the model's vocabulary and governs how raw text is converted into token sequence fed to the model. The efficiency of the tokenization scheme—i.e., how compactly a text is represented as tokens—has significant impact on model performance. In particular, a more *compact* tokenization yields three key benefits: (1) larger effective context window; (2) lower computational (and thus monetary) cost; and (3) shorter response times.

Despite its importance, the tokenizer used in most LLMs produces a fixed, static vocabulary using algorithms such as Byte Pair Encoding (Sennrich et al., 2016) over large-scale, general-purpose web corpora. While this globally optimized vocabulary performs reasonably well on average, it often fails to adapt to domain-specific or language-specific distributions (Ahia et al., 2023; Petrov et al., 2023), where the text distribution diverges significantly from the pretraining data. The resulting mismatch leads to longer token sequences, increasing both memory and compute demands, as well as the end user's cost by a factor of 2-3x when processing domain-specific text (Ahia et al., 2023). To mitigate this issue, prior work has explored expanding the token vocabulary during domain or language adaptation to improve tokenization efficiency (Wang et al., 2019; Zhao et al., 2024; Kim et al., 2024; Liu et al., 2023; 2024). While effective, this approach needs to be repeated for each target domain or language and requires maintaining separate tokenizers. Meanwhile, commercial LLM providers trend toward increasing the size of token vocabularies—growing from 32K to 128K (Grattafiori et al., 2024) and even up to 200K (Abdin et al., 2024) tokens—to improve overall tokenization efficiency. However, prior work (Dagan et al., 2024; Liang et al., 2023) shows that simply enlarging the vocabulary yields diminishing returns in domain adaptation, and vocabularies past a certain size can potentially degrade model performance (Liang et al., 2023).

These limitations point to a compelling need for an adaptive tokenization mechanism—one that can dynamically tailor the vocabulary to the input text at inference time, without retraining the model or maintaining separate tokenizers. Such a mechanism would allow the model to construct new domain-specific tokens on-the-fly, so to enhance tokeniza-

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

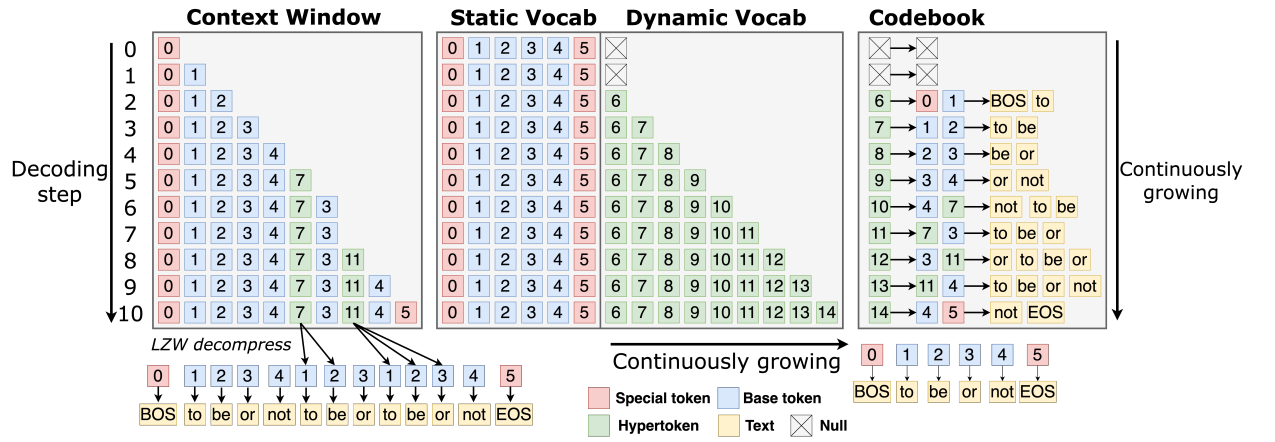


Figure 1: **Overview of the `zip2zip` inference pipeline.** At each decoding step, the model has a growing **context** composed of both **base tokens** (blue) and **hypertokens** (green). The **static vocabulary** of size 6 remains fixed, while the **dynamic vocabulary** is continuously expanded by merging co-occurring tokens using **LZW compression**. The **codebook** (right) maps hypertoken IDs to their corresponding base tokens. As decoding progresses, new hypertokens created at step t (e.g., "to be", "or not") become **immediately** available for reuse at step $t+$. Additionally, output tokens, once generated, **instantly** become eligible for compression. Hypertokens are also eligible for merging, enabling the formation of **nested hypertokens**. The final output sequence (bottom) is reconstructed via LZW decompression.

tion efficiency. However, adaptive tokenization poses architectural challenges, as both the embedding layer and the language modeling head in transformer models (Vaswani et al., 2017) are static matrices tied to a fixed vocabulary size.

In this paper, we propose `zip2zip` (with a hat-tip to seq2seq (Sutskever et al., 2014)), a method that equips LLMs with a dynamic token vocabulary, enabling inference-time token adaptation. `zip2zip` achieves adaptive tokenization through *continuous vocabulary expansion* at runtime, allowing the model to represent a repeated or domain-specific pattern with a single long token rather than inefficient short tokens. This requires modest modifications to both the transformer architecture and the language modeling objective. `zip2zip` comprises three key components: (1) **Tokenizer:** an integration of Lempel-Ziv-Welch (LZW) compression[1] (Welch, 1984) into the tokenization process, which continuously merges frequently co-occurring token sequences into reusable longer tokens (hypertokens) at runtime; (2) **Architecture:** a lightweight encoder added to the transformer that computes embeddings for newly formed tokens on the fly; (3) **Training:** a compression-aware causal language modeling variant that trains the model directly on LZW-compressed sequences, aligning learning with the inference-time token distribution. The name `zip2zip` reflects its dual role in achieving compression of both the input tokens (the first *zip*) and output tokens (the second *zip*), thereby jointly improving the efficiency of input encoding and output decoding. We finetune `Phi-3-4B` and `Phi-3-14B` to support `zip2zip` using as few as 100M tokens—requiring only 10 and 40 H100 GPU hours, respectively—for effective adaptation. The resulting models demonstrate strong inference-time compression capabilities and achieve 20–60% reductions in both input and output sequence lengths, translating to up to 60% improvements in end-to-end latency.

To make it easy to upgrade existing LLMs to `zip2zip`, we release an efficient, open-source implementation of the framework. It includes (1) a fast Rust-based LZW tokenizer, (2) a drop-in model architecture compatible with Hugging Face Transformers and vLLM, (3) a training pipeline for LZW-compression-based finetuning. Existing LLMs can be seamlessly extended with `zip2zip`, gaining adaptive tokenization capabilities through parameter-efficient finetuning—without any changes to the base model or tokenizer.

2. `zip2zip`

2.1. Dynamic Token Vocabulary

To enable dynamic tokenization at inference time, we associate each LLM with a *hyper-vocabulary* $\mathcal{V}_h$ that augments the model's static token vocabulary. Tokens from the original vocabulary $\mathcal{V}$ are referred to as *base tokens*. Each entry in the hyper-vocabulary is a *hypertoken*, representing a merged sequence of base tokens. The total vocabulary for a `zip2zip` model is the union $\mathcal{V} \cup \mathcal{V}_h$. At the beginning of each inference session, $\mathcal{V}_h$ is initialized as an empty set, and is incrementally populated during decoding by identifying and merging recurring token subsequences in the context window, as illustrated in Figure 1.

[1] LZW is the algorithm used in zip compression tool, which inspired the name `zip2zip`.

Continuous Vocabulary Expansion. As decoding proceeds, `zip2zip` continuously merge co-occurring tokens as new hypertokens to $\mathcal{V}_h$ and recurrently apply merging on newly generated tokens. This *continual expansion* allows the model to represent longer, recurring sequences of base tokens compactly. Hypertokens are treated as first-class tokens within the model, used interchangeably with base tokens throughout the decoding process. Importantly, this process occurs entirely during inference, without modifying the underlying tokenizer or requiring model retraining.

LZW Algorithm. We implement vocabulary expansion using the Lempel-Ziv-Welch (LZW) compression algorithm—a dictionary-based, lossless compression method that incrementally builds a codebook of variable-length sequences. In our setting, the codebook is initialized with the base token vocabulary $\mathcal{V}$ and expands by adding new hypertokens on the fly as recurring token patterns are encountered. To control the growth of the dynamically expanding vocabulary, we impose a maximum merge size M that restricts how many base tokens a single hypertoken can represent. LZW is particularly well-suited for `zip2zip` due to the following properties:

(1) it is ***streaming***—hypertokens created at step t can be immediately reusable at step $t+1$; in contrast, methods like BPE require access to the full sequence and operate offline;

(2) it is ***self-contained***—input base tokens can be perfectly reconstructed from the compressed token sequence alone[2];

(3) it is ***unambiguous***—when both base tokens and hypertokens are available, which one to use is consistently determined by the LZW algorithm without ambiguity.

2.2. Hyper-Embedding and Hyper-Projection

Hypertokens do not have fixed embedding vectors in the original model's embedding layer (and projection layer), as they are not part of the original vocabulary. To compute the embedding of a hypertoken, we learn a mapping from the base token embeddings to the hypertoken embedding. We achieve this by introducing a *hyper-encoder*, which is a neural network that takes the embeddings of the constituent base tokens as input and outputs the corresponding hypertoken embedding. Specifically, for a sequence of M base tokens $y_{1:M} := y_1 \dots y_M$, the hyper-encoder $f_\phi : \mathcal{V}^M \to \mathbb{R}^d$ produces the hypertoken embedding $h = f_\phi(y_{1:M}) \in \mathbb{R}^d$, where M is the maximum merge size and d is the embedding dimension. For hypertokens composed of fewer than M base tokens, we pad the input sequence to length M. Since the embedding map for base tokens remains unchanged, the hyper-encoder f_ϕ essentially maps the concatenated base token embeddings from a $(M \times d)$-dimensional space to a d-dimensional hypertoken embedding vector, performing nonlinear dimensionality reduction.

For the output projection layer, if the underlying transformer ties the embedding and the projection matrices, one can reuse the same hyper-encoder to compute the representation used for projection. Otherwise, a separate hyper-encoder is trained to produce the hypertoken projection vectors.

[2]There is no need to persist or transmit the codebook across inference calls, preserving compatibility with existing LLM libraries and interfaces.

2.3. Architecture

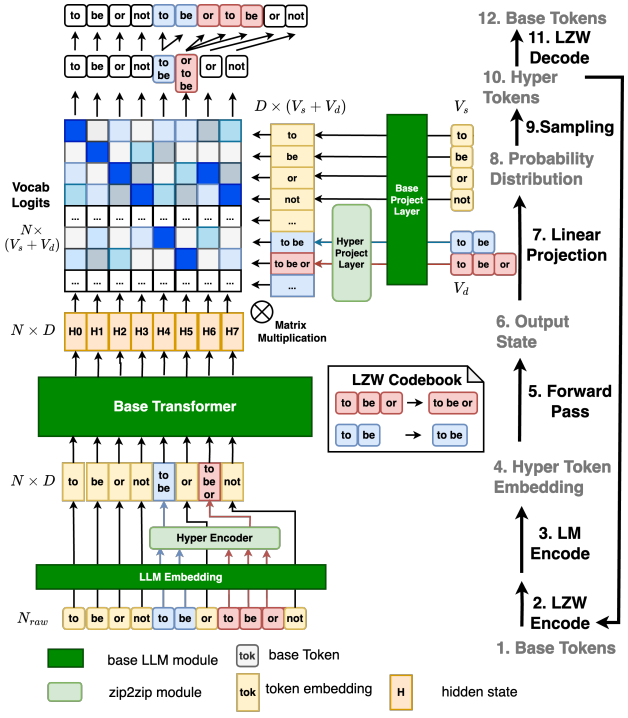


Figure 2: **`zip2zip` architecture.** At inference time, base tokens are compressed into hypertokens using LZW (**STEPS 1–2**). A hyper-encoder computes embeddings for hypertokens (**STEP 3–4**), which are processed by the base LLM (**STEPS 5–6**). Output representations are projected jointly on base and hyper-projection layers (**STEP 7**), producing joint logits and sampled tokens (**STEPS 8–10**), which can be decoded back to base tokens (**STEPS 11–12**).

We illustrate the architecture of `zip2zip` in Figure 2. The input text is first tokenized into base tokens (**STEP 1**), which are then passed through an online LZW compressing module that compresses the token sequence into a stream of hypertokens (**STEP 2**). Since hypertokens are not part of the model's original embedding layer, their embedding vectors are computed on-the-fly using the *hyper-encoder* during inference (**STEP 3–4**). Once embedded, both base token embeddings and hypertokens embeddings are passed through the standard transformer layers of the base model, producing contextualized hidden states (**STEP 5–6**). This step is identical to vanilla transformer, with hypertokens and base tokens treated equally. At the output projection layer, hyper-

token projection vectors (same as the hypertoken embedding vectors in the tied case, and computed by a separate hyper-encoder otherwise) are appended to the original projection matrix in the language modeling head (**STEP 7**). This allows the model to compute a joint softmax over the union of the base vocabulary and the hyper vocabulary $\mathcal{V} \cup \mathcal{V}_h$ (**STEP 8**). The resulting probability distribution is over $\mathcal{V} \cup \mathcal{V}_h$, and the sampled token may be either a base token or a hypertoken (**STEP 9**). In the next cycle, the newly generated token (**STEP 10**) —whether base or hyper—is appended to the input sequence, and the process repeats (back to **STEP 1**). At the end of generation, the hypertoken sequence is decompressed via the LZW decoding function into a sequence of base tokens (**STEP 11–12**).The whole process works in a fully *autoregressive* way, where newly generated tokens will also be merged into hypertokens for future steps. Furthermore, we highlight two points:

Consistent Vocabulary Updates. The expanding vocabulary—comprising newly created hypertokens—must be updated in a *consistent* manner across both the input embedding layer and the output projection layer, maintaining a consistent view of the hypertoken set. Failure to update both sides consistently can result in two types of errors: (1) hypertokens that cannot be decoded, or (2) the model attempting to decode a non-existing hypertoken.

Hyper-Embedding Cache. Although hypertoken embeddings are computed on-the-fly, they are context-independent and can thus be cached across inference steps. Similar to the transformer's KV-cache, this enables *incremental* updates: only newly created hypertokens need to be processed at each step. Since the codebook grows linearly with the number of tokens in the context n, the total cache size grows also linearly in memory. Thus, the computational cost for hypertoken embeddings remains constant per step—i.e., one token embedding is computed per step.

2.4. Training `zip2zip` models

Objective. Let $\mathcal{D}$ denote the target text distribution. Given a language model π_θ parameterized by θ, standard pretraining seeks to minimize the causal language modeling (CLM) objective, which corresponds to the expected negative log-probability of data sequences under the model:

$$\min_{\theta} \mathbb{E}_{y\sim\mathcal{D}} \left[-\log \pi_\theta(y)\right], \tag{1}$$

where $\pi_\theta(y)$ denotes the probability of the token sequence y under the model π_θ.

Let $\mathcal{C}$ be an *online* compression algorithm (e.g., LZW), and ϕ be the parameters of the hyper-encoder. Given a sequence $y \sim \mathcal{D}$, let $z = \mathcal{C}(y)$ be its compressed form. In `zip2zip`, we aim to optimize the same CLM loss, but over the compressed sequences z. The training objective becomes:

$$\begin{aligned} &\min_{\theta,\phi} \mathbb{E}_{y\sim\mathcal{D}} \left[-\log \pi_{\theta,\phi}(\mathcal{C}(y))\right] \\ &= \min_{\theta,\phi} \mathbb{E}_{z\sim\mathcal{C}(\mathcal{D})} \left[-\log \pi_{\theta,\phi}(z)\right]. \end{aligned} \tag{2}$$

Here, we slightly abuse the notation to let $\pi_{\theta,\phi}(z)$ denote the probability assigned to the compressed sequence z, parameterized by the base model weights θ and the hyper-encoder parameters ϕ.

To construct the compressed dataset $\mathcal{C}(\mathcal{D})$, we first tokenize the corpus using a standard tokenizer, and then apply the LZW compression algorithm. This preprocessing step is performed once prior to training and can be efficiently parallelized through batching.

Parallelizable Training via Causal Masking. Although hypertokens introduce additional vocabulary dynamics, training remains fully parallelizable. We leverage the standard causal masking mechanism used in language models, allowing the model to predict the next token—whether a base token or a hypertoken—at each position in parallel. To eliminate the need for sequential codebook updates during inference, we precompute a fixed codebook by applying LZW compression to the entire input sequence. This precomputed codebook is then used consistently throughout training to condition token predictions, ensuring efficiency and compatibility with standard training pipelines.

Auxiliary Reconstruction Loss. We introduce an auxiliary reconstruction objective that encourages a hypertoken embedding to retain sufficient information about its underlying base token sequence. Specifically, the model is trained to reconstruct the original base token embeddings from the hypertoken embedding. We jointly optimize the language model and the hyper-encoder using a combined loss that includes both the standard next-token prediction loss and the auxiliary reconstruction loss. Formally, we optimize:

$$\begin{aligned} &\min_{\theta,\phi,\psi} \mathbb{E}_{y\sim\mathcal{D}} \left[-\log \pi_{\theta,\phi}(\mathcal{C}(y))\right] \\ &+\lambda\, \mathbb{E}_{y_{1:M}} \left[\Delta\left(y_{1:M}, f_\psi\left(f_\phi(y_{1:M})\right)\right)\right], \end{aligned} \tag{3}$$

where $f_\phi : \mathcal{V}^M \to \mathbb{R}^d$ is the hyper-encoder, $f_\psi : \mathbb{R}^d \to \mathcal{V}^M$ is the decoder aiming to reconstruct the corresponding base tokens from their hyper-embedding, and $\Delta : \mathcal{V}^M \times \mathcal{V}^M \to \mathbb{R}$ is the reconstruction loss function, such as the cross-entropy loss, between the base tokens $y_{1:M}$ and the reconstructed base tokens $f_\psi\left(f_\phi(y_{1:M})\right)$. The hyperparameter λ controls the trade-off between the prediction error of the language model and the reconstruction error of the autoencoder. This joint optimization objective encourages the hyper-encoder to learn a compact d-dimensional manifold embedded in the higher-dimensional $(M \times d)$ space of base token embeddings, while the language model $\pi_{\theta,\phi}$ learns to predict the next (hyper)token given the preceding context.

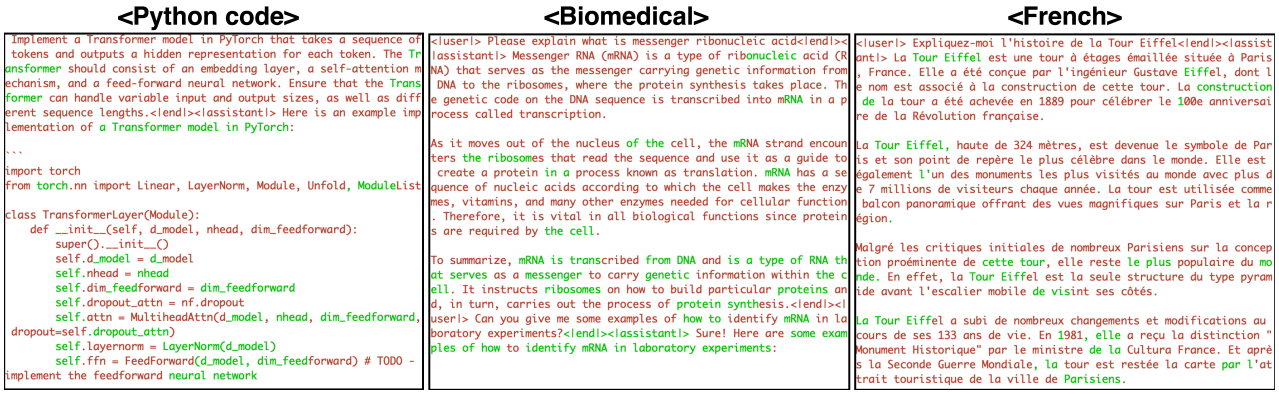


Figure 3: **Examples of zip2zip model outputs**. The green text indicates hypertokens and the red text indicates the base tokens with magnified annotations illustrating hypertoken formation.

The reconstruction loss can be viewed as a form of auto-encoding, where the hypertoken acts as a compressed latent representation and reconstruction encourages the preservation of semantic content and the compression to be lossless.

Adapting Pretrained Language Models. The proposed objectives (Equation 2, 3) integrate naturally with pretrained language models. In this setting, the base model can be frozen while training only the hyper-encoder to adapt to compressed token sequences. Parameter-efficient methods such as LoRA (Hu et al., 2022) may also be used to adapt select components of the base model, enabling effective adaption with minimal computes.

2.5. Efficiency Advantage

`zip2zip` improves efficiency by increasing the average token length, thereby reducing the number of tokens required to represent the same text. This compression applies to both inputs (e.g., prompts) and outputs (e.g., completions), leading to shorter effective context lengths. As a result, the model performs fewer computations—both in the attention mechanism and the feedforward layers—and, more importantly, requires fewer autoregressive decoding steps during inference. Since the latency of large language models is primarily driven by the cost of sequential decoding, reducing the number of output tokens by $n\%$ leads to an approximate $n\%$ speedup in decoding latency, which we will demonstrate empirically in Section 3.6. A more detailed discussion of FLOPs is provided in Appendix B for completeness.

3. Experiments

To evaluate the effectiveness of `zip2zip`, we adapt the Phi-3 models (3B and 14B) within the `zip2zip` framework. We evaluate our adapted models across four dimensions: (1) token efficiency, (2) language modeling perplexity, (3) downstream task performance, (4) inference efficiency. For perplexity and downstream benchmarks, we use the widely adopted lm-evaluation-harness framework (Gao et al., 2024).

3.1. Training Setup

Rather than updating the full model weights, we adopt parameter-efficient finetuning using LoRA (Hu et al., 2022). In addition, we train the *hyper-embedding* and *hyper-projection* modules. We set the maximum merge size to $M = 3$ and use a two-layer transformer encoder as the hyper-encoder. Ablation studies on M and hyper-encoder architecture can be found in Appendix A. For comparison, we also perform continual finetuning of the base model using LoRA under identical training conditions, serving as a baseline (denoted as Cont. Finetune in the Tables) The finetuning process is highly efficient, requiring approximately 10 H100-GPU hours for a 4B-parameter model and up to 40 H100-GPU hours for a 14B-parameter model, using only 0.1 billion training tokens. Interestingly, the *reconstruction loss* converges to **near zero** during training, indicating that the model can almost perfectly recover the original base token sequences from the hypertoken representations. This highlights the learned compression is highly information-preserving. Details of the training setup, compute infrastructure, and dataset curation are provided in Appendices D and E.

3.2. Sample Outputs and Hypertoken Patterns

We present several examples to provide intuition into how the `zip2zip` model generates text.

We see that the model successfully generates a mixture of hypertokens and base tokens in the output (see Figure 3). The hypertoken ratio is as high as 40% in the Python code generation example, and 20% in the biomedical text generation example. Many of the hypertokens correspond to semantically meaningful units or domain-specific terms as

Table 1: Examples of hypertokens formed by zip2zip across three domains

Code Generation	Biomedical	French
Py + Torch = PyTorch	m + R + NA = mRNA	tour + istique = touristique
Layer + Norm = LayerNorm	mess + enger = messenger	le + plus = le plus
Trans + former = Transformer	gen + etic = genetic	Tour + Eiff + el = Tour Eiffel
d_ + model = d_model	D + NA = DNA	Paris + iens = Parisiens

Table 2: **Token efficiency** (bytes/token) across domains for different tokenizers w/wo `zip2zip`.

Tokenizer	Code	Math	Chat	Multilingual	Web
Llama-3-128K (Grattafiori et al., 2024)	4.1	2.7	5.1	3.8	4.6
+zip2zip	6.3 (+54%)	4.0 (+48%)	6.4 (+25%)	4.7 (+24%)	5.4 (+17%)
Qwen-2-150K (Yang et al., 2024)	4.0	2.3	5.1	3.7	4.4
+zip2zip	6.2 (+55%)	3.7 (+61%)	6.4 (+25%)	4.6 (+24%)	5.2 (+18%)
Phi-4-200K (Abdin et al., 2024)	4.1	2.7	5.4	4.6	4.7
+zip2zip	6.3 (+54%)	4.1 (+52%)	6.7 (+24%)	5.5 (+20%)	5.4 (+15%)
Gemma-3-256K (Team et al., 2025)	3.3	2.3	5.0	4.4	4.5
+zip2zip	5.6 (+70%)	3.7 (+61%)	6.4 (+28%)	5.4 (+23%)	5.4 (+20%)

shown in Table 1. For a more fine-grained visualization of hypertoken with `zip2zip` , we provide visualizations of token streams in Appendix 8.

3.3. Token Efficiency

Given an input text x and a tokenizer, we define *token efficiency* $\eta := \frac{\text{Bytes}(x)}{\text{Tokens}(x)}$ as the average number of bytes represented by each token, where $\text{Bytes}(x)$ refers to the number of bytes in the UTF-8 encoding of x. This measures how compactly a tokenizer encodes input text—higher values of η indicate more efficient tokenization.

We evaluate token efficiency using the tokenizers of four LLMs—Llama-3 (Grattafiori et al., 2024), Qwen-2 (Yang et al., 2024), Phi-4 (Abdin et al., 2024), and Gemma-3 (Team et al., 2025)—each associated with a different base vocabulary size ranging from 128K to 256K. Token efficiency is measured across five representative domains, sampled from publicly available datasets: code (Lozhkov et al., 2024b), math (LI et al., 2024), chat (Ding et al., 2023), multilingual (Penedo et al., 2024), and web (Lozhkov et al., 2024a). Table 2 shows that applying LZW `zip2zip` consistently improves token efficiency across all tokenizer and domains. Gains are particularly strong in structured domains like code and math—50% higher than the base tokenizer. Interestingly, models with larger vocabulary sizes do not always achieve better token efficiency, suggesting that simply enlarging the vocabulary size is not sufficient to improve it.

3.4. Perplexity

We evaluate the perplexity of `zip2zip` models on four corpora: Wikitext (Merity et al., 2016), the Pile (Gao et al., 2020), and two subsets of Paloma (Magnusson et al., 2023): mC4, a multilingual subset of C4, and dC4 (aka C4-100D),

Table 3: Byte-perplexity (↓) on four corpora using a 1024-token context window.

Model	Method	Wiki	Pile	mC4	dC4
Phi-3.5-4B	Base	1.62	1.88	1.94	1.77
	Cont. finetune	1.63	1.89	1.94	1.77
	zip2zip	1.71	2.02	2.04	1.84
Phi-3-14B	Base	1.43	1.72	1.82	1.67
	Cont. finetune	1.47	1.79	1.86	1.68
	zip2zip	1.56	1.90	1.96	1.75

a subset of C4 spanning 100 domains. Given a token sequence $x = x_1, \ldots, x_N$, and a model q, perplexity and byte-level perplexity (Radford et al., 2019; Magnusson et al., 2023) are defined as: $\text{PPL} := \left(\prod_{i=1}^{N} q(x_i)\right)^{-1/N}$, $\text{Byte-PPL} := \left(\prod_{i=1}^{N} q(x_i)\right)^{-1/B} = \text{PPL}^{1/\eta}$, where B is the number of UTF-8 bytes of the text, and η denotes the token efficiency (i.e., bytes per token). Token-level perplexity depends on the tokenization scheme and is unsuitable for cross-tokenizer comparison. We instead report byte-level perplexity, a vocabulary-agnostic metric that normalizes for tokenization differences. Table 3 shows that `zip2zip` model has a modest increase in Byte-perplexity, indicating a drop in language modeling performance.

3.5. Evaluation on NLP Benchmarks

We next evaluate `zip2zip`'s performance on real-world tasks. We evaluate on seven widely used NLP benchmarks, including ARC-[Challenge, Easy] (Clark et al., 2018), HellaSwag (Zellers et al., 2019), LAMBADA (Paperno et al., 2016), OpenbookQA (Mihaylov et al., 2018), PIQA (Bisk et al., 2019), Winogrande (Sakaguchi et al., 2019) and

Benchmark	Phi-3.5-4B			Phi-3-14B		
	Base	C.F.	Z2Z	Base	C.F.	Z2Z
ARC-c	0.60	0.60	0.57	0.62	0.62	0.62
ARC-e	0.83	0.82	0.83	0.80	0.88	0.86
HS	0.66	0.63	0.61	0.70	0.66	0.68
OBQA	0.46	0.47	0.46	0.51	0.52	0.51
PIQA	0.79	0.82	0.82	0.83	0.87	0.85
WG	0.75	0.75	0.75	0.76	0.80	0.79
GSM8K	0.82	0.40	0.15	0.84	0.52	0.25

Table 4: Two-shot accuracy (in %) across 7 NLP benchmarks. Higher is better. Standard deviations (bootstrapped) ≈ 0.02 across all tasks. C.F.=Continuous finetune, Z2Z=zip2zip.

GSM8K (Cobbe et al., 2021). As shown in Table **??**, the model finetuned with `zip2zip` performs similarly to the baseline on most tasks. However, on GSM8K, where the primary task involves numerical computation, the model exhibits significant degradation. Due to the sensitivity of such tasks to tokenization, it occasionally generates malformed or repeated numbers. While token-level operations are already known to be challenging for LLMs (Singh & Strouse, 2024), adaptive tokenization appears to exacerbate this issue.

To validate the effectiveness of `zip2zip` on non-English languages, we evaluate the model on machine translation tasks, including WMT14 (Macháček & Bojar, 2014), WMT16 (Bojar et al., 2016). The results, shown in Table 5, indicate a small performance degradation across BLEU, CHRF, and TER metrics when using `zip2zip`. However, the drop is relatively minor, suggesting that the model retains strong multilingual capabilities even in the compressed representation.

3.6. Inference Efficiency

`zip2zip` reduces decoding time by lowering the number of tokens that need to be generated. However, it introduces additional FLOPs due to the on-the-fly computation of hyper-embeddings by the hyper-encoder. To address this overhead, we implement hyper-embedding caching and optimize the computation using a custom Triton kernel. We report separate timings for *prefill* and *decoding* across multiple models, with and without `zip2zip`, in Table 6.

As we show in Table 6, `zip2zip` achieves a significant speedup in all four settings. Both prefill and decoding times are significantly reduced, with the most substantial gains observed in the 512+256 setting with the Phi-3.5-4B model. Improvements are significantly stronger on datacenter-grade GPUs like the NVIDIA H100 and more modest on consumer hardware (e.g., Apple M1).

Table 5: Machine translation performance on WMT benchmarks. Scores are averaged across both translation directions. Standard deviations (approximately $1.0 \sim 2.0$) are reported in Table 10 in Appendix C.

Model	Method	Metric	WMT14 En-Fr	WMT16 En-De	WMT16 En-Ro
Phi-3.5-4B	Base	BLEU↑	33.6	39.2	17.7
		CHRF↑	58.3	63.2	45.5
		TER↓	53.0	47.9	73.4
	Cont. finetune	BLEU↑	36.5	42.3	16.7
		CHRF↑	61.0	65.4	45.8
		TER↓	51.5	44.9	79.7
	zip2zip	BLEU↑	34.1	39.7	14.3
		CHRF↑	59.4	64.5	44.2
		TER↓	54.5	48.0	93.5
Phi-3-14B	Base	BLEU↑	39.1	43.1	21.3
		CHRF↑	62.6	65.6	51.0
		TER↓	49.3	44.1	70.5
	Cont. finetune	BLEU↑	38.9	48.4	21.8
		CHRF↑	63.2	70.1	52.0
		TER↓	48.8	39.8	68.3
	zip2zip	BLEU↑	36.4	44.8	19.5
		CHRF↑	62.8	68.1	50.1
		TER↓	51.2	42.9	72.9

Efficient LZW Tokenization. `zip2zip` introduces an additional LZW compression step during inference and a decompression step at the end of generation. As a result, the efficiency of LZW-integrated tokenization is important to overall performance. To minimize overhead, we implemented a Rust-based `zip2zip` tokenizer that outperforms the Python version (see Figure 4) and matches the latency of HuggingFace's fast BPE tokenizer.

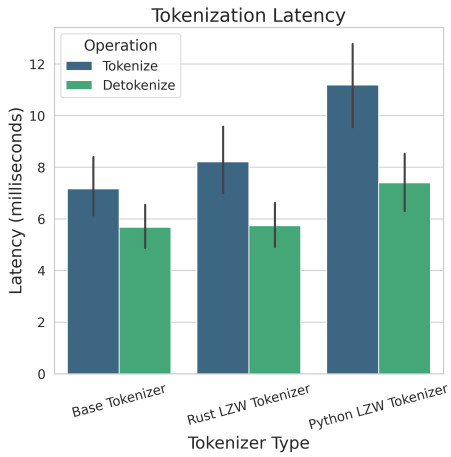


Figure 4: `zip2zip` tokenizer latency (ms) vs. HF tokenizer.

Table 6: **Throughput (tokens/sec)** comparison of the `zip2zip` framework against the baseline Hugging Face Transformers `generate` and MLX `generate` implementation. Performance is detailed for prefill and decode phases across various context lengths (first value in column headers) combined with a 256-token generation length. `zip2zip` demonstrates notable throughput improvements, in both prefill and decoding phase.

Setting	Method	256+256		512+256		1024+256		2048+256	
		Prefill	**Decode**	**Prefill**	**Decode**	**Prefill**	**Decode**	**Prefill**	**Decode**
Hardware: Apple M1 (16GB RAM)									
Phi-3-4B	Base model	165.0	7.3	211.3	7.5	200.9	7.1	196.6	6.8
	zip2zip	145.5	7.9	231.4	10.1	189.6	7.4	233.8	7.3
	Relative %	-11.8%	+7.5%	+9.5%	+34.8%	-6.6%	+3.9%	+18.9%	+7.5%
Hardware: NVIDIA H100 80GB GPU									
Phi-3.5-4B	Base model	700.9	56.2	1347.2	54.4	2689.4	52.8	4993.2	53.1
	zip2zip	936.6	61.4	2722.1	79.8	4326.7	61.5	9258.1	61.9
	Relative %	+33.6%	+9.3%	+102.6%	+46.6%	+60.9%	+16.6%	+85.4%	+16.5%
Phi-3-14B	Base model	724.4	44.6	1356.3	43.8	2328.6	45.1	3849.5	42.2
	zip2zip	1024.6	54.9	1973.0	61.1	3657.0	66.8	7239.1	46.3
	Relative %	+41.5%	+23.0%	+45.5%	+39.5%	+57.0%	+48.1%	+88.1%	+9.6%

4. Related Work

Vocabulary Expansion. Several works have explored expanding the tokenizer vocabulary to better support specific domains or languages. Zhao et al. (2024); Kim et al. (2024); Liu et al. (2023; 2024) adapt LLaMA to Chinese, Korean, and specialized domains such as mental health and law by appending new tokens. Wang et al. (2025); Liu et al. (2024) conducted studies on how to effectively expand the vocabulary by better selecting the subset of tokens to add. In contrast, `zip2zip` is the first to enable *dynamic vocabulary expansion at inference time*, constructing new tokens based on the input context without requiring retraining or modifying the tokenizer ahead of time.

Prompt Compression. Prompt compression methods include GistTokens (Mu et al., 2023), Selective Context (Li et al., 2023), LLMLingua (Jiang et al., 2023), Summary Vectors (Chevalier et al., 2023), In-context Autoencoder (Ge et al., 2024), and others (Wingate et al., 2022) reduce the input token length and but do not impact the number of output tokens, which often dominates overall generation time. In contrast, `zip2zip` compresses *both* the input and output token sequences.

Latent Tokens Representation. The concept of latent token representations, or *patches*, has been mostly explored in computer vision, with methods like Token Merging (Bolya et al., 2023) and Token Pooling (Marin et al., 2023) aiming to reduce sequence length while preserving semantic content. Recently, Byte Latent Transformer (BLT) (Pagnoni et al., 2024) extended this concept to language modeling by discarding tokens entirely and operating directly at the byte level. Both BLT and `zip2zip` adopt a hierarchical modeling of input for LLMs, but they differ in three key ways: (1) **Goal**: BLT aims to replace the tokenizer, whereas `zip2zip` seeks to expand and improve it; (2) **Algorithm**: BLT uses entropy-based segmentation, while `zip2zip` applies LZW-based token compression; (3) **Training**: BLT requires training from scratch, whereas `zip2zip` enables continued adaptation of pretrained models. Lester et al. (2024) propose improving language model efficiency by training LLMs directly on text compressed with arithmetic coding. While both approaches leverage compression to enhance efficiency, `zip2zip` emphasizes dynamic vocabulary expansion to enable uptraining of existing models. In contrast, Lester et al. (2024) requires training from scratch.

5. Discussion and Limitations

Beyond LZW. While we adopt LZW for dynamic construction of hypertokens, `zip2zip` is broadly compatible with any online compression algorithm. Future work may explore alternative schemes that provide different trade-offs between compression efficiency and model performance.

Codebook Management Strategy. The LZW algorithm grows the codebook linearly with the number of tokens in the context window. Empirical results show that only about 25% of hypertokens are reused during generation, leaving substantial room for optimization. Two potential improvements are: (1) **pruning** or **selective retention** strategies to reduce unused entries, and (2) **codebook prefilling**, which could be beneficial if likely tokens can be anticipated before input processing.

Compression–Quality Trade-off. There is an inherent trade-off between compression and modeling: as the token space is compressed more aggressively, redundancy is reduced—but so is predictability—making it harder for

the model to forecast the next (hyper)token. In the extreme, optimal compression schemes such as arithmetic coding produce sequences that are statistically indistinguishable from random noise, rendering them unlearnable by language models (Lester et al., 2024). Empirically, we observe this effect as increased perplexity under higher compression levels (Table 3), which can undermine the benefits of compression by degrading generation quality (though minor in the tasks in Table **??** and Table 5). Striking the right balance between compression and model performance remains an important direction for future research.

6. Conclusion

We introduced `zip2zip`, a framework for inference-time vocabulary adaptation with LLMs. By integrating LZW-based token compression with a dynamic hypertoken embedding mechanism, `zip2zip` enables substantial reductions in sequence length and decoding steps, leading to improved inference efficiency with minimal architectural modifications. Our experiments demonstrate that `zip2zip` maintains strong performance across a range of tasks while achieving significant gains in inference efficiency. These findings highlight the promise of integrating dynamic tokenization into LLMs, opening up new directions for research in LLM efficiency.

References

Abdin, M., Aneja, J., Behl, H., Bubeck, S., Eldan, R., Gunasekar, S., Harrison, M., Hewett, R. J., Javaheripi, M., Kauffmann, P., Lee, J. R., Lee, Y. T., Li, Y., Liu, W., Mendes, C. C. T., Nguyen, A., Price, E., de Rosa, G., Saarikivi, O., Salim, A., Shah, S., Wang, X., Ward, R., Wu, Y., Yu, D., Zhang, C., and Zhang, Y. Phi-4 technical report, 2024. URL https://arxiv.org/abs/2412.08905.

Ahia, O., Kumar, S., Gonen, H., Kasai, J., Mortensen, D., Smith, N., and Tsvetkov, Y. Do all languages cost the same? tokenization in the era of commercial language models. In Bouamor, H., Pino, J., and Bali, K. (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/.

Bisk, Y., Zellers, R., Bras, R. L., Gao, J., and Choi, Y. Piqa: Reasoning about physical commonsense in natural language. In *AAAI Conference on Artificial Intelligence*, 2019. URL https://api.semanticscholar.org/CorpusID:208290939.

Bojar, O., Chatterjee, R., Federmann, C., Graham, Y., Haddow, B., Huck, M., Jimeno Yepes, A., Koehn, P., Logacheva, V., Monz, C., Negri, M., Névéol, A., Neves, M., Popel, M., Post, M., Rubino, R., Scarton, C., Specia, L., Turchi, M., Verspoor, K., and Zampieri, M. Findings of the 2016 conference on machine translation. In Bojar, O., Buck, C., Chatterjee, R., Federmann, C., Guillou, L., Haddow, B., Huck, M., Yepes, A. J., Névéol, A., Neves, M., Pecina, P., Popel, M., Koehn, P., Monz, C., Negri, M., Post, M., Specia, L., Verspoor, K., Tiedemann, J., and Turchi, M. (eds.), *Proceedings of the First Conference on Machine Translation: Volume 2, Shared Task Papers*, pp. 131–198, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/W16-2301. URL https://aclanthology.org/W16-2301/.

Bolya, D., Fu, C.-Y., Dai, X., Zhang, P., Feichtenhofer, C., and Hoffman, J. Token merging: Your vit but faster, 2023. URL https://arxiv.org/abs/2210.09461.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. Language models are few-shot learners. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H. (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 1877–1901. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

Bubeck, S., Chandrasekaran, V., Eldan, R., Gehrke, J., Horvitz, E., Kamar, E., Lee, P., Lee, Y. T., Li, Y., Lundberg, S., Nori, H., Palangi, H., Ribeiro, M. T., and Zhang, Y. Sparks of artificial general intelligence: Early experiments with gpt-4, 2023. URL https://arxiv.org/abs/2303.12712.

Chevalier, A., Wettig, A., Ajith, A., and Chen, D. Adapting language models to compress contexts. In Bouamor, H., Pino, J., and Bali, K. (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/.

Clark, P., Cowhey, I., Etzioni, O., Khot, T., Sabharwal, A., Schoenick, C., and Tafjord, O. Think you have solved question answering? try arc, the ai2 reasoning challenge, 2018. URL https://arxiv.org/abs/1803.05457.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., Hesse, C., and Schulman, J. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Dagan, G., Synnaeve, G., and Rozière, B. Getting the most out of your tokenizer for pre-training and domain adaptation, 2024. URL https://arxiv.org/abs/2402.01035.

Ding, N., Chen, Y., Xu, B., Qin, Y., Zheng, Z., Hu, S., Liu, Z., Sun, M., and Zhou, B. Enhancing chat language models by scaling high-quality instructional conversations, 2023.

Frieder, S., Pinchetti, L., Chevalier, A., Griffiths, R.-R., Salvatori, T., Lukasiewicz, T., Petersen, P. C., and Berner, J. Mathematical capabilities of chatgpt, 2023. URL https://arxiv.org/abs/2301.13867.

Gao, L., Biderman, S., Black, S., Golding, L., Hoppe, T., Foster, C., Phang, J., He, H., Thite, A., Nabeshima, N., Presser, S., and Leahy, C. The pile: An 800gb dataset of diverse text for language modeling, 2020. URL https://arxiv.org/abs/2101.00027.

Gao, L., Tow, J., Abbasi, B., Biderman, S., Black, S., DiPofi, A., Foster, C., Golding, L., Hsu, J., Le Noac'h, A., Li, H., McDonell, K., Muennighoff, N., Ociepa, C., Phang, J., Reynolds, L., Schoelkopf, H., Skowron, A., Sutawika, L., Tang, E., Thite, A., Wang, B., Wang, K., and Zou, A. The language model evaluation harness, 07 2024. URL https://zenodo.org/records/12608602.

Ge, T., Jing, H., Wang, L., Wang, X., Chen, S.-Q., and Wei, F. 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.

Grattafiori, A., Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Vaughan, A., Yang, A., Fan, A., Goyal, A., Hartshorn, A., Yang, A., Mitra, A., Sravankumar, A., Korenev, A., Hinsvark, A., Rao, A., Zhang, A., Rodriguez, A., Gregerson, A., Spataru, A., Roziere, B., Biron, B., Tang, B., Chern, B., Caucheteux, C., Nayak, C., Bi, C., Marra, C., McConnell, C., Keller, C., Touret, C., Wu, C., Wong, C., Ferrer, C. C., Nikolaidis, C., Allonsius, D., Song, D., Pintz, D., Livshits, D., Wyatt, D., Esiobu, D., Choudhary, D., Mahajan, D., Garcia-Olano, D., Perino, D., Hupkes, D., Lakomkin, E., AlBadawy, E., Lobanova, E., Dinan, E., Smith, E. M., Radenovic, F., Guzmán, F., Zhang, F., Synnaeve, G., Lee, G., Anderson, G. L., Thattai, G., Nail, G., Mialon, G., Pang, G., Cucurell, G., Nguyen, H., Korevaar, H., Xu, H., Touvron, H., Zarov, I., Ibarra, I. A., Kloumann, I., Misra, I., Evtimov, I., Zhang, J., Copet, J., Lee, J., Geffert, J., Vranes, J., Park, J., Mahadeokar, J., Shah, J., van der Linde, J., Billock, J., Hong, J., Lee, J., Fu, J., Chi, J., Huang, J., Liu, J., Wang, J., Yu, J., Bitton, J., Spisak, J., Park, J., Rocca, J., Johnstun, J., Saxe, J., Jia, J., Alwala, K. V., Prasad, K., Upasani, K., Plawiak, K., Li, K., Heafield, K., Stone, K., El-Arini, K., Iyer, K., Malik, K., Chiu, K., Bhalla, K., Lakhotia, K., Rantala-Yeary, L., van der Maaten, L., Chen, L., Tan, L., Jenkins, L., Martin, L., Madaan, L., Malo, L., Blecher, L., Landzaat, L., de Oliveira, L., Muzzi, M., Pasupuleti, M., Singh, M., Paluri, M., Kardas, M., Tsimpoukelli, M., Oldham, M., Rita, M., Pavlova, M., Kambadur, M., Lewis, M., Si, M., Singh, M. K., Hassan, M., Goyal, N., Torabi, N., Bashlykov, N., Bogoychev, N., Chatterji, N., Zhang, N., Duchenne, O., Çelebi, O., Alrassy, P., Zhang, P., Li, P., Vasic, P., Weng, P., Bhargava, P., Dubal, P., Krishnan, P., Koura, P. S., Xu, P., He, Q., Dong, Q., Srinivasan, R., Ganapathy, R., Calderer, R., Cabral, R. S., Stojnic, R., Raileanu, R., Maheswari, R., Girdhar, R., Patel, R., Sauvestre, R., Polidoro, R., Sumbaly, R., Taylor, R., Silva, R., Hou, R., Wang, R., Hosseini, S., Chennabasappa, S., Singh, S., Bell, S., Kim, S. S., Edunov, S., Nie, S., Narang, S., Raparthy, S., Shen, S., Wan, S., Bhosale, S., Zhang, S., Vandenhende, S., Batra, S., Whitman, S., Sootla, S., Collot, S., Gururangan, S., Borodinsky, S., Herman, T., Fowler, T., Sheasha, T., Georgiou, T., Scialom, T., Speckbacher, T., Mihaylov, T., Xiao, T., Karn, U., Goswami, V., Gupta, V., Ramanathan, V., Kerkez, V., Gonguet, V., Do, V., Vogeti, V., Albiero, V., Petrovic, V., Chu, W., Xiong, W., Fu, W., Meers, W., Martinet, X., Wang, X., Wang, X., Tan, X. E., Xia, X., Xie, X., Jia, X., Wang, X., Goldschlag, Y., Gaur, Y., Babaei, Y., Wen, Y., Song, Y., Zhang, Y., Li, Y., Mao, Y., Coudert, Z. D., Yan, Z., Chen, Z., Papakipos, Z., Singh, A., Srivastava, A., Jain, A., Kelsey, A., Shajnfeld, A., Gangidi, A., Victoria, A., Goldstand, A., Menon, A., Sharma, A., Boesenberg, A., Baevski, A., Feinstein, A., Kallet, A., Sangani, A., Teo, A., Yunus, A., Lupu, A., Alvarado, A., Caples, A., Gu, A., Ho, A., Poulton, A., Ryan, A., Ramchandani, A., Dong, A., Franco, A., Goyal, A., Saraf, A., Chowdhury, A., Gabriel, A., Bharambe, A., Eisenman, A., Yazdan, A., James, B., Maurer, B., Leonhardi, B., Huang, B., Loyd, B., Paola, B. D., Paranjape, B., Liu, B., Wu, B., Ni, B., Hancock, B., Wasti, B., Spence, B., Stojkovic, B., Gamido, B., Montalvo, B., Parker, C., Burton, C., Mejia, C., Liu, C., Wang, C., Kim, C., Zhou, C., Hu, C., Chu, C.-H., Cai, C., Tindal, C., Feichtenhofer, C., Gao, C., Civin, D., Beaty, D., Kreymer, D., Li, D., Adkins, D., Xu, D., Testuggine, D., David, D., Parikh, D., Liskovich, D., Foss, D., Wang, D., Le, D., Holland, D., Dowling, E., Jamil, E., Montgomery, E., Presani, E., Hahn, E., Wood, E., Le, E.-T., Brinkman, E., Arcaute, E., Dunbar, E., Smothers, E., Sun, F., Kreuk, F., Tian, F., Kokkinos, F., Ozgenel, F., Cag-

gioni, F., Kanayet, F., Seide, F., Florez, G. M., Schwarz, G., Badeer, G., Swee, G., Halpern, G., Herman, G., Sizov, G., Guangyi, Zhang, Lakshminarayanan, G., Inan, H., Shojanazeri, H., Zou, H., Wang, H., Zha, H., Habeeb, H., Rudolph, H., Suk, H., Aspegren, H., Goldman, H., Zhan, H., Damlaj, I., Molybog, I., Tufanov, I., Leontiadis, I., Veliche, I.-E., Gat, I., Weissman, J., Geboski, J., Kohli, J., Lam, J., Asher, J., Gaya, J.-B., Marcus, J., Tang, J., Chan, J., Zhen, J., Reizenstein, J., Teboul, J., Zhong, J., Jin, J., Yang, J., Cummings, J., Carvill, J., Shepard, J., McPhie, J., Torres, J., Ginsburg, J., Wang, J., Wu, K., U, K. H., Saxena, K., Khandelwal, K., Zand, K., Matosich, K., Veeraraghavan, K., Michelena, K., Li, K., Jagadeesh, K., Huang, K., Chawla, K., Huang, K., Chen, L., Garg, L., A, L., Silva, L., Bell, L., Zhang, L., Guo, L., Yu, L., Moshkovich, L., Wehrstedt, L., Khabsa, M., Avalani, M., Bhatt, M., Mankus, M., Hasson, M., Lennie, M., Reso, M., Groshev, M., Naumov, M., Lathi, M., Keneally, M., Liu, M., Seltzer, M. L., Valko, M., Restrepo, M., Patel, M., Vyatskov, M., Samvelyan, M., Clark, M., Macey, M., Wang, M., Hermoso, M. J., Metanat, M., Rastegari, M., Bansal, M., Santhanam, N., Parks, N., White, N., Bawa, N., Singhal, N., Egebo, N., Usunier, N., Mehta, N., Laptev, N. P., Dong, N., Cheng, N., Chernoguz, O., Hart, O., Salpekar, O., Kalinli, O., Kent, P., Parekh, P., Saab, P., Balaji, P., Rittner, P., Bontrager, P., Roux, P., Dollar, P., Zvyagina, P., Ratanchandani, P., Yuvraj, P., Liang, Q., Alao, R., Rodriguez, R., Ayub, R., Murthy, R., Nayani, R., Mitra, R., Parthasarathy, R., Li, R., Hogan, R., Battey, R., Wang, R., Howes, R., Rinott, R., Mehta, S., Siby, S., Bondu, S. J., Datta, S., Chugh, S., Hunt, S., Dhillon, S., Sidorov, S., Pan, S., Mahajan, S., Verma, S., Yamamoto, S., Ramaswamy, S., Lindsay, S., Lindsay, S., Feng, S., Lin, S., Zha, S. C., Patil, S., Shankar, S., Zhang, S., Zhang, S., Wang, S., Agarwal, S., Sajuyigbe, S., Chintala, S., Max, S., Chen, S., Kehoe, S., Satterfield, S., Govindaprasad, S., Gupta, S., Deng, S., Cho, S., Virk, S., Subramanian, S., Choudhury, S., Goldman, S., Remez, T., Glaser, T., Best, T., Koehler, T., Robinson, T., Li, T., Zhang, T., Matthews, T., Chou, T., Shaked, T., Vontimitta, V., Ajayi, V., Montanez, V., Mohan, V., Kumar, V. S., Mangla, V., Ionescu, V., Poenaru, V., Mihailescu, V. T., Ivanov, V., Li, W., Wang, W., Jiang, W., Bouaziz, W., Constable, W., Tang, X., Wu, X., Wang, X., Wu, X., Gao, X., Kleinman, Y., Chen, Y., Hu, Y., Jia, Y., Qi, Y., Li, Y., Zhang, Y., Zhang, Y., Adi, Y., Nam, Y., Yu, Wang, Zhao, Y., Hao, Y., Qian, Y., Li, Y., He, Y., Rait, Z., DeVito, Z., Rosnbrick, Z., Wen, Z., Yang, Z., Zhao, Z., and Ma, Z. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Jiang, H., Wu, Q., Lin, C.-Y., Yang, Y., and Qiu, L. LLMLingua: Compressing prompts for accelerated inference of large language models. In Bouamor, H., Pino, J., and Bali, K. (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.

Jiang, J., Wang, F., Shen, J., Kim, S., and Kim, S. A survey on large language models for code generation, 2024. URL https://arxiv.org/abs/2406.00515.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. Scaling laws for neural language models, 2020. URL https://arxiv.org/abs/2001.08361.

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

Lester, B., Lee, J., Alemi, A., Pennington, J., Roberts, A., Sohl-Dickstein, J., and Constant, N. Training llms over neurally compressed text, 2024. URL https://arxiv.org/abs/2404.03626.

LI, J., Beeching, E., Tunstall, L., Lipkin, B., Soletskyi, R., Huang, S. C., Rasul, K., Yu, L., Jiang, A., Shen, Z., Qin, Z., Dong, B., Zhou, L., Fleureau, Y., Lample, G., and Polu, S. Numinamath. [https://huggingface.co/AI-MO/NuminaMath-1.5](https://github.com/project-numina/aimo-progress-prize/blob/main/report/numina_dataset.pdf), 2024.

Li, Y., Dong, B., Guerin, F., and Lin, C. Compressing context to enhance inference efficiency of large language models. In Bouamor, H., Pino, J., and Bali, K. (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/.

Liang, D., Gonen, H., Mao, Y., Hou, R., Goyal, N., Ghazvininejad, M., Zettlemoyer, L., and Khabsa, M. XLM-V: Overcoming the vocabulary bottleneck in multilingual masked language models. In Bouamor, H., Pino, J., and Bali, K. (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 13142–13152, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/

2023.emnlp-main.813. URL https://aclanthology.org/2023.emnlp-main.813/.

Liu, C., Wang, S., Qing, L., Kuang, K., Kang, Y., Sun, C., and Wu, F. Gold panning in vocabulary: An adaptive method for vocabulary expansion of domain-specific LLMs. In Al-Onaizan, Y., Bansal, M., and Chen, Y.-N. (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 7442–7459, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.424. URL https://aclanthology.org/2024.emnlp-main.424/.

Liu, S., Deng, N., Sabour, S., Jia, Y., Huang, M., and Mihalcea, R. Task-adaptive tokenization: Enhancing long-form text generation efficacy in mental health and beyond. In Bouamor, H., Pino, J., and Bali, K. (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/.

Lozhkov, A., Ben Allal, L., von Werra, L., and Wolf, T. Fineweb-edu: the finest collection of educational content, 2024a. URL https://huggingface.co/datasets/HuggingFaceFW/fineweb-edu.

Lozhkov, A., Li, R., Allal, L. B., Cassano, F., Lamy-Poirier, J., Tazi, N., Tang, A., Pykhtar, D., Liu, J., Wei, Y., Liu, T., Tian, M., Kocetkov, D., Zucker, A., Belkada, Y., Wang, Z., Liu, Q., Abulkhanov, D., Paul, I., Li, Z., Li, W.-D., Risdal, M., Li, J., Zhu, J., Zhuo, T. Y., Zheltonozhskii, E., Dade, N. O. O., Yu, W., Krauß, L., Jain, N., Su, Y., He, X., Dey, M., Abati, E., Chai, Y., Muennighoff, N., Tang, X., Oblokulov, M., Akiki, C., Marone, M., Mou, C., Mishra, M., Gu, A., Hui, B., Dao, T., Zebaze, A., Dehaene, O., Patry, N., Xu, C., McAuley, J., Hu, H., Scholak, T., Paquet, S., Robinson, J., Anderson, C. J., Chapados, N., Patwary, M., Tajbakhsh, N., Jernite, Y., Ferrandis, C. M., Zhang, L., Hughes, S., Wolf, T., Guha, A., von Werra, L., and de Vries, H. Starcoder 2 and the stack v2: The next generation, 2024b.

Macháček, M. and Bojar, O. Results of the WMT14 metrics shared task. In Bojar, O., Buck, C., Federmann, C., Haddow, B., Koehn, P., Monz, C., Post, M., and Specia, L. (eds.), *Proceedings of the Ninth Workshop on Statistical Machine Translation*, pp. 293–301, Baltimore, Maryland, USA, June 2014. Association for Computational Linguistics. doi: 10.3115/v1/W14-3336. URL https://aclanthology.org/W14-3336/.

Magnusson, I., Bhagia, A., Hofmann, V., Soldaini, L., Jha, A., Tafjord, O., Schwenk, D., Walsh, P., Elazar, Y., Lo, K., Groeneveld, D., Beltagy, I., Hajishirzi, H., Smith, N. A., Richardson, K., and Dodge, J. Paloma: A benchmark for evaluating language model fit. *ArXiv*, abs/2312.10523, 2023. URL https://api.semanticscholar.org/CorpusID:266348815.

Marin, D., Chang, J.-H. R., Ranjan, A., Prabhu, A., Rastegari, M., and Tuzel, O. Token pooling in vision transformers for image classification. In *2023 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pp. 12–21, 2023. doi: 10.1109/WACV56688.2023.00010.

Merity, S., Xiong, C., Bradbury, J., and Socher, R. Pointer sentinel mixture models, 2016.

Mihaylov, T., Clark, P., Khot, T., and Sabharwal, A. Can a suit of armor conduct electricity? a new dataset for open book question answering. In Riloff, E., Chiang, D., Hockenmaier, J., and Tsujii, J. (eds.), *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2381–2391, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1260. URL https://aclanthology.org/D18-1260/.

Mu, J., Li, X., and Goodman, N. Learning to compress prompts with gist tokens. In Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., and Levine, S. (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.

Nori, H., King, N., McKinney, S. M., Carignan, D., and Horvitz, E. Capabilities of gpt-4 on medical challenge problems, 2023. URL https://arxiv.org/abs/2303.13375.

Pagnoni, A., Pasunuru, R., Rodriguez, P., Nguyen, J., Muller, B., Li, M., Zhou, C., Yu, L., Weston, J., Zettlemoyer, L., Ghosh, G., Lewis, M., Holtzman, A., and Iyer, S. Byte latent transformer: Patches scale better than tokens, 2024. URL https://arxiv.org/abs/2412.09871.

Paperno, D., Kruszewski, G., Lazaridou, A., Pham, N. Q., Bernardi, R., Pezzelle, S., Baroni, M., Boleda, G., and Fernández, R. The LAMBADA dataset: Word prediction requiring a broad discourse context. In Erk, K. and Smith, N. A. (eds.), *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1525–1534, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1144. URL https://aclanthology.org/P16-1144/.

Penedo, G., Kydlíček, H., Sabolčec, V., Messmer, B., Foroutan, N., Jaggi, M., von Werra, L., and Wolf, T. Fineweb2: A sparkling update with 1000s of languages, 2024. URL https://huggingface.co/datasets/HuggingFaceFW/fineweb-2.

Petrov, A., La Malfa, E., H. S. Torr, P., and Bibi, A. Language model tokenizers introduce unfairness between languages. In *Advances in Neural Information Processing Systems*, 2023. URL https://arxiv.org/abs/2305.15425.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. Language models are unsupervised multitask learners. *OpenAI blog*, 2019. URL https://api.semanticscholar.org/CorpusID:160025533.

Sakaguchi, K., Bras, R. L., Bhagavatula, C., and Choi, Y. Winogrande. *Communications of the ACM*, 64:99 – 106, 2019. URL https://api.semanticscholar.org/CorpusID:198893658.

Sennrich, R., Haddow, B., and Birch, A. Neural machine translation of rare words with subword units. In Erk, K. and Smith, N. A. (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/.

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

Sutskever, I., Vinyals, O., and Le, Q. V. Sequence to sequence learning with neural networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, pp. 3104–3112, Cambridge, MA, USA, 2014. MIT Press.

Team, G., Kamath, A., Ferret, J., Pathak, S., Vieillard, N., Merhej, R., Perrin, S., Matejovicova, T., Ramé, A., Rivière, M., Rouillard, L., Mesnard, T., Cideron, G., bastien Grill, J., Ramos, S., Yvinec, E., Casbon, M., Pot, E., Penchev, I., Liu, G., Visin, F., Kenealy, K., Beyer, L., Zhai, X., Tsitsulin, A., Busa-Fekete, R., Feng, A., Sachdeva, N., Coleman, B., Gao, Y., Mustafa, B., Barr, I., Parisotto, E., Tian, D., Eyal, M., Cherry, C., Peter, J.-T., Sinopalnikov, D., Bhupatiraju, S., Agarwal, R., Kazemi, M., Malkin, D., Kumar, R., Vilar, D., Brusilovsky, I., Luo, J., Steiner, A., Friesen, A., Sharma, A., Sharma, A., Gilady, A. M., Goedeckemeyer, A., Saade, A., Feng, A., Kolesnikov, A., Bendebury, A., Abdagic, A., Vadi, A., György, A., Pinto, A. S., Das, A., Bapna, A., Miech, A., Yang, A., Paterson, A., Shenoy, A., Chakrabarti, A., Piot, B., Wu, B., Shahriari, B., Petrini, B., Chen, C., Lan, C. L., Choquette-Choo, C. A., Carey, C., Brick, C., Deutsch, D., Eisenbud, D., Cattle, D., Cheng, D., Paparas, D., Sreepathihalli, D. S., Reid, D., Tran, D., Zelle, D., Noland, E., Huizenga, E., Kharitonov, E., Liu, F., Amirkhanyan, G., Cameron, G., Hashemi, H., Klimczak-Plucińska, H., Singh, H., Mehta, H., Lehri, H. T., Hazimeh, H., Ballantyne, I., Szpektor, I., Nardini, I., Pouget-Abadie, J., Chan, J., Stanton, J., Wieting, J., Lai, J., Orbay, J., Fernandez, J., Newlan, J., yeong Ji, J., Singh, J., Black, K., Yu, K., Hui, K., Vodrahalli, K., Greff, K., Qiu, L., Valentine, M., Coelho, M., Ritter, M., Hoffman, M., Watson, M., Chaturvedi, M., Moynihan, M., Ma, M., Babar, N., Noy, N., Byrd, N., Roy, N., Momchev, N., Chauhan, N., Sachdeva, N., Bunyan, O., Botarda, P., Caron, P., Rubenstein, P. K., Culliton, P., Schmid, P., Sessa, P. G., Xu, P., Stanczyk, P., Tafti, P., Shivanna, R., Wu, R., Pan, R., Rokni, R., Willoughby, R., Vallu, R., Mullins, R., Jerome, S., Smoot, S., Girgin, S., Iqbal, S., Reddy, S., Sheth, S., Põder, S., Bhatnagar, S., Panyam, S. R., Eiger, S., Zhang, S., Liu, T., Yacovone, T., Liechty, T., Kalra, U., Evci, U., Misra, V., Roseberry, V., Feinberg, V., Kolesnikov, V., Han, W., Kwon, W., Chen, X., Chow, Y., Zhu, Y., Wei, Z., Egyed, Z., Cotruta, V., Giang, M., Kirk, P., Rao, A., Black, K., Babar, N., Lo, J., Moreira, E., Martins, L. G., Sanseviero, O., Gonzalez, L., Gleicher, Z., Warkentin, T., Mirrokni, V., Senter, E., Collins, E., Barral, J., Ghahramani, Z., Hadsell, R., Matias, Y., Sculley, D., Petrov, S., Fiedel, N., Shazeer, N., Vinyals, O., Dean, J., Hassabis, D., Kavukcuoglu, K., Farabet, C., Buchatskaya, E., Alayrac, J.-B., Anil, R., Dmitry, Lepikhin, Borgeaud, S., Bachem, O., Joulin, A., Andreev, A., Hardin, C., Dadashi, R., and Hussenot, L. Gemma 3 technical report, 2025. URL https://arxiv.org/abs/2503.19786.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. Attention is all you need. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Wang, H., Yu, D., Sun, K., Chen, J., and Yu, D. Improving pre-trained multilingual model with vocabulary expansion. In Bansal, M. and Villavicencio, A. (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/.

Wang, S., Xie, Y., Ding, B., Gao, J., and Zhang, Y. Language adaptation of large language models: An em-

pirical study on LLaMA2. In Rambow, O., Wanner, L., Apidianaki, M., Al-Khalifa, H., Eugenio, B. D., and Schockaert, S. (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.

Wingate, D., Shoeybi, M., and Sorensen, T. Prompt compression and contrastive conditioning for controllability and toxicity reduction in language models. In Goldberg, Y., Kozareva, Z., and Zhang, Y. (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/.

Yang, A., Yang, B., Hui, B., Zheng, B., Yu, B., Zhou, C., Li, C., Li, C., Liu, D., Huang, F., Dong, G., Wei, H., Lin, H., Tang, J., Wang, J., Yang, J., Tu, J., Zhang, J., Ma, J., Xu, J., Zhou, J., Bai, J., He, J., Lin, J., Dang, K., Lu, K., Chen, K., Yang, K., Li, M., Xue, M., Ni, N., Zhang, P., Wang, P., Peng, R., Men, R., Gao, R., Lin, R., Wang, S., Bai, S., Tan, S., Zhu, T., Li, T., Liu, T., Ge, W., Deng, X., Zhou, X., Ren, X., Zhang, X., Wei, X., Ren, X., Fan, Y., Yao, Y., Zhang, Y., Wan, Y., Chu, Y., Liu, Y., Cui, Z., Zhang, Z., and Fan, Z. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2024.

Zellers, R., Holtzman, A., Bisk, Y., Farhadi, A., and Choi, Y. HellaSwag: Can a machine really finish your sentence? In Korhonen, A., Traum, D., and Màrquez, L. (eds.), *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4791–4800, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1472. URL https://aclanthology.org/P19-1472/.

Zhao, J., Zhang, Z., Gao, L., Zhang, Q., Gui, T., and Huang, X. Llama beyond english: An empirical study on language capability transfer, 2024. URL https://arxiv.org/abs/2401.01055.

A. Ablation Studies

Definition A.1 (Compression Rate)**.** *We define the* compression rate *as the ratio between the number of tokens after compression (N_{comp}) and the number of tokens in the original uncompressed text (N_{orig}), expressed as a percentage:*

$$\textit{Compression Rate} = \frac{N_{comp}}{N_{orig}} \times 100\%.$$

A lower compression rate indicates greater reduction in token count, and thus more effective compression.

A.1. LZW Maximum Merge Size

The last column of Table 7 shows how the maximum merge size M affects compression rate when the context window length is 2048. As M increases, compression rate improves significantly, especially from $M = 1$ to $M = 3$. Beyond that, gains diminish, suggesting $M = 3$ strikes a good balance between efficiency and compression rate.

Interestingly, the relationship between maximum merge size and training loss in Figure 5 as well as perplexity in Table 7 is non-monotonic. The baseline case with $M = 1$ (i.e., no zip2zip compression) yields the lowest perplexity overall, which is expected and consistent with prior findings that compression typically incurs a trade-off in model performance. Among the compressed settings, the case $M = 2$ performs the worst, with noticeably slower convergence and higher final loss. In contrast, the case $M = 3$ achieves the best performance within the compressed configurations, striking a favorable balance between compression and prediction performance. While $M = 4$ and $M = 5$ also perform reasonably well, they exhibit slightly higher loss than $M = 3$, suggesting diminishing returns or possible over-compression at larger maximum merge sizes (see Figure 5).

Table 7: **Effect of maximum merge size (M) on byte-level perplexity and compression rate.** Perplexity is measured for Phi-3.5-4B across four corpora with a 1024-token context window. Compression rate is evaluated over the training corpus with a 2048-token context. $M = 1$ corresponds to no compression.

M	Wiki	Pile	mC4	dC4	Compression Rate(%)
1	1.62	1.70	2.00	1.91	100.00
2	1.96	2.21	2.55	2.22	75.30
3	1.72	1.84	2.15	2.00	71.21
4	1.71	1.84	2.14	1.99	68.93
5	1.72	1.84	2.14	1.99	68.41

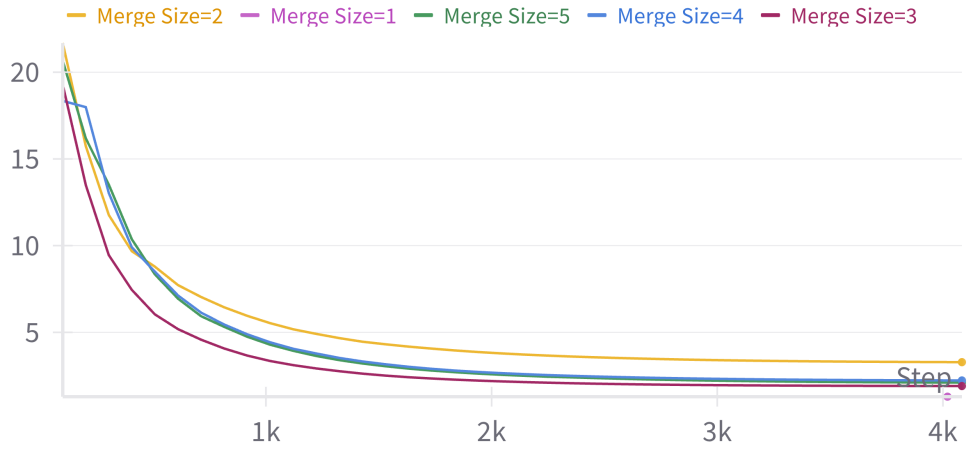


Figure 5: **Effect of maximum merge size M on zip2zip training loss**: $M = 1$ (no compression) achieves the lowest loss overall. Among compressed settings, $M = 3$ performs best, while $M = 2$ shows the worst convergence. Larger M (4 and 5) yield slightly worse results than $M = 3$.

Table 7 reports the byte-level perplexity across four corpora using a 1024-token context window. The results align closely with the training loss trends observed earlier. Setting $M = 1$ (i.e., no compression) consistently achieves the lowest perplexity across all datasets, reaffirming that compression introduces a performance trade-off. Notably, $M = 2$ performs the worst across all corpora, exhibiting the highest perplexity values. For merge sizes $M = 3$, $M = 4$, and $M = 5$, perplexity scores are nearly identical, suggesting that moderate compression can be achieved without significantly sacrificing

language modeling quality—provided $M = 2$ is avoided. This consistency across loss and perplexity metrics further supports the robustness of maximum merge size $M = 3$ as the most effective trade-off point.

A.2. Hyper-encoder architecture

We ablate the architecture of the hyper-encoder to evaluate its effect on language modeling performance, as shown in Table 8. We compare increasingly expressive architectures, starting with a simple averaging method that introduces no additional parameters. This baseline yields the highest perplexity, highlighting its limited capacity. Adding a single attention layer significantly improves performance, and further gains are observed with a 1-layer transformer encoder. The 2-layer transformer offers marginal additional benefit, suggesting that a lightweight transformer (1–2 layers) is sufficient for effective hyper-token modeling.

Table 8: **Ablation of hyper-encoder architecture** on byte-perplexity (↓) across four corpora using a 1024-token context window. Performance improves with increasingly expressive architectures.

Model	Method	Wiki	Pile	mC4	dC4
Phi-3.5-4B	averaging	1.81	1.97	2.29	2.08
	1-attention-layer	1.73	1.86	2.16	2.01
	1-transformer-layer	1.71	1.83	2.13	1.99
	2-transformer-layer	1.72	1.84	2.15	2.00

Figure 6 illustrates the effect of hyper-encoder architecture on zip2zip training loss. We observe that the simple averaging method converges the fastest but plateaus at a relatively high loss, reflecting its limited capacity. As model complexity increases—with attention and transformer layers—the convergence becomes slower, yet the final loss is significantly lower. Notably, the 1-layer and 2-layer transformer encoders yield the best performance, demonstrating that additional parameters enable the model to better capture structure, albeit at the cost of slower training dynamics.

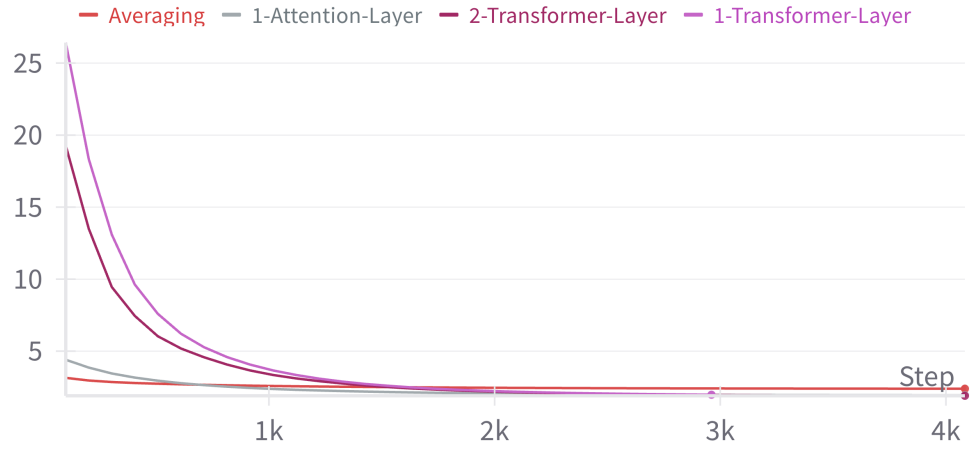


Figure 6: **Effect of hyper-encoder architecture on zip2zip training loss.** Averaging (no additional parameters) converges quickly but to a higher loss. As architectural complexity increases—from attention to transformer layers—convergence becomes slower, but the final loss is lower. This highlights a trade-off between training speed and modeling capacity.

B. FLOPs Estimation for `zip2zip`

Following the assumptions of Kaplan et al. (2020), we estimate training FLOPs (Γ) as:

$$\Gamma \approx 6 \cdot N_{\text{tokens}} \cdot N_{\text{params}},$$

where N_{tokens} is the total number of processed tokens and N_{params} is the number of trainable parameters. This estimate ignores the quadratic attention cost, assuming:

$$12 \cdot d_{\text{model}} \ll \text{sequence length}.$$

For `zip2zip`, this becomes:

$$\Gamma_{\text{z2z}} \approx 6 \cdot N_{\text{tokens}} \cdot \rho \cdot N_{\text{params}}(1+\alpha),$$

where ρ is the compression ratio, and α accounts for the overhead of the hyper-encoder applied at the embedding and LM head. The relative FLOPs ratio is then:

$$\frac{\Gamma_{\text{z2z}}}{\Gamma} = \rho \cdot (1+\alpha).$$

Assuming the hyper-encoder mirrors the base model's configuration, we estimate:

$$\alpha \approx \frac{lM}{L},$$

where l is the number of hyper-encoder layers, M is the maximum merge size, and L is the number of base model layers. We illustrate this estimate across several model scales in Table 9, showing that the relative FLOPs overhead from the hyper-module remains modest (typically under 15%).

Model	**L**	**M**	**l**	$\alpha = \frac{lM}{L}$
LLM-4B	14	2	1	0.14
LLM-7B	32	2	2	0.13
LLM-70B	80	3	3	0.11
LLM-400B	128	3	4	0.09

Table 9: Relative FLOPs overhead from the hyper-module across different model sizes.

C. Additional Results

Machine Translation

We report standard deviations for machine translation results across WMT benchmarks in Table 10, computed using the lm-evaluation-harness codebase.

Table 10: Standard deviations (bootstrapped) for machine translation scores across WMT benchmarks.

Model	**Method**	**WMT14 En-Fr**			**WMT16 En-De**			**WMT16 En-Ro**		
		BLEU	CHRF	TER	BLEU	CHRF	TER	BLEU	CHRF	TER
Phi-3-4B	+Raw	±2.1	±1.4	±1.7	±1.9	±1.6	±1.8	±1.5	±1.3	±2.4
	+Finetune	±2.2	±1.6	±1.8	±1.8	±1.4	±1.7	±1.4	±1.5	±2.3
	+zip2zip	±1.9	±1.5	±2.0	±1.7	±1.6	±1.9	±1.6	±1.4	±2.5
Phi-3-14B	+Raw	±2.0	±1.4	±1.9	±2.0	±1.5	±1.7	±1.5	±1.4	±2.2
	+Finetune	±2.2	±1.4	±1.9	±2.0	±1.3	±1.9	±1.4	±1.3	±2.9
	+zip2zip	±2.1	±1.5	±1.8	±2.1	±1.6	±1.8	±1.5	±1.3	±2.6

D. Technical Details

Model and Training Configuration

- **Pretrained Model:** `microsoft/Phi-3-medium-4k-instruct`
- **Sequence Length:** 1024
- **Total Batch Size:** 32,768 tokens
- **Learning Rate Schedule:** Cosine decay
- **Learning Rate Range:** Max = 3e-4, Min = 1e-5

- **LoRA rank and alpha value:** Both are 32
- **Training Steps:** 10,000
- **Validation Interval:** Every 100 steps
- **Checkpoint Interval:** Every 500 steps
- **Pytorch Model Compilation:** Enabled

LoRA Configuration

- **Rank:** 16
- **Alpha:** 16
- **Target Modules:** `qkv_proj, o_proj, gate_proj, down_proj, up_proj`

System and Libraries

- **Hardware:** 4 × NVIDIA A100-SXM4-80GB GPUs, 64-core CPU (128 threads)
- **Key Libraries:**
  - `PyTorch >= 2.5.0`
  - `Transformers >= 4.47.0`
  - `Datasets <= 3.1.0`
  - `Accelerate >= 0.26.0`

Compute Resources

We report the compute resources used for training our models in Table 11. All training was conducted on internal servers equipped with NVIDIA H100 GPUs. We estimate GPU-hours by multiplying wall-clock training time by the number of GPUs used. No additional compute was used beyond the reported experiments; we did not perform parameter grid search, large-scale hyperparameter tuning, or exploratory runs that were excluded from the paper.

Table 11: Training compute resources for zip2zip experiments.

Model	GPUs	Time	GPU Type	GPU-Hours
Phi-3.5-Medium (14B)	4	15h 46m	NVIDIA H100 80GB	63.0
Phi-3.5-Mini (4B)	2	7h 0m	NVIDIA H100 80GB	14.0

Inference. All evaluations complete within 1 hour on a single A100 GPU, demonstrating the runtime efficiency of `zip2zip`.

E. Data Mixture

To support effective fine-tuning, we construct a curated dataset with balanced representation across diverse domains, including code, mathematics, dialogue, general web content, and multilingual text. The final dataset contains approximately 1 billion compressed tokens.

Table 12 summarizes the constituent datasets and their respective proportions. A visualization of the dataset composition and sequence length characteristics is shown in Figure 7.

The multilingual subset in `fineweb-2` includes the following languages: Mandarin Chinese (`cmn_Hani`), German (`deu_Latn`), Japanese (`jpn_Jpan`), Spanish (`spa_Latn`), French (`fra_Latn`), Italian (`ita_Latn`), Portuguese (`por_Latn`), Dutch (`nld_Latn`), and Arabic (`arb_Arab`).

F. Token Stream Visualization

Dataset	Domain	Proportion (%)
`HuggingFaceFW/fineweb-edu`(Lozhkov et al., 2024a)	Web / Knowledge	20%
`devngho/the-stack-llm-annotations-v2`(Lozhkov et al., 2024b)	Code	25%
`AI-MO/NuminaMath-1.5`(LI et al., 2024)	Math	20%
`HuggingFaceH4/ultrachat_200k`(Ding et al., 2023)	Chat / Dialogue	20%
`HuggingFaceFW/fineweb-2`(Penedo et al., 2024)	Multilingual	15%

Table 12: Training data composition across domains.

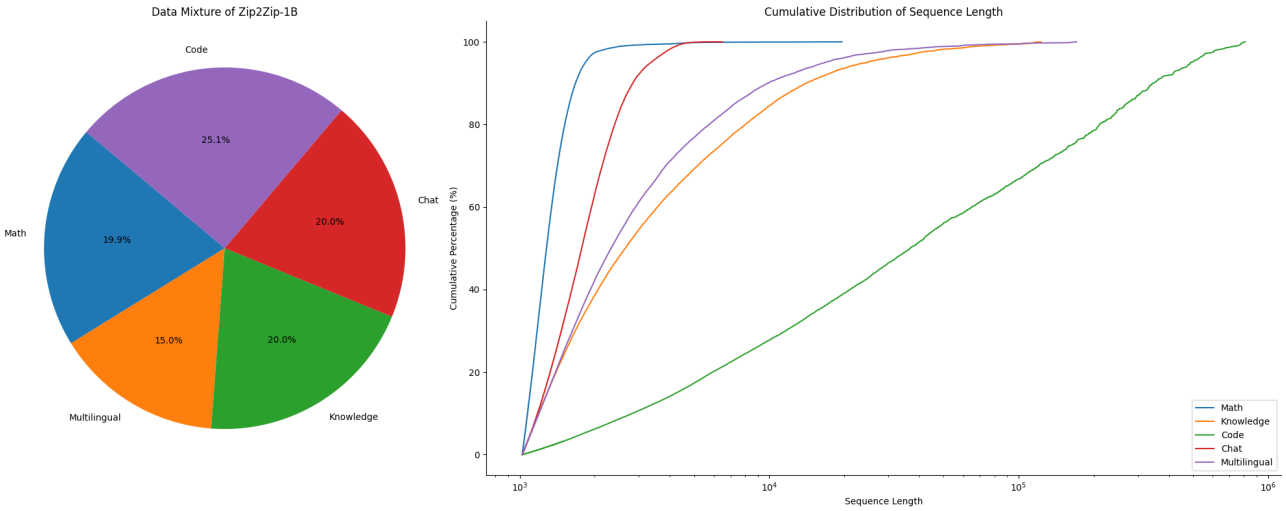
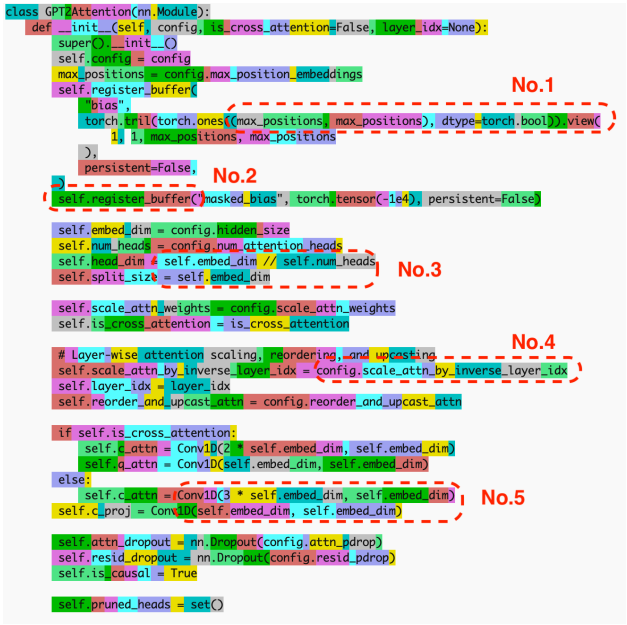


Figure 7: Left: Proportional breakdown of the fine-tuning dataset across five domains. Right: Cumulative distribution of input sequence lengths per domain (log scale). Code and multilingual data exhibit longer tail distributions, indicating greater variability in sequence lengths.

(a) Default Tokenization of some Python code.



(b) The same code with adaptive tokenization.

(c) Default Tokenization of some biomedical text.

(d) The same text with adaptive tokenization.

(e) Default Tokenization of text in French.

(f) The same text with adaptive tokenization.

Figure 8: Examples comparing default and adaptive tokenization. Dotted-line frames highlight where the differences are most noticeable.