# SuperBPE: Space Travel for Language Models

***Alisa Liu**[♡♠] ***Jonathan Hayase**[♡]
**Valentin Hofmann**[♢♡] **Sewoong Oh**[♡] **Noah A. Smith**[♡♢] **Yejin Choi**[♠]
[♡]University of Washington [♠]NVIDIA [♢]Allen Institute for AI



## Abstract

The assumption across nearly all language model (LM) tokenization schemes is that tokens should be *subwords*, i.e., contained within word boundaries. While providing a seemingly reasonable inductive bias, is this common practice limiting the potential of modern LMs? Whitespace is not a reliable delimiter of meaning, as evidenced by multi-word expressions (e.g., *by the way*), crosslingual variation in the number of words needed to express a concept (e.g., *spacesuit helmet* in German is *Raumanzughelm*), and languages that do not use whitespace at all (e.g., Chinese). To explore the potential of tokenization beyond subwords, we introduce a "superword" tokenizer, **SuperBPE**, which incorporates a simple pretokenization curriculum into the byte-pair encoding (BPE) algorithm to first learn subwords, then superwords that bridge whitespace. This brings dramatic improvements in encoding efficiency: when fixing the vocabulary size to 200k, SuperBPE encodes a fixed piece of text with up to 33% fewer tokens than BPE on average. In experiments, we pretrain 8B transformer LMs from scratch while fixing the model size, vocabulary size, and train compute, varying *only* the algorithm for learning the vocabulary. Our model trained with SuperBPE achieves an average +4.0% absolute improvement over the BPE baseline across 30 downstream tasks (including +8.2% on MMLU), while simultaneously requiring 27% less compute at inference time. In analysis, we find that SuperBPE results in segmentations of text that are more uniform in per-token difficulty. Qualitatively, this may be because SuperBPE tokens often capture common multi-word expressions that function semantically as a single unit. SuperBPE is a straightforward, local modification to tokenization that improves both encoding efficiency and downstream performance, yielding better language models overall.[1]

## 1 Introduction

Tokenizers are the lens through which language models (LMs) view data: they segment a stream of bytes into a sequence of tokens in the LM vocabulary. In the era of transformer LMs, tokenization is done at the level of subwords, meaning that tokens consist of *parts* of words (including complete words), but cannot bridge whitespace. Intuitively, subword tokens capture meaningful and composable semantic units.

While seemingly reasonable, is this common practice a good one? Whitespace is an unreliable delimiter of meaning (Martin, 2017)—many groups of words (e.g., *a lot of* or *search engine*) function semantically as single units, and English speakers store thousands of such *multi-word expressions* in their mental lexicon (Church, 2011; Contreras Kallens & Christiansen, 2022). Crosslingually, there is considerable variation in whether a given meaning is conveyed by a single word or several words. At the extreme, languages such as Chinese and Japanese do not use whitespace at all, so that tokens in these languages can span multiple words or even entire sentences (e.g., in the tokenizers of GPT-4O [OpenAI, 2024] or DEEPSEEKV3 [DeepSeek-AI et al., 2025]), yet this has seemingly not hindered LMs from

[*]Equal contribution.
[1]Code and artifacts are available at https://superbpe.github.io/.

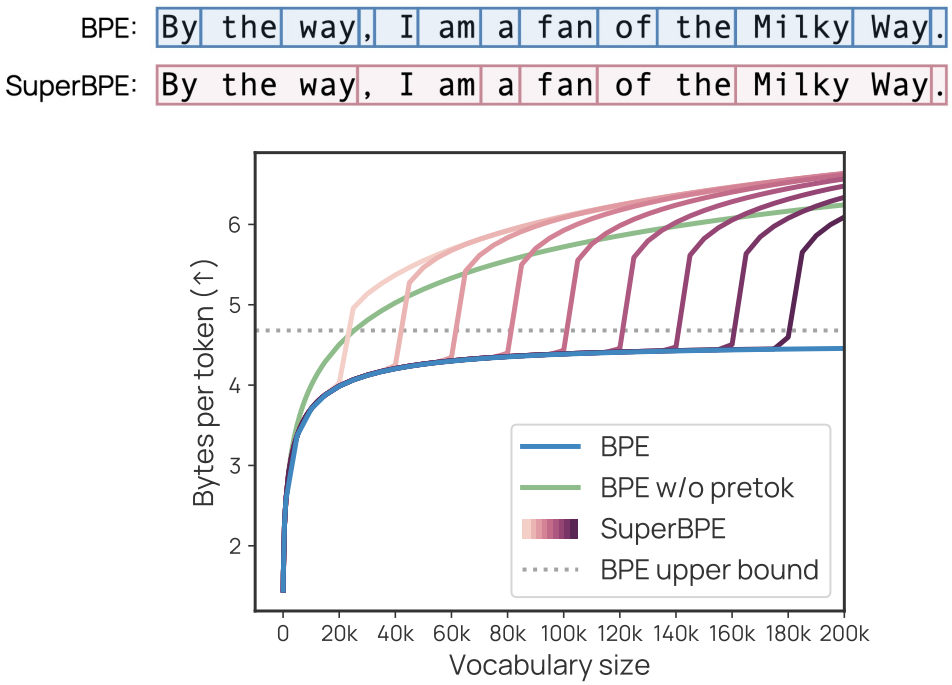



Figure 1: **SuperBPE tokenizers encode text much more efficiently than BPE, and the gap grows with larger vocabulary size.** Encoding efficiency (*y*-axis) is measured with *bytes-per-token*, the number of bytes encoded per token on average over a large corpus of text. In the above text with 40 bytes, SuperBPE uses 7 tokens and BPE uses 13, so the methods' efficiencies are $40/7 = 5.7$ and $40/13 = 3.1$ bytes-per-token, respectively. In the graph, the encoding efficiency of **BPE** plateaus early due to exhausting the valuable whitespace-delimited words in the training data. In fact, it is bounded above by the gray dotted line, which shows the *maximum* achievable encoding efficiency with BPE, if every whitespace-delimited word were in the vocabulary. On the other hand, **SuperBPE** has dramatically better encoding efficiency that continues to improve with increased vocabulary size, as it can continue to add common word *sequences* to treat as tokens to the vocabulary. The different gradient lines show different transition points from learning subword to superword tokens, which always gives an immediate improvement. SuperBPE also has better encoding efficiency than a **naive variant of BPE** that does not use whitespace pretokenization at all.

performing well on these languages. Including multi-word tokens promises to be beneficial in several ways: it can lead to shorter token sequences, lowering the computational costs of LM training and inference, and may also offer representational advantages by segmenting text into more semantically cohesive units (Salehi et al., 2015; Otani et al., 2020; Hofmann et al., 2021).

In this work, we introduce a *superword tokenization* algorithm that produces a vocabulary of both subword and "superword" tokens, which we use to refer to tokens that bridge more than one word. Our method, **SuperBPE**, introduces a *pretokenization curriculum* to the popular byte-pair encoding (BPE) algorithm (Sennrich et al., 2016): whitespace pretokenization is initially used to enforce learning of subword tokens only (as done in conventional BPE), but is disabled in a second stage, where the tokenizer transitions to learning superword tokens. Notably, SuperBPE tokenizers scale much better with vocabulary size—while BPE quickly hits a point of diminishing returns and begins adding increasingly rare subwords to the vocabulary, SuperBPE can continue to discover common word *sequences* to treat as single tokens and improve encoding efficiency (see Figure 1).

In our main experiments, we pretrain English LMs at 8B scale from scratch. When fixing the model size, vocabulary size, and training compute—varying only the algorithm for learning the vocabulary—we find that models trained with SuperBPE tokenizers consistently and significantly improve over counterparts trained with a BPE tokenizer, *while also being 27–33% more efficient at inference time*. Our best SuperBPE model achieves an average +4.0%

improvement over 30 downstream tasks, including +8.2% on MMLU, and wins on 25 of the 30 individual tasks (Table 1).

In analysis, we find that SuperBPE tokenizers produce segmentations that are more evenly distributed in difficulty. This makes sense from a qualitative linguistic analysis: SuperBPE tokens often correspond to multi-word expressions in English, i.e., word sequences that function as a single semantic unit (see Table 3 for examples). For instance, many prepositional phrases (e.g., *by accident* or *in the long run*) are essentially fixed and require memorization. The individual words in these expressions have very little possible variation in context, leading to very low-loss predictions under BPE models.

SuperBPE is a straightforward and local modification to tokenization, requiring no changes to the model architecture, training framework, or decoding strategy. Under the same training setup, SuperBPE provides a remarkable boost in both encoding efficiency and performance, yielding better language models overall.

## 2 SuperBPE

We first explain the standard byte-pair encoding (BPE; Sennrich et al., 2016) tokenization algorithm (§2.1), then introduce SuperBPE, which extends BPE to superword tokens (§2.2).

### 2.1 Background on BPE

BPE is a tokenization algorithm that greedily learns a subword vocabulary given training data.[2] The algorithm takes a sample of text and a target vocabulary size $T$ as input. Note that although the creation of a tokenizer is referred to as "learning," there are no parameters involved in the case of BPE, and the algorithm is completely deterministic given the data.

The first step of BPE is *pretokenization*, which splits the text into chunks that limit the extent of tokenization: merges cannot bridge these chunks, so the final learned tokens will be *parts* of these chunks. Canonically, pretokenization in BPE consists of splitting on whitespace, so that common word sequences do not become a single token. This made sense in the historical context of Sennrich et al. (2016), which aimed to improve word-level tokenization by segmenting words into morphologically meaningful subwords.

After pretokenization, the iterative learning algorithm begins. The training text is first split into bytes; the starting vocabulary is the set of all bytes. Then, the frequencies of all pairs of neighboring tokens are recorded, and the most frequent pair is merged into a single, new token at every position in the text where it occurs. The newly merged token is added to the vocabulary. For instance, if the merge is (`t`, `he`), then all instances of the token sequence [`t`, `he`] will be replaced with `the`, which is added to the vocabulary. The token pair frequencies are then updated, and the next most frequent pair is again merged into a new token. This continues until vocabulary reaches the target size $T$.

### 2.2 SuperBPE tokenization

SuperBPE introduces a simple intervention on the pretokenization step, separating tokenizer training into two discrete phases so that the tokenizer first learns subwords (by using pretokenization to prevent merges across whitespace) and then superwords (by lifting this restriction). Stage 1 is equivalent to regular BPE training and continues up to a certain vocabulary size $t$, which we call the *transition point* ($t < T$). In stage 2, tokenizer training resumes from the vocabulary learned so far, but this time whitespace pretokenization is skipped. As a result, token pairs bridging whitespace are considered, making it possible for superwords to be added to the vocabulary. Intuitively, we wish for our tokenizer to first learn base units of semantic meaning, then combine these units into common sequences for a much more efficient vocabulary. Note that $t = T$ corresponds to BPE, and $t = 0$ corresponds to a naive revision of BPE that foregoes whitespace pretokenization at any point in training.

[2]The algorithm originated in 1994 in the field of data compression (Gage, 1994).

| Category | Task | BPE | SuperBPE | Δ |
|---|---|---|---|---|
| Knowledge | ARC-Easy (MC) | 46.6 | **67.1** | +20.5** |
| | ARC-Challenge (MC) | 35.1 | **50.6** | +15.5** |
| | Jeopardy (EM) | **42.1** | 41.8 | −0.3 |
| | MMLU (MC) | 36.5 | **44.7** | +8.2** |
| | OpenbookQA (MC) | 33.2 | **54.4** | +21.2** |
| | TriviaQA (EM) | 60.6 | **61.3** | +0.7 |
| | WikidataQA (EM) | 69.7 | **70.9** | +1.2* |
| Math & Reasoning | Arithmetic (EM) | 54.8 | **59.3** | +4.5** |
| | GSM8K (EM) | 6.4 | **6.7** | +0.3 |
| | LSAT-AR (MC) | 21.3 | **23.0** | +1.7 |
| | Operators (EM) | **35.5** | 33.6 | −1.9 |
| | Repeat-Copy-Logic (EM) | 3.1 | **6.2** | +3.1 |
| Coding | HumanEval (pass@10) | **15.9** | 13.4 | −2.5 |
| | MBPP (pass@10) | 27.5 | **28.3** | +0.8 |
| Reading Comprehension | BoolQ (MC) | 59.7 | **64.6** | +4.9** |
| | CoQA (EM) | 12.6 | **13.2** | +0.6 |
| | DROP (EM) | 31.3 | **31.4** | +0.1 |
| | HotpotQA (EM) | 53.5 | **55.2** | +1.7* |
| | SQuAD (EM) | 75.1 | **75.8** | +0.7 |
| Commonsense | CommonsenseQA (MC) | 33.5 | **53.8** | +20.3** |
| | COPA (MC) | 77.0 | **85.8** | +8.8** |
| | PIQA (MC) | 55.2 | **59.8** | +4.6* |
| | Winograd (MC) | 50.4 | **53.1** | +2.7 |
| | Winogrande (MC) | 47.3 | **52.6** | +5.3* |
| Language Understanding | HellaSwag (MC) | 29.7 | **33.7** | +4.0** |
| | LAMBADA (EM) | **77.0** | 70.6 | −6.4** |
| | Language Identification (EM) | 8.8 | **9.0** | +0.2 |
| String Manipulation | CS Algorithms (EM) | 46.1 | **48.6** | +2.5 |
| | CUTE (EM) | 31.3 | **32.6** | +1.3 |
| | Dyck-Languages (EM) | **15.9** | 14.2 | −1.7 |
| Average | | 39.8 | **43.8** | +4.0 |

Table 1: **Performance of BPE and SuperBPE models (with transition point $t = 180$k) on 30 downstream tasks.** The two models are fixed in model parameters (8B), vocabulary size (200k), and training FLOPs (corresponding to ∼330B tokens), differing only in the algorithm for learning the vocabulary. The SuperBPE model outperforms the baseline on 25 out of 30 tasks, while also requiring 27% less compute at inference time. See Figure 3 for the moving task average during pretraining and §A.3 for further evaluation details. $^*p < 0.05$, $^{**}p < 0.005$ under a McNemar test.

We note that training tokenizers requires more system memory and CPU time when done without whitespace pretokenization (as in stage 2 of SuperBPE). This is because the training data is typically represented by a dictionary of "words" along with their counts. *With* whitespace pretokenization, the "words" are whitespace-separated chunks (e.g., common words) stored once alongside a large count, granting substantial savings in memory. *Without* whitespace pretokenization, the "words" are extremely long (e.g., entire training documents), leading to minimal deduplication of the text and excessively large dictionaries. Fortunately, tokenizer training must be done only once; in our experiments, SuperBPE tokenizers train in a few hours on 100 CPUs, a negligible cost compared to LLM pretraining.

### 2.3 Encoding efficiency

A tokenizer's encoding efficiency can be measured in *bytes-per-token*, i.e., how many UTF-8 bytes are encoded, on average, in each token over a large corpus of text (see calculation in Figure 1). We train a series of tokenizers on a 10 GB subset of data from OLMO 2's pretraining corpus, and evaluate encoding efficiency on a held-out subset.

Shown in Figure 1, SuperBPE scales much better with vocabulary size than BPE. BPE quickly plateaus around a vocabulary size of ∼50K, achieving 4.45 bytes-per-token at a vocabulary size of 200k. In fact, even with infinite vocabulary size (namely, if *every* whitespace-delimited word were in the vocabulary), BPE cannot exceed 4.68 bytes-per-token, i.e., the average word length in the held-out subset. SuperBPE exceeds this upper bound with a mere ∼12k vocabulary size, and reaches 5.55 bytes-per-token at 50K and 6.63 at 200k.

Perhaps surprisingly, SuperBPE is also more efficient than BPE with whitespace pretokenization completely disabled. Since BPE is a greedy algorithm, completely disabling whitespace pretokenization may cause BPE to make highly suboptimal choices early on. In particular, tokens in this setting often consist of the end of the previous word and start of the next word, as opposed to sequences of complete words. By keeping whitespace pretokenization on at the beginning, we can avoid suboptimal choices while still obtaining a tokenizer with superwords.

Figure 2 shows how SuperBPE's encoding efficiency depends on the choice of transition point $t$. The relationship is smooth, with $t = 80$k achieving the best encoding efficiency. However, we will see in our experiments that the optimal tokenizer for LM pretraining is not necessarily the most encoding-efficient.

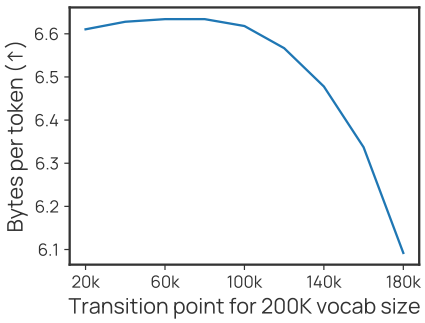



Figure 2: Encoding efficiency varies smoothly with the choice of transition point $t$ in SuperBPE's pretokenization curriculum.

## 3 Experiments

In our main experiments, we pretrain models from scratch while fixing the total training FLOPs and vocabulary size, changing only the algorithm for learning the vocabulary.

### 3.1 Setup

We first pretrain 8B models with BPE and SuperBPE tokenizers. We use the OLMO2 7B (OLMo et al., 2024) training configuration,[3] including the model architecture, training hyperparameters, and pretraining corpus, but reduce the total number of training steps to correspond to ∼330B tokens (compared to 4T) for the sake of compute. Following prior work (Pagnoni et al., 2024), we also fix the *effective* context size (measured in bytes) for each model. This prevents SuperBPE models from gaining an advantage by seeing more textual context for the same next-token prediction (Xiong et al., 2024). As more efficient models will have a shorter context length in tokens, the training steps are adjusted accordingly to match the total train FLOPs at the end of training.[4] Note that in this setting, a same-sized SuperBPE model uses fewer inference FLOPs compared to the BPE model.

We fix the vocabulary size of all tokenizers to 200,000 (in the same ballpark as e.g., GEMMA at 250k [Google, 2024], GPT-4O at 200k, LLAMA3 at 130k [Meta, 2024]).[5] We consider three transition points for SuperBPE: $t = 80$k, which has the best encoding efficiency, and two later transitions, $t = 160$k and $t = 180$k. All tokenizers are trained on the same 10 GB subset of OLMO2's pretraining mix. We provide further details about tokenizer training in §A.1.

We also train a slightly larger 11B SuperBPE model with $t = 180$k which approximately matches the 8B BPE baseline in total bytes of training data seen as well as both train *and* inference compute. See Table 2 for exact specifications for all runs.

[3] OLMO2 7B has 7.30B parameters while our 8B BPE and SuperBPE models have 8.12B parameters due to their increased vocabulary size.

[4] In practice, models using our more efficient tokenizers could shift some or all of the "saved" context FLOPs to longer effective contexts instead of more training steps.

[5] For 8B models, 200k vocabulary size is close to the recommendation of Tao et al. (2024) based on primarily-English data. We fix the vocabulary size to simplify comparisons between models.

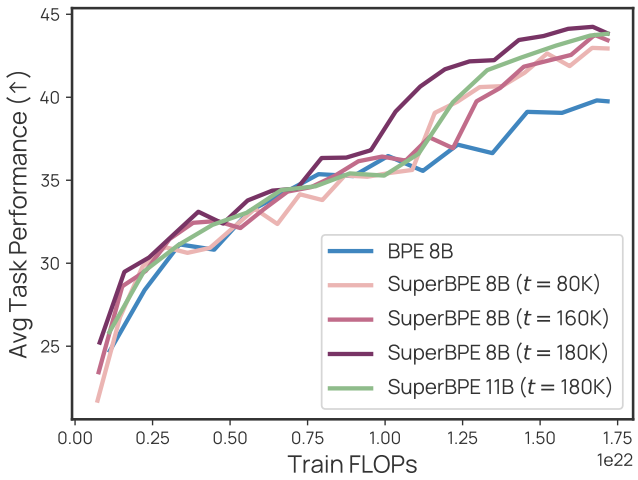



Figure 3: **Average task performance on 30 downstream tasks, evaluated at every 5000 steps in model pretraining**. We see that SuperBPE models consistently outperform the baseline that uses a BPE tokenizer. All compared models share the same vocabulary size and train budget; $t$ denotes the transition point in SuperBPE's pretokenization curriculum.

### 3.2 Results on downstream tasks

We evaluate on 30 benchmarks covering knowledge, math & reasoning, coding, reading comprehension, commonsense, language understanding, and string manipulation. The full evaluation suite is shown in Table 1; evaluation details can be found in §A.3.

Figure 3 shows the task average during pretraining. All SuperBPE models substantially outperform the BPE baseline at the end of training. The strongest 8B SuperBPE model, which has transition point $t = 180$k (the latest one we consider), outperforms the baseline by 4.0% on average, and wins on 25 out of 30 individual tasks. Table 1 shows the per-task performance for this model (see §A.3 for results for the other models). The largest gains appear to be on multiple choice tasks; when considering these alone, the performance improvement is +9.7%. The only task on which SuperBPE loses in a statistically significant way is LAMBADA; here, we observe that SuperBPE is actually ahead for the majority of training checkpoints, but accuracy dips at the end from 75.8% to 70.6% (see Figure 11).

Note that while the choice of transition point affects the performance of the resulting model, all reasonable choices are significantly stronger than the baseline. When using the most encoding-efficient transition point of $t = 80$k, we see a +3.1% task improvement over BPE while reducing inference compute by 35%. Later transition points empirically give up some gains in encoding efficiency in exchange for further improved performance.[6]

## 4 Analysis

### 4.1 Language modeling

Following prior work (Biderman et al., 2023; Xue et al., 2022; Yu et al., 2023; Wang et al., 2024), we evaluate language modeling performance using *bits-per-byte* (BPB), which normalizes the loss by the encoding efficiency of the tokenizer to fairly compare models with different tokenizers. This is necessary because longer tokens, on average, contain more information and therefore are more difficult to predict. Bits-per-byte is defined as $BPB(x) = \mathcal{L}_{\text{CE}}(x)/(\ln(2) \cdot n_{\text{bytes}})$ where $n_{\text{bytes}}$ is the length of the text in bytes and $\mathcal{L}_{\text{CE}}(x)$

[6]This finding adds to the ongoing debate about the relationship between tokenization compression and LM performance (Gallé, 2019; Goldman et al., 2024; Schmidt et al., 2024), providing further evidence that a higher compression does not necessarily lead to better performance.

| | BPE 8B | SuperBPE 8B | | | SuperBPE 11B |
|---|---|---|---|---|---|
| SuperBPE transition point | | $t = 80$k | $t = 160$k | $t = 180$k | $t = 180$k |
| Parameter count (billion) | 8.12 | 8.12 | 8.12 | 8.12 | 11.30 |
| Train steps | 76,543 | 118,419 | 112,722 | 107,982 | 77,525 |
| Average context length (bytes) | 18,262 | 18,272 | 18,263 | 18,268 | 18,268 |
| Vocabulary size | 200k | 200k | 200k | 200k | 200k |
| Context length (tokens) | 4,096 | 2,756 | 2,884 | 3,000 | 3,000 |
| Encoding efficiency (bytes/token) | 4.46 | 6.63 | 6.33 | 6.09 | 6.09 |
| Train compute ($10^{21}$ FLOPs) | 17.2 | 17.2 | 17.2 | 17.2 | 17.2 |
| Inference compute ($10^{9}$ FLOPs/byte) | 3.75 | 2.42 | 2.54 | 2.65 | 3.75 |

Table 2: **Training setup for the models we compare.** We fix the vocabulary size and effective context size (measured in bytes) that each model sees, and adjust the total number of training steps accordingly so that each model has the same total train budget (in FLOPs). The 8B SuperBPE models match the 8B BPE model in train compute while using less inference compute; the 11B SuperBPE model matches the 8B baseline in both train *and* inference compute. Numbers fixed across model settings are highlighted in the same color.

is the sum of the cross-entropy loss over the entire text.[7] We find that BPE 8B, SuperBPE 8B ($t = 180$k), and SuperBPE 11B attain 0.7465, 0.7482, and 0.7445 BPB, respectively, at the end of training. Though the numbers are all close, the rankings of the models according to BPB and downstream task performance are not consistent.

### 4.2 Fine-grained analysis of bits-per-byte

Why does the SuperBPE 8B model achieve slightly higher normalized language modeling loss (§4.1) than the baseline BPE model, despite outperforming it on a wide variety of downstream tasks (§3.2)? To investigate this, we plot the distribution of per-token BPB[8] for both models on data sampled from the pretraining data mixture in Figure 4.

We see that, although the BPE and SuperBPE models have very similar BPB on average, that loss is distributed very differently over the training data. Compared to the baseline, the SuperBPE model makes fewer predictions with either very high or very low loss.

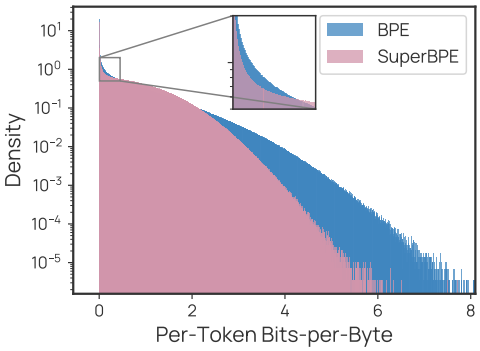



Figure 4: Histogram of per-token losses for both models from Table 1, measured over a large corpus of text. We see the SuperBPE model is a more consistent performer, making fewer predictions with very high or very low loss.

**Low-loss tokens.** We find that the reduction in low-loss tokens is attributable to a small set of extremely common words that the BPE model can easily predict, but are not available to SuperBPE as they are merged into larger superword tokens. For instance, the tokens `_the`, `_of`, and `_to` (the three most common words in the corpus) appear an order of magnitude more often under BPE than SuperBPE in the same corpus of text. *When excluding these three token types alone, the BPB ranking reverses*, with SuperBPE achieving 0.02 lower BPB than BPE.

The reduction in low-loss tokens also makes sense from a qualitative linguistic analysis of SuperBPE tokens. In Table 3, we show the most common POS tags among superword tokens in SuperBPE, along with random examples for each tag. The tokens often capture common multi-word expressions (*by accident*, *of course*, *for a living*) that function as a single

[7] Bits-per-byte of different models are considered comparable because total cross-entropy loss is a universal quantity representing the number of additional bits required to reconstruct the text given the model. This quantity is normalized by the number of bytes for easier interpretation.

[8] The per-token BPB is the per-token loss (in bits) divided by the average encoding efficiency.

| POS tag | # | Example Tokens |
|---|---|---|
| NN, IN | 906 | _case_of, _hint_of, _availability_of, _emphasis_on, _distinction_between |
| VB, DT | 566 | _reached_a, _discovered_the, _identify_the, _becomes_a, _issued_a |
| DT, NN | 498 | _this_month, _no_idea, _the_earth, _the_maximum, _this_stuff |
| IN, NN | 406 | _on_top, _by_accident, _in_effect, _for_lunch, _in_front |
| IN, DT | 379 | _on_the, _without_a, _alongside_the, _for_each |
| IN, DT, NN | 333 | _for_a_living, _by_the_way, _into_the_future, _in_the_midst |
| NN, IN, DT | 270 | _position_of_the, _component_of_the, _review_of_the, _example_of_this |
| IN, DT, JJ | 145 | _like_any_other, _with_each_other, _for_a_short, _of_the_entire |
| VB, IN, DT | 121 | _worked_as_a, _based_on_the, _combined_with_the, _turned_into_a |
| IN, DT, NN, IN | 33 | _at_the_time_of, _in_the_presence_of, _in_the_middle_of, _in_a_way_that |
| ,, CC, PRP, VB | 20 | ,_and_it_was, ,_but_I_think, ,_but_I_have, ,_but_I_am |
| IN, DT, JJ, NN | 18 | _in_the_long_run, _on_the_other_hand, _for_the_first_time, _in_the_same_way |

Table 3: **The most common POS tags for tokens of 2, 3, and 4 words in SuperBPE**, along with random example tokens for each tag. NN = noun, IN = preposition, VB = verb, DT = determiner, CC = conjunction, JJ = adjective, PRP = pronoun.

semantic unit (Schneider et al., 2014). As an example, prepositions (IN) figure prominently in superword tokens (e.g., *depend on*, *distinction between*) and require lexeme-specific memorization. The individual words in these fixed expressions are often semantically vacuous and have little possible variation in context, so they are easy to predict once memorized.

**High-loss tokens.** On the other hand, the much thinner tail of very high-loss tokens shows that, *in the worst case, the SuperBPE model consistently puts more probability mass on the correct token*. On average, we expect models to suffer high loss on tokens that are difficult to predict. This may explain why SuperBPE can outperform BPE on downstream tasks while having higher average BPB: the tokens that are scored in task evaluations tend to be among the hardest to predict. This is consistent with prior findings that models generally continue to improve in downstream tasks even as their overall loss plateaus, due to improving on a narrow and difficult slice of the distribution (Liu et al., 2023).

### 4.3 Scaling

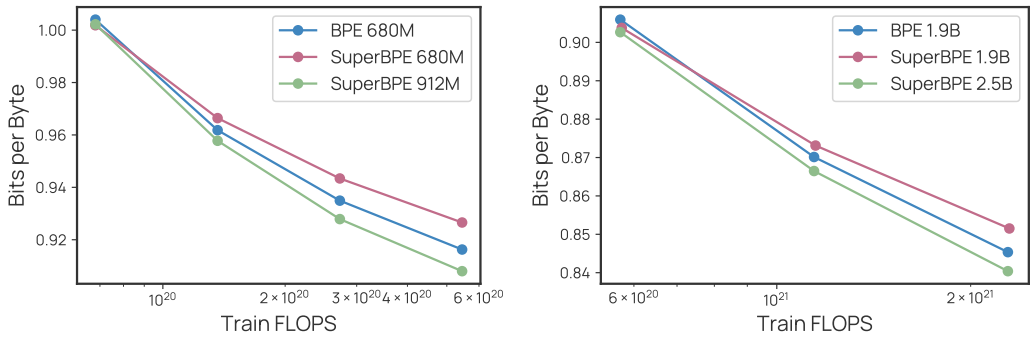
To characterize the scaling behavior of SuperBPE, we also perform experiments at smaller scales.[9] We train baseline models at 680M and 1.9B, and scale the base number of training tokens proportionately to the number of parameters. We also perform runs at $0.5\times$, $2\times$, and $4\times$ the base number of tokens to observe the trend with respect to training duration. Then, we train two SuperBPE models that match the training budget of each baseline BPE model: one that matches the baseline in parameter count (analogous to SuperBPE 8B), and a larger model that matches in both train and inference compute (analogous to SuperBPE 11B). We focus on the $t = 180$k tokenizer to reduce complexity.

We plot BPB at the end of training for each run in Figure 5. In the undertrained regime, both SuperBPE models achieve lower BPB than the baseline. In the overtrained regime, the ranking from worst to best is SuperBPE (matching parameter count), BPE, and SuperBPE (matching inference compute). Additionally, the separation between the models increases with further overtraining. We provide more results and comments on scaling in §B.3.

## 5 Related Work

**Tokenization beyond subwords** Prior work has explored processing text at multiple levels of granularity (Lai et al., 2021; Zhang et al., 2021) or creating multi-word tokens through frequency-based identification of $n$-grams (Gee et al., 2023; Kumar & Thawani, 2022). However, these were explored in limited experimental contexts (mainly for machine

[9]For scaling, we focus on BPB, since our downstream evals are too noisy for our small models to make meaningful comparisons.




(a) 680M model size

(b) 1.9B model size

Figure 5: **Scaling results for 680M and 1.9B baseline model sizes.** Compared to the **BPE** baseline, **SuperBPE with matching parameter count** achieves lower BPB in the undertrained regime, while **SuperBPE with matching inference compute** achieves lower BPB than the baseline at every model size and every training budget tested. Note that *BPB comparisons between BPE and SuperBPE models do not track downstream task accuracy*, due to differences in how BPE and SuperBPE models distribute loss over tokens (§4.2).

translation) and had mixed effectiveness. Naively disabling pretokenization in BPE has been found to severely hurt model performance (Dagan et al., 2024; Schmidt et al., 2024; Kudo, 2018), though it may be more promising for unigram tokenization (Kudo & Richardson, 2018), as adopted by JURASSIC (Lieber et al., 2021) and BLOOMBERGGPT (Wu et al., 2023). In concurrent work, Huang et al. (2025) disentangle the input and output vocabulary, expanding only the former to include $n$-gram tokens. Their method requires significant modifications of the LM input component and considers fixed length of $n$-gram tokens.

**Multi-token prediction** Multi-token prediction (MTP) equips LMs with some extra parameters to predict multiple tokens in a single time step (Qi et al., 2020; Gloeckle et al., 2024), and was recently adopted by DEEPSEEK-V3, though the MTP module is discarded at inference-time. The effectiveness corroborates that LMs are capable of predicting more than one subword in a forward pass. However, these approaches fix the number of tokens predicted in each time step and require modifications to the architecture and training objective. Nonetheless, the benefits of MTP and superword tokens may be orthogonal.

**Tokenizer-free language modeling** Some works have explored the possibility of removing tokenization entirely from LMs and directly modeling text as a sequence of bytes (Clark et al., 2022; Xue et al., 2022; Wang et al., 2024). To overcome the increased compute requirement due to expanded sequence lengths, alternative architectures have been proposed that either segment bytes into fixed-length patches (Tay et al., 2022; Yu et al., 2023) or dynamically predict patch boundaries with sub-modules (Nawrot et al., 2023; Pagnoni et al., 2024; Ahia et al., 2024), increasing the complexity of the model.

## 6 Conclusion

Although tokenization lies at the foundation of language modeling—acting as the lens through which models view text—the algorithms in use have remained largely unchanged over the past decade. SuperBPE builds on the natural observation that tokens need not be limited to subwords, extending the BPE algorithm to superword tokens. When replacing subword BPE tokenizers with SuperBPE tokenizers in pretraining, we find that language models achieve better performance over a large suite of downstream tasks, while also being substantially more efficient at inference time. These benefits are achieved without modifying the underlying model architecture, making SuperBPE a compelling alternative to BPE that seamlessly integrates with modern language model ecosystems.

## Acknowledgments

We would like to thank Alex Fang for pretraining advice, Vivek Ramanujan for helping debug our distributed training setup, Ian Magnusson for helpful comments on LM evaluation, and Zhaofeng Wu, Alexander Fang, and Xiaochuang Han for feedback on drafts. We would also like to thank Luca Soldaini, Gonçalo Faria, Shrimai Prabhumoye, Matt Jordan, Artidoro Pagnoni, Mike Lewis, Doug Downey, Shannon Shen, and the UW NLP community for valuable conversations about this work. Both co-first authors, AL and JH, are supported by the NSF Graduate Research Fellowship Program. JH and SO are supported in part by the Microsoft Grant for Customer Experience Innovation. This work was further supported in part by NSF DMS-2134012, NSF CCF-2019844, ONR N00014-24-1-2207, NSF 2113530, and NVIDIA resources provided through the National AI Research Resource Pilot (NAIRR).

## References

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

Orevaoghene Ahia, Sachin Kumar, Hila Gonen, Valentin Hofmann, Tomasz Limisiewicz, Yulia Tsvetkov, and Noah A. Smith. MAGNET: Improving the multilingual fairness of language models with adaptive gradient-based tokenization. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=1e3MOwHSIX.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models, 2021. URL https://arxiv.org/abs/2108.07732.

Stella Biderman, Hailey Schoelkopf, Quentin Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, Aviya Skowron, Lintang Sutawika, and Oskar van der Wal. Pythia: A suite for analyzing large language models across training and scaling, 2023. URL https://arxiv.org/abs/2304.01373.

BIG-bench. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=uyTL5Bvosj.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. Piqa: Reasoning about physical commonsense in natural language. In *Thirty-Fourth AAAI Conference on Artificial Intelligence*, 2020.

Thorsten Brants, Ashok C. Popat, Peng Xu, Franz J. Och, and Jeffrey Dean. Large language models in machine translation. In Jason Eisner (ed.), *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pp. 858–867, Prague, Czech Republic, June 2007. Association for Computational Linguistics. URL https://aclanthology.org/D07-1090/.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. 2020.

Anthony Chen, Pallavi Gudipati, Shayne Longpre, Xiao Ling, and Sameer Singh. Evaluating entity disambiguation and the role of popularity in retrieval-based NLP. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli (eds.), *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 4472–4485, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.345. URL https://aclanthology.org/2021.acl-long.345.

Kenneth Church. How many multiword expressions do people know? In *Proceedings of the Workshop on Multiword Expressions: From Parsing and Generation to the Real World*, pp. 137–144, Portland, Oregon, USA, 2011. Association for Computational Linguistics. URL https://aclanthology.org/W11-0823/.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. BoolQ: Exploring the surprising difficulty of natural yes/no questions. In Jill Burstein, Christy Doran, and Thamar Solorio (eds.), *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 2924–2936, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1300. URL https://aclanthology.org/N19-1300.

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. doi: 10.1162/tacl_a_00448. URL https://aclanthology.org/2022.tacl-1.5.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge, 2018. URL https://arxiv.org/abs/1803.05457.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168.

Pablo Contreras Kallens and Morten H. Christiansen. Models of language and multiword expressions. *Frontiers in Artificial Intelligence*, 5, 2022. doi: 10.3389/frai.2022.781962. URL https://www.frontiersin.org/journals/artificial-intelligence/articles/10.3389/frai.2022.781962.

Gautier Dagan, Gabriel Synnaeve, and Baptiste Rozière. Getting the most out of your tokenizer for pre-training and domain adaptation. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Haowei Zhang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Li, Hui Qu, J. L. Cai, Jian Liang, Jianzhong Guo, Jiaqi Ni, Jiashi Li, Jiawei Wang, Jin Chen, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, Junxiao Song, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Xu, Leyi Xia, Liang Zhao, Litong Wang, Liyue Zhang, Meng Li, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Ning Tian, Panpan Huang, Peiyi Wang, Peng Zhang, Qiancheng Wang, Qihao Zhu, Qinyu Chen, Qiushi Du, R. J. Chen, R. L. Jin, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runxin Xu, Ruoyu Zhang, Ruyi Chen, S. S. Li, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaoqing Wu, Shengfeng Ye, Shengfeng Ye, Shirong Ma, Shiyu Wang, Shuang Zhou, Shuiping Yu, Shunfeng Zhou, Shuting Pan, T. Wang, Tao Yun, Tian Pei, Tianyu Sun, W. L. Xiao, Wangding Zeng, Wanjia Zhao, Wei An, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, X. Q. Li, Xiangyue Jin, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaojin Shen, Xiaokang Chen,

Xiaokang Zhang, Xiaosha Chen, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingkai Yu, Xinnan Song, Xinxia Shan, Xinyi Zhou, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, Y. K. Li, Y. Q. Wang, Y. X. Wei, Y. X. Zhu, Yang Zhang, Yanhong Xu, Yanhong Xu, Yanping Huang, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Li, Yaohui Wang, Yi Yu, Yi Zheng, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Ying Tang, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yu Wu, Yuan Ou, Yuchen Zhu, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yukun Zha, Yunfan Xiong, Yunxian Ma, Yuting Yan, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Z. F. Wu, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhen Huang, Zhen Zhang, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhipeng Xu, Zhiyu Wu, Zhongyu Zhang, Zhuoshu Li, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Ziyi Gao, and Zizheng Pan. Deepseek-v3 technical report, 2025. URL https://arxiv.org/abs/2412.19437.

Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. DROP: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In Jill Burstein, Christy Doran, and Thamar Solorio (eds.), *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 2368–2378, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1246. URL https://aclanthology.org/N19-1246.

Lukas Edman, Helmut Schmid, and Alexander Fraser. CUTE: Measuring LLMs' understanding of their tokens. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 3017–3026, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.177. URL https://aclanthology.org/2024.emnlp-main.177.

Philip Gage. A new algorithm for data compression. *The C Users Journal archive*, 12:23–38, 1994. URL https://api.semanticscholar.org/CorpusID:59804030.

Matthias Gallé. Investigating the effectiveness of BPE: The power of shorter sequences. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (eds.), *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 1375–1381, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1141. URL https://aclanthology.org/D19-1141.

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

Fabian Gloeckle, Badr Youbi Idrissi, Baptiste Roziere, David Lopez-Paz, and Gabriel Synnaeve. Better & faster large language models via multi-token prediction. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=pEWAcejiU2.

Omer Goldman, Avi Caciularu, Matan Eyal, Kris Cao, Idan Szpektor, and Reut Tsarfaty. Unpacking tokenization: Evaluating text compression and its correlation with model performance. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 2274–2286, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.134. URL https://aclanthology.org/2024.findings-acl.134.

Google. Gemma: Open models based on gemini research and technology, 2024.

Jonathan Hayase, Alisa Liu, Yejin Choi, Sewoong Oh, and Noah A. Smith. Data mixture inference: What do bpe tokenizers reveal about their training data?, 2024. URL https://arxiv.org/abs/2407.16607.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=d7KBjmI3GmQ.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, pp. 30016–30030, 2022.

Valentin Hofmann, Janet Pierrehumbert, and Hinrich Schütze. Superbizarre is not superb: Derivational morphology improves BERT's interpretation of complex words. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli (eds.), *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 3594–3608, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.279. URL https://aclanthology.org/2021.acl-long.279.

Hongzhi Huang, Defa Zhu, Banggu Wu, Yutao Zeng, Ya Wang, Qiyang Min, and Xun Zhou. Over-tokenized transformer: Vocabulary is generally worth scaling, 2025. URL https://arxiv.org/abs/2501.16975.

Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In Regina Barzilay and Min-Yen Kan (eds.), *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1601–1611, Vancouver, Canada, July 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1147. URL https://aclanthology.org/P17-1147.

Guy Kaplan, Matanel Oren, Yuval Reif, and Roy Schwartz. From tokens to words: On the inner lexicon of LLMs. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=328vch6tRs.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Pride Kavumba, Naoya Inoue, Benjamin Heinzerling, Keshav Singh, Paul Reisert, and Kentaro Inui. When choosing plausible alternatives, clever hans can be clever. In Simon Ostermann, Sheng Zhang, Michael Roth, and Peter Clark (eds.), *Proceedings of the First Workshop on Commonsense Inference in Natural Language Processing*, pp. 33–42, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-6004. URL https://aclanthology.org/D19-6004.

Taku Kudo. Sentencepiece experiments. https://github.com/google/sentencepiece/blob/master/doc/experiments.md, 2018.

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

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

Vedang Lad, Wes Gurnee, and Max Tegmark. The remarkable robustness of llms: Stages of inference?, 2024. URL https://arxiv.org/abs/2406.19384.

Yuxuan Lai, Yijia Liu, Yansong Feng, Songfang Huang, and Dongyan Zhao. Lattice-BERT: Leveraging multi-granularity representations in Chinese pre-trained language models. In Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tur, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou (eds.), *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 1716–1731, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.137. URL https://aclanthology.org/2021.naacl-main.137.

Sander Land. A short introduction to pre-tokenization weirdness, 2024. URL https://tokencontributions.substack.com/p/a-short-introduction-to-pre-tokenization.

Sander Land and Max Bartolo. Fishing for magikarp: Automatically detecting under-trained tokens in large language models. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 11631–11646, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.649. URL https://aclanthology.org/2024.emnlp-main.649.

Hector J. Levesque, Ernest Davis, and Leora Morgenstern. The winograd schema challenge. In *Proceedings of the Thirteenth International Conference on Principles of Knowledge Representation and Reasoning*, pp. 552–561. AAAI Press, 2012.

Jeffrey Li, Alex Fang, Georgios Smyrnis, Maor Ivgi, Matt Jordan, Samir Gadre, Hritik Bansal, Etash Guha, Sedrick Keh, Kushal Arora, Saurabh Garg, Rui Xin, Niklas Muennighoff, Reinhard Heckel, Jean Mercat, Mayee Chen, Suchin Gururangan, Mitchell Wortsman, Alon Albalak, Yonatan Bitton, Marianna Nezhurina, Amro Abbas, Cheng-Yu Hsieh, Dhruba Ghosh, Josh Gardner, Maciej Kilian, Hanlin Zhang, Rulin Shao, Sarah Pratt, Sunny Sanyal, Gabriel Ilharco, Giannis Daras, Kalyani Marathe, Aaron Gokaslan, Jieyu Zhang, Khyathi Chandu, Thao Nguyen, Igor Vasiljevic, Sham Kakade, Shuran Song, Sujay Sanghavi, Fartash Faghri, Sewoong Oh, Luke Zettlemoyer, Kyle Lo, Alaaeldin El-Nouby, Hadi Pouransari, Alexander Toshev, Stephanie Wang, Dirk Groeneveld, Luca Soldaini, Pang Wei Koh, Jenia Jitsev, Thomas Kollar, Alexandros G. Dimakis, Yair Carmon, Achal Dave, Ludwig Schmidt, and Vaishaal Shankar. Datacomp-lm: In search of the next generation of training sets for language models, 2024. URL https://arxiv.org/abs/2406.11794.

Opher Lieber, Or Sharir, Barak Lenz, and Yoav Shoham. Jurassic-1: Technical details and evaluation, 2021. URL https://uploads-ssl.webflow.com/60fd4503684b466578c0d307/61138924626a6981ee09caf6_jurassic_tech_paper.pdf.

Hong Liu, Sang Michael Xie, Zhiyuan Li, and Tengyu Ma. Same pre-training loss, better downstream: Implicit bias matters for language models. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 22188–22214. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/liu23ao.html.

Jiacheng Liu, Sewon Min, Luke Zettlemoyer, Yejin Choi, and Hannaneh Hajishirzi. Infini-gram: Scaling unbounded n-gram language models to a trillion tokens. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=u2vAyMeLMm.

Scott Lundberg. The art of prompt design: Prompt boundaries and token healing, 2023. URL https://medium.com/towards-data-science/the-art-of-prompt-design-prompt-boundaries-and-token-healing-3b2448b0be38.

Haspelmath Martin. The indeterminacy of word segmentation and the nature of morphology and syntax. *Folia Linguistica*, 51(s1000):31–80, 2017. doi: doi:10.1515/flin-2017-1005. URL https://doi.org/10.1515/flin-2017-1005.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. Locating and editing factual associations in gpt. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh

(eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 17359–17372. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/6f1d43d5a82a37e89b0665b33bf3a182-Paper-Conference.pdf.

Meta. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

Sabrina J. Mielke, Zaid Alyafeai, Elizabeth Salesky, Colin Raffel, Manan Dey, Matthias Gallé, Arun Raja, Chenglei Si, Wilson Y. Lee, Benoît Sagot, and Samson Tan. Between words and characters: A brief history of open-vocabulary modeling and tokenization in nlp, 2021. URL https://arxiv.org/abs/2112.10508.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii (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.

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

Rodrigo Nogueira, Zhiying Jiang, and Jimmy Lin. Investigating the limitations of transformers with simple arithmetic tasks, 2021. URL https://arxiv.org/abs/2102.13019.

Team OLMo, Pete Walsh, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Shane Arora, Akshita Bhagia, Yuling Gu, Shengyi Huang, Matt Jordan, Nathan Lambert, Dustin Schwenk, Oyvind Tafjord, Taira Anderson, David Atkinson, Faeze Brahman, Christopher Clark, Pradeep Dasigi, Nouha Dziri, Michal Guerquin, Hamish Ivison, Pang Wei Koh, Jiacheng Liu, Saumya Malik, William Merrill, Lester James V. Miranda, Jacob Morrison, Tyler Murray, Crystal Nam, Valentina Pyatkin, Aman Rangapur, Michael Schmitz, Sam Skjonsberg, David Wadden, Christopher Wilhelm, Michael Wilson, Luke Zettlemoyer, Ali Farhadi, Noah A. Smith, and Hannaneh Hajishirzi. 2 olmo 2 furious, 2024. URL https://arxiv.org/abs/2501.00656.

OpenAI. Hello GPT-4o, 2024. URL https://openai.com/index/hello-gpt-4o/.

Naoki Otani, Satoru Ozaki, Xingyuan Zhao, Yucen Li, Micaelah St Johns, and Lori Levin. Pre-tokenization of multi-word expressions in cross-lingual word embeddings. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (eds.), *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 4451–4464, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.360. URL https://aclanthology.org/2020.emnlp-main.360.

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

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The LAMBADA dataset: Word prediction requiring a broad discourse context. In Katrin Erk and Noah A. Smith (eds.), *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1525–1534, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1144. URL https://aclanthology.org/P16-1144.

Jackson Petty, Sjoerd van Steenkiste, Fei Sha, Ishita Dasgupta, Dan Garrette, and Tal Linzen. The impact of depth and width on transformer language model generalization. 2023.

Buu Phan, Marton Havasi, Matthew Muckley, and Karen Ullrich. Understanding and mitigating tokenization bias in language models, 2024. URL https://arxiv.org/abs/2406.16829.

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

Weizhen Qi, Yu Yan, Yeyun Gong, Dayiheng Liu, Nan Duan, Jiusheng Chen, Ruofei Zhang, and Ming Zhou. ProphetNet: Predicting future n-gram for sequence-to-SequencePre-training. In Trevor Cohn, Yulan He, and Yang Liu (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp. 2401–2410, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.findings-emnlp.217. URL https://aclanthology.org/2020.findings-emnlp.217.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019. URL https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ questions for machine comprehension of text. In Jian Su, Kevin Duh, and Xavier Carreras (eds.), *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 2383–2392, Austin, Texas, November 2016. Association for Computational Linguistics. doi: 10.18653/v1/D16-1264. URL https://aclanthology.org/D16-1264.

Siva Reddy, Danqi Chen, and Christopher D. Manning. CoQA: A conversational question answering challenge. *Transactions of the Association for Computational Linguistics*, 7:249–266, 2019. doi: 10.1162/tacl_a_00266. URL https://aclanthology.org/Q19-1016.

Marco Tulio Ribeiro. A guidance language for controlling large language models, 2023. URL https://github.com/guidance-ai/guidance?tab=readme-ov-file#text-not-tokens.

Melissa Roemmele, Cosmin Adrian Bejan, and Andrew S. Gordon. Choice of plausible alternatives: An evaluation of commonsense causal reasoning. In *Proceedings of the Association for the Advancement of Artificial Intelligence (AAAI) Spring Symposium*, 2011.

Jessica Rumbelow and Matthew Watkins. Solidgoldmagikarp (plus, prompt generation), 2023. URL https://www.lesswrong.com/posts/aPeJE8bSo6rAFoLqg/solidgoldmagikarp-plus-prompt-generation.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: an adversarial winograd schema challenge at scale. *Commun. ACM*, 64(9):99–106, August 2021. ISSN 0001-0782. URL https://doi.org/10.1145/3474381.

Bahar Salehi, Paul Cook, and Timothy Baldwin. A word embedding approach to predicting the compositionality of multiword expressions. In Rada Mihalcea, Joyce Chai, and Anoop Sarkar (eds.), *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 977–983, Denver, Colorado, 2015. Association for Computational Linguistics. doi: 10.3115/v1/N15-1099. URL https://aclanthology.org/N15-1099/.

Craig W Schmidt, Varshini Reddy, Haoran Zhang, Alec Alameddine, Omri Uzan, Yuval Pinter, and Chris Tanner. Tokenization is more than compression. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 678–702, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.40. URL https://aclanthology.org/2024.emnlp-main.40.

Nathan Schneider, Spencer Onuffer, Nora Kazour, Emily Danchik, Michael T. Mordowanec, Henrietta Conrad, and Noah A. Smith. Comprehensive annotation of multiword expressions in a social web corpus. In Nicoletta Calzolari, Khalid Choukri, Thierry Declerck,

Hrafn Loftsson, Bente Maegaard, Joseph Mariani, Asuncion Moreno, Jan Odijk, and Stelios Piperidis (eds.), *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, pp. 455–461, Reykjavik, Iceland, May 2014. European Language Resources Association (ELRA). URL https://aclanthology.org/L14-1433/.

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

Anya Sims, Cong Lu, Klara Kaleb, Jakob Nicolaus Foerster, and Yee Whye Teh. Stochastok: Improving fine-grained subword understanding in LLMs. In *ICLR 2025 Workshop on Building Trust in Language Models and Applications*, 2025. URL https://openreview.net/forum?id=PZnDZdkGsE.

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

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. CommonsenseQA: A question answering challenge targeting commonsense knowledge. In Jill Burstein, Christy Doran, and Thamar Solorio (eds.), *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4149–4158, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1421. URL https://aclanthology.org/N19-1421.

Chaofan Tao, Qian Liu, Longxu Dou, Niklas Muennighoff, Zhongwei Wan, Ping Luo, Min Lin, and Ngai Wong. Scaling laws with vocabulary: Larger models deserve larger vocabularies. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

Yi Tay, Mostafa Dehghani, Jinfeng Rao, William Fedus, Samira Abnar, Hyung Won Chung, Sharan Narang, Dani Yogatama, Ashish Vaswani, and Donald Metzler. Scale efficiently: Insights from pre-training and fine-tuning transformers. *arXiv preprint arXiv:2109.10686*, 2021.

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. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=JtBRnrlOEFN.

Avijit Thawani, Jay Pujara, Filip Ilievski, and Pedro Szekely. Representing numbers in NLP: a survey and a vision. In Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tur, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou (eds.), *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 644–656, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.53. URL https://aclanthology.org/2021.naacl-main.53.

Menan Velayuthan and Kengatharaiyer Sarveswaran. Egalitarian language representation in language models: It all begins with tokenizers. In Owen Rambow, Leo Wanner, Marianna Apidianaki, Hend Al-Khalifa, Barbara Di Eugenio, and Steven Schockaert (eds.), *Proceedings of the 31st International Conference on Computational Linguistics*, pp. 5987–5996, Abu Dhabi, UAE, January 2025. Association for Computational Linguistics. URL https://aclanthology.org/2025.coling-main.400/.

Bandhav Veluri, Justin Chan, Malek Itani, Tuochao Chen, Takuya Yoshioka, and Shyamnath Gollakota. Real-time target sound extraction. In *ICASSP*, pp. 1–5, 2023. URL https://doi.org/10.1109/ICASSP49357.2023.10094573.

Tim Vieira, Ben LeBrun, Mario Giulianelli, Juan Luis Gastaldi, Brian DuSell, John Terilla, Timothy J O'Donnell, and Ryan Cotterell. From language models over tokens to language models over characters. *arXiv preprint arXiv:2412.03719*, 2024.

Junxiong Wang, Tushaar Gangavarapu, Jing Nathan Yan, and Alexander M Rush. Mambabyte: Token-free selective state space model. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=X1xNsuKssb.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. Transformers: State-of-the-art natural language processing. In Qun Liu and David Schlangen (eds.), *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 38–45, Online, October 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-demos.6. URL https://aclanthology.org/2020.emnlp-demos.6.

Shijie Wu, Ozan Irsoy, Steven Lu, Vadim Dabravolski, Mark Dredze, Sebastian Gehrmann, Prabhanjan Kambadur, David Rosenberg, and Gideon Mann. Bloomberggpt: A large language model for finance, 2023. URL https://arxiv.org/abs/2303.17564.

Yizhe Xiong, Xiansheng Chen, Xin Ye, Hui Chen, Zijia Lin, Haoran Lian, Zhenpeng Su, Jianwei Niu, and Guiguang Ding. Temporal scaling law for large language models, 2024. URL https://arxiv.org/abs/2404.17785.

Linting Xue, Aditya Barua, Noah Constant, Rami Al-Rfou, Sharan Narang, Mihir Kale, Adam Roberts, and Colin Raffel. ByT5: Towards a token-free future with pre-trained byte-to-byte models. *Transactions of the Association for Computational Linguistics*, 10:291–306, 2022. doi: 10.1162/tacl_a_00461. URL https://aclanthology.org/2022.tacl-1.17.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii (eds.), *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2369–2380, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1259. URL https://aclanthology.org/D18-1259.

Lili Yu, Daniel Simig, Colin Flaherty, Armen Aghajanyan, Luke Zettlemoyer, and Mike Lewis. MEGABYTE: Predicting million-byte sequences with multiscale transformers. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=JTmO2V9Xpz.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a machine really finish your sentence? In Anna Korhonen, David Traum, and Lluís Màrquez (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.

Xinsong Zhang, Pengshuai Li, and Hang Li. AMBERT: A pre-trained language model with multi-grained tokenization. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli (eds.), *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pp. 421–435, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.findings-acl.37. URL https://aclanthology.org/2021.findings-acl.37.

Wanjun Zhong, Siyuan Wang, Duyu Tang, Zenan Xu, Daya Guo, Yining Chen, Jiahai Wang, Jian Yin, Ming Zhou, and Nan Duan. Analytical reasoning of text. In Marine

Carpuat, Marie-Catherine de Marneffe, and Ivan Vladimir Meza Ruiz (eds.), *Findings of the Association for Computational Linguistics: NAACL 2022*, pp. 2306–2319, Seattle, United States, July 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.findings-naacl.177. URL https://aclanthology.org/2022.findings-naacl.177.

Wanjun Zhong, Ruixiang Cui, Yiduo Guo, Yaobo Liang, Shuai Lu, Yanlin Wang, Amin Saied, Weizhu Chen, and Nan Duan. AGIEval: A human-centric benchmark for evaluating foundation models. In Kevin Duh, Helena Gomez, and Steven Bethard (eds.), *Findings of the Association for Computational Linguistics: NAACL 2024*, pp. 2299–2314, Mexico City, Mexico, June 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-naacl.149. URL https://aclanthology.org/2024.findings-naacl.149.

# A Experimental setup details

## A.1 Tokenizer training

We use the HuggingFace `tokenizers` (Wolf et al., 2020) library for tokenizer training.

### *A.1.1 Tokenizer training data*

We produce the tokenizer training data by sampling documents uniformly at random from the OLMO2 stage 2 pretraining data, referred to as `olmo-mix`. We use a 10 GB subset because early experiments showed that data beyond even ∼10 MB does not make a difference in the resulting tokenizer's encoding efficiency.

We found that `olmo-mix` had several extremely long documents, with the longest 1% of documents making up 15% of the data. In particular, a full academic paper (specifically Veluri et al., 2023) is duplicated 2,224 times back-to-back inside one document (as delimited by special EOS tokens). Because our tokenizers are trained on small sets of data, these extremely long documents can take up a large proportion of the data, resulting in unusual tokens like `_chunk-based_processing`. To circumvent possible data duplication issues, we truncate the longest 1% of documents in the tokenizer training data to the 99% percentile of document lengths. As future practitioners train SuperBPE tokenizers, we encourage especial attention to deduplication, which may have an outsized impact on SuperBPE tokenizers.

### *A.1.2 Limit on the size of superword tokens*

Even after truncating the longest 1% of documents, we found that SuperBPE tokenizers can still have extremely long tokens consisting of highly duplicated boilerplate text such as the Project Gutenberg license or common internet phrases such as `You_are_commenting_using_your`. This issue is already present in BPE tokenizers trained on Chinese, which contain sentence-long tokens clearly taken from pornographic content. For instance, tokens in GPT-4O's tokenizer include 最新高清无码 = *latest HD uncensored* and 娱乐网址 = *entertainment website*. To prevent concerns about the tokenizer directly revealing parts of the training data (Hayase et al., 2024), we enforce an upper bound of 4 words in our tokens. Empirically, we found that this had no measurable impact on the encoding efficiency of the tokenizers or the resulting trained LMs.

### *A.1.3 Pretokenization rules*

We implement whitespace pretokenization with the default `regex` string from `tokenizers` which was adopted by the GPT-2 tokenizer.

```
?\p{L}+| ?[^\s\p{L}\p{N}]+|\s+(?!\S)|\s+
```

Note that the original GPT-2 pretokenization regex string also splits on contractions, e.g., splitting `I'm` into `[I, 'm]`. Since this choice is not universal among commercial tokenizers and is not related to whitespace pretokenization (and furthermore creates plenty of undesirable edge cases [Land, 2024]), we do not include this rule.

Independently of whitespace pretokenization (i.e., for both BPE and SuperBPE tokenizers), we follow recent convention (as introduced by GPT-3.5 and borrowed by LLAMA3, OLMO2) and pretokenize digits into blocks of 3. We make one modification, by grouping digits into 3 from the right rather than from the left, so that, e.g., `1000` would be pretokenized as [`1`, `000`] instead of [`100`, `0`]. This choice was recently found to yield improved performance on math benchmarks, even when applied solely at inference time (Singh & Strouse, 2024). Digit pretokenization is enforced with the following regex.

```
(?=(\d{3})+(?!\d))
```

#### *A.1.4 Special casing of colon*

In order to make our tokenizer compatible with the common question-answering format where the prompt ends with a colon and the continuation is expected to start with a space, we "special-case" colon by preventing the algorithm from learning any tokens that contain ": " as a substring. Without this fix, common question/answer prompts might produce distorted distributions over completions. Please see §C.3 for further discussion. This affects the resulting tokenizer minimally in terms of the learned vocabulary.

### A.2 Scaling model configurations

When matching inference compute, the goal is to match the average flops per byte of generated text between two models with different tokenizers. To do so, we scale the model up to cancel the effect of longer tokens, which requires precise control over the model's size. To produce a model config with an arbitrary inference compute cost, we first represent the inference flops per token as a polynomial in terms of the model dimension, MLP hidden dimension, and number of layers. Conveniently, once the model dimension and number of layers are chosen, the flops are affine in the MLP hidden dimension, so we can easily solve for the MLP hidden dimension that gets us closest to the desired budget. We fix the head dimension to that of the base model.

To find the best config overall, we grid search over the hidden dimension (which must remain a multiple of the head dimension) and number of layers, solving for the MLP hidden dimension at each step. We choose the config which expands the transformer by the most uniform factors. This is measured by taking the ratios of the current parameters with the base config's parameters, applying the logarithm, and taking the standard deviation. While prior work has explored the best way to scale transformer models (Tay et al., 2021; Petty et al., 2023), we believe that scaling all parameters uniformly is reasonable since we are only increasing the model size by a small amount.

We present the exact model hyperparameters for all model sizes used in our experiments in Table 4.

| | **680M** | **910M** | **1.9B** | **2.5B** | **8B** | **11B** |
|---|---|---|---|---|---|---|
| Parameter count | 678.2M | 912.5M | 1.893B | 2.536B | 8.115B | 11.30B |
| Model dimension | 1024 | 1,216 | 2,048 | 2,304 | 4,096 | 4,608 |
| MLP hidden dimension | 8,192 | 9,728 | 16,384 | 18,432 | 22,016 | 24,704 |
| Head dimension | 64 | 64 | 128 | 128 | 128 | 128 |
| Number of heads | 16 | 19 | 16 | 18 | 32 | 36 |
| Number of layers | 16 | 18 | 16 | 19 | 32 | 37 |
| Vocabulary size | 20,0005 | 20,0005 | 20,0005 | 20,0005 | 20,0005 | 20,0005 |

Table 4: **Model parameters for all model sizes.** Model sizes 910M, 2.5B, and 11B are scaled versions of 680M, 1.9B, and 8B respectively. All other parameters match those of OLMO 300M (from the OLMO model ladder) for sizes 680M and 910M, OLMO 1B for sizes 1.9B and 2.5B, or OLMO2 7B for sizes 8B and 11B, respectively. Maximum sequence length values for various tokenizers are listed in Table 2.

### A.3 Evaluation Suite

Our evaluation suite builds on DataComp-LM's core evaluation of 22 tasks (Li et al., 2024), which was found to provide low-variance signal of learning. We add 8 more popular tasks (e.g., MMLU, GSM8K) while also covering string manipulation tasks (e.g., CUTE), which are known to be challenging for LMs due to their tokenizers.

All evaluations are based on decoding from the model and scoring the generation by either comparing it to the ground truth or evaluating its functional correctness (in the case of coding tasks). For multiple choice (MC) tasks, we check whether the predicted answer choice is an exact match (EM) to the target (we observe that effectively 100% of model generations are valid answer choices, especially for later checkpoints). For open-ended tasks, we check whether the generated output contains the ground truth answer exactly, and for coding tasks, we report pass@10.

We provide 5 in-context examples for all tasks, except for CoQA, which naturally contains in-context examples in the conversational context, and the coding tasks (HumanEval and MBPP), which are evaluated zero-shot following prior work. We use a maximum of 5,000 examples from each dataset, though some datasets contain much fewer examples. BB below stands for BIG-Bench.

**ARC** consists of 4-way MC questions from grades 3–9 science exams. It contains two splits, ARC-Easy, which require knowledge of basic science, and ARC-Challenge, which require some procedural reasoning (Clark et al., 2018).

**Arithmetic** contains simple arithmetic problems (Brown et al., 2020).[10] We use the `2da`, `2dm`, and `2ds` splits for addition, multiplication, and division of (up to) 2-digit numbers.

**BoolQ** contains naturally occurring yes/no questions paired with passages that provide an answer (Clark et al., 2019).

**CommonsenseQA** contains 5-way MC questions that require commonsense knowledge to answer (Talmor et al., 2019).

**COPA** contains two-way MC questions about cause and effect (Roemmele et al., 2011; Kavumba et al., 2019).

**CoQA** consists of passages with a series of conversational questions about the passage Reddy et al. (2019). Each question requires the prior conversational context, due to possible coreference across questions. Because these contextual questions naturally serve as in-context examples, we do not provide additional in-context examples.

**BB CS Algorithms** consists of two subtasks, determining whether a given series of parentheses is balanced and identifying the longest common subsequence in two letter strings (BIG-bench, 2023).

**CUTE** contains questions that require the model to understand and manipulate spelling, such as replacing all instances of a particular letter in a word with another letter (Edman et al., 2024).

**DROP** contains questions about passages, potentially requiring reasoning over multiple pieces of information in the passage (Dua et al., 2019).

**BB Dyck Languages** consists of a sequence of parentheses and requires the model to predict the correct sequence of closing parentheses so that the entire sequence is well-balanced.

[10] https://huggingface.co/datasets/EleutherAI/arithmetic

**GSM8K** contains grade school math word problems that require between 2 and 8 steps to solve. In the in-context examples, we provide the answer passage that contains intermediate steps with calculator annotations removed. The model is expected to provide the final numerical answer after four hashtags (####) that delimit the reasoning and final answer (Cobbe et al., 2021).

**HellaSwag** contains 4-way MC questions which ask for the most natural continuation given the context (Zellers et al., 2019).

**HotpotQA** contains questions along with a corresponding passage from Wikipedia containing the answer (Yang et al., 2018).

**HumanEval** contains programming problems where the model is tasked with completing a Python function given its docstring (Chen et al., 2021). We use "`\nclass`," "`\ndef`," "`\n#`," "`\nif`," as stop tokens. Following the original paper, we sample 20 continuations with top $p = 0.95$ and temperature $= 0.8$. Models are allowed to generate for a maximum of 128 new tokens. The functional correctness of generations is automatically evaluated using test cases. We use the 20 generation to make an unbiased estimate of the pass@10 rate, i.e., how likely at least one of 10 sampled solutions for a problem is correct.

**Jeopardy** contains open-ended questions from the "Jeopardy!" quiz show. [11]

**Lambada** contains narratives without the last word, which is inferrable given the context (Paperno et al., 2016). This task requires models to attend to the full narrative instead of only the local context.

**BB Language Identification** contains sentences in different languages, and the task is to choose the language of the sentence from a long list of options.

**LSAT-AR** contains MC questions that evaluate the analytical reasoning (AR) ability of LMs (Zhong et al., 2022; 2024). Test questions are drawn from the Law School Admission Test (LSAT) from 1991 to 2016.

**MBPP** contains Python programming problems where the model is given a description of the desired function and a series of unit tests. We use the same evaluation setup as HumanEval.

**MMLU** contains 4-way MC questions covering 57 different domains, covering both world knowledge and problem-solving abilities (Hendrycks et al., 2021). Note that we report a straight average over the 5000-example sample, rather than a macro-average over subjects.

**OpenbookQA** contains 4-way MC questions that require multi-step reasoning and commonsense knowledge (Mihaylov et al., 2018).

**BB Operators** contains questions where the model is given a function definition and asked to compute the output of that function given a particular input.

**PIQA** contains MC questions that require physical commonsense reasoning (Bisk et al., 2020).

**BB Repeat-Copy-Logic** contains instructions that ask the model to produce a particular string (Austin et al., 2021).

**SQuAD** contains passages paired with questions about the passage (Rajpurkar et al., 2016). The answer is always a span from the passage.

[11] https://www.kaggle.com/datasets/tunguz/200000-jeopardy-questions

| Category | Task | BPE 8B | SuperBPE 8B | | | SuperBPE 11B |
|---|---|---|---|---|---|---|
| | | | $t = 80$k | $t = 160$k | $t = 180$k | |
| Knowledge | ARC-Easy (MC) | 46.6 | 60.8 | 63.6 | **67.1** | 60.6 |
| | ARC-Challenge (MC) | 35.1 | 46.4 | 43.9 | **50.6** | 43.9 |
| | Jeopardy (EM) | 42.1 | 40.2 | 41.8 | 41.8 | **42.2** |
| | MMLU (MC) | 36.5 | 41.9 | 42.6 | **44.7** | 41.0 |
| | OpenbookQA (MC) | 33.2 | 49.8 | 49.4 | **54.4** | 46.4 |
| | TriviaQA (EM) | 60.6 | 59.7 | 61.9 | 61.3 | **62.3** |
| | WikidataQA (EM) | 69.7 | 68.2 | 69.5 | **70.9** | **70.9** |
| Math & Reasoning | Arithmetic (EM) | 54.8 | **63.2** | 58.6 | 59.3 | 56.9 |
| | GSM8K (EM) | 6.4 | 6.9 | 6.7 | 6.7 | **7.4** |
| | LSAT-AR (MC) | 21.3 | 23.9 | **24.3** | 23.0 | 20.9 |
| | Operators (EM) | 35.5 | 32.2 | 35.5 | 33.6 | **37.9** |
| | Repeat-Copy-Logic (EM) | 3.1 | **6.2** | **6.2** | **6.2** | 3.1 |
| Coding | HumanEval (pass@10) | **15.9** | 15.0 | 14.4 | 13.4 | **15.9** |
| | MBPP (pass@10) | 27.5 | 25.3 | 28.4 | 28.3 | **29.4** |
| Reading Comprehension | BoolQ (MC) | 59.7 | **65.2** | 62.3 | 64.6 | 64.7 |
| | CoQA (EM) | 12.6 | 12.8 | 12.5 | **13.2** | 13.1 |
| | DROP (EM) | 31.3 | 28.6 | 32.8 | 31.4 | **33.1** |
| | HotpotQA (EM) | 53.5 | 52.5 | 54.7 | **55.2** | 54.6 |
| | SQuAD (EM) | 75.1 | 74.3 | 76.2 | 75.8 | **77.2** |
| Commonsense | CommonsenseQA (MC) | 33.5 | 50.0 | 52.3 | **53.8** | 50.5 |
| | COPA (MC) | 77.0 | 86.6 | 87.6 | 85.8 | **97.0** |
| | PIQA (MC) | 55.2 | 57.7 | **61.8** | 59.8 | 59.2 |
| | Winograd (MC) | 50.4 | 52.5 | **55.2** | 53.1 | 52.3 |
| | Winogrande (MC) | 47.3 | 51.2 | 51.6 | **52.6** | 50.2 |
| Language Understanding | HellaSwag (MC) | 29.7 | 31.2 | 30.3 | 33.7 | **36.6** |
| | LAMBADA (EM) | **77.0** | 72.8 | 75.1 | 70.6 | 75.8 |
| | Language Identification (EM) | 8.8 | **10.2** | 9.7 | 9.0 | 10.1 |
| String Manipulation | CS Algorithms (EM) | 46.1 | 47.3 | 42.6 | 48.6 | **49.1** |
| | CUTE (EM) | 31.3 | 32.2 | 32.8 | 32.6 | **35.7** |
| | Dyck-Languages (EM) | 15.9 | **23.2** | 18.8 | 14.2 | 16.7 |
| Average | | 39.8 | 42.9 | 43.4 | **43.8** | **43.8** |

Table 5: **Performance of BPE and SuperBPE models on 30 downstream tasks.** This is an expansion of Table 1 with more models.

**TriviaQA** contains open-ended questions about world knowledge (Joshi et al., 2017).

**BB WikidataQA** require models to complete factual statements with the correct continuation.

**Winograd** contains binary MC questions where the model is given a context and asked to determine which entity a pronoun refers to, between two options (Levesque et al., 2012). Correctly answer the question requires commonsense knowledge and contextual reasoning.

**Winogrande** contain questions with the same schema as Winograd, but increases both the scale and difficulty of the dataset (Sakaguchi et al., 2021).

# B Additional Results

## B.1 Task evaluation

We report the individual task performance of BPE and all SuperBPE models in Table 5 (this an expansion of Table 1). We also show a subset of task-specific performance curves during pretraining in Figure 11.

## B.2 BPB evaluation

See Figure 6 for the bits-per-byte during pretraining of all models we compare.

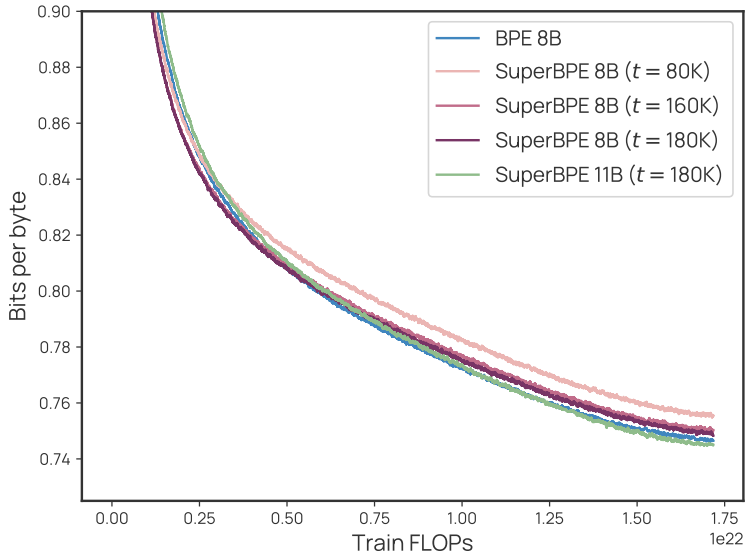



Figure 6: **Bits-per-byte of BPE and SuperBPE models during pretraining**. The BPE 8B, SuperBPE 8B ($t = 180$k), and SuperBPE 11B attain 0.7465, 0.7482, and 0.7445 BPB respectively at the end of training.

### B.3 Additional scaling experiments

Our tokenizer has several interesting interactions with LM scaling, purely due to its increased efficiency. For the purpose of this section, let $\alpha$ denote the ratio of our tokenizer's efficiency to the efficiency of a normal BPE tokenizer. (For example, we have $\alpha \approx 1.49$ for our most efficient tokenizer.)

The primary advantage of a more efficient tokenizer is a reduction of the context length (in tokens) for the same effective context length (in bytes). All other things being equal, this gives:

1. A $1/\alpha^2$ reduction in attention compute.
2. A $1/\alpha$ reduction in non-attention compute.
3. A $1/\alpha$ reduction in activation memory during training and KV-cache size during inference.

Thus, if the context length is short, the total compute savings will be close to $1/\alpha$. For longer contexts, the compute savings may approach $1/\alpha^2$. Given a fixed training budget, there are two natural ways to convert these savings into improved performance.

#### *B.3.1 Matching model parameter count*

In many applications of language models, such as deployment to consumer or edge devices, it is crucial to keep the model's size under control. In this regime, we will assume the model size fixed. This directly grants the aforementioned benefits, and we will turn to increasing the number of training steps to match the training budget.

Since the amount of text seen per step is remains the same due to the fixed effective context length, a more efficient tokenizer allows the model to see more text during training for the same budget. This may lead to improved performance on downstream tasks since the model is more likely to have seen relevant training examples during training. Additionally, although the model is the same size, it requires less compute and memory at inference time to perform the same tasks. In some settings, these gains can be used to amplify inference-time scaling (Snell et al., 2024), leading to further potential gains.

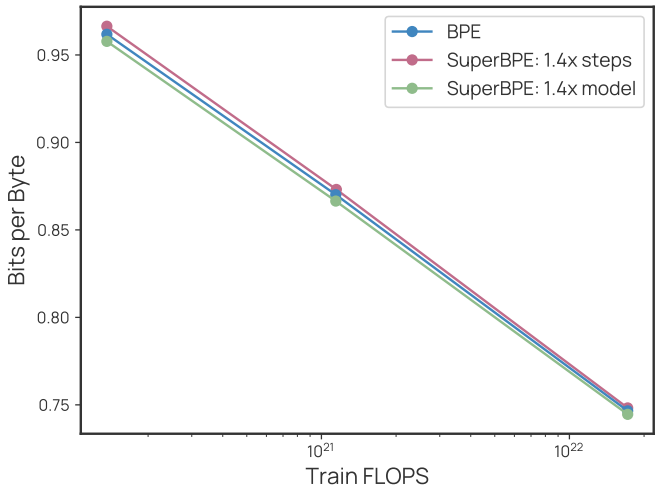



Figure 7: Results for scaling both model parameters and train tokens proportionally. Here we see the spread between the three settings decreases with scale.

#### *B.3.2 Matching inference compute*

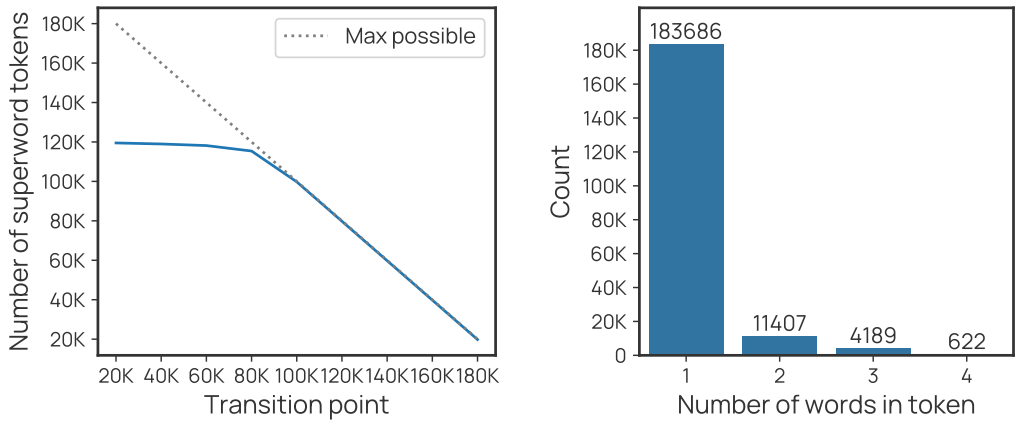
In other applications of language models, model size is less critical compared to inference compute. In these situations, it may be more desirable to scale the model size up to absorb the extra compute.

Changing the model size has a strong impact on scaling. Depending on the context length, we may scale the model by a factor of anywhere between $\alpha$ and $\alpha^2$ in order to match inference compute. Since each training step involves $1/\alpha$ as many tokens, the ratio of tokens to model parameters per step may be reduced by as much as $1/\alpha^3$. Prior work on LM scaling (Hoffmann et al., 2022; Kaplan et al., 2020) reports diminishing gains once the ratio of the numbers of train tokens and model parameters becomes too large. An $\alpha$ times more efficient tokenizer allows us to train for up to $\alpha^3$ times longer while maintaining the same token/parameter ratio and without increasing inference compute, delaying the regime of diminishing gains.

#### *B.3.3 Experiments*

We train 680M and 1.9B sized BPE models on various numbers of tokens—ranging from $\approx 20$ to $\approx 80$ tokens per parameter—to establish a baseline scaling trend. We then train two models with SuperBPE tokenizers for each baseline model: one with matching parameter count and one with matching inference compute cost.

There are a couple interesting ways to visualize these results: in Figure 5, we hold the model size fixed and increase the number of training tokens, and in Figure 7, we hold the ratio of train tokens to model parameters fixed (inference compute matched will be fixed 0.7 times lower) and vary both the model size and the number of training tokens. The general trends observed from these results are that matching inference compute is almost universally the best, while matching parameter count tends to be worse than the baseline except in the undertrained regime, where it is better than the baseline. The differences between the different settings increases with overtraining, but decreases when scaling both model size and training tokens at the same time.




(a) Superword density

(b) Superword length distribution

Figure 8: (Left) The number of superword tokens in a SuperBPE tokenizer, as a function of the transition point. A superword token is any token that violates the whitespace pretokenization rule from Stage 1. With an early transition point of $t = 60$K, about 85% of the tokens learned in Stage 2 are superword tokens. For $t > 100$k, close to 100% of Stage 2 tokens are superwords. (Right) The distribution of superword token lengths in terms of number of words, for $t = 180$k.

## C Analysis of SuperBPE Tokenizers

### C.1 Superword token analysis

How many superword tokens are in SuperBPE tokenizers? While the second stage of the pretokenization curriculum allows learning of superword tokens, subword tokens can still be learned. Shown in Figure 8a, for transition points $t < 80$k, the number of superword tokens is relatively steady around 120k. Past $t > 100$k, almost all tokens learned in Stage 2 are superword tokens. Figure 8b shows the number of whitespace-delimited words in the superword tokens of SuperBPE with $t = 180$k.

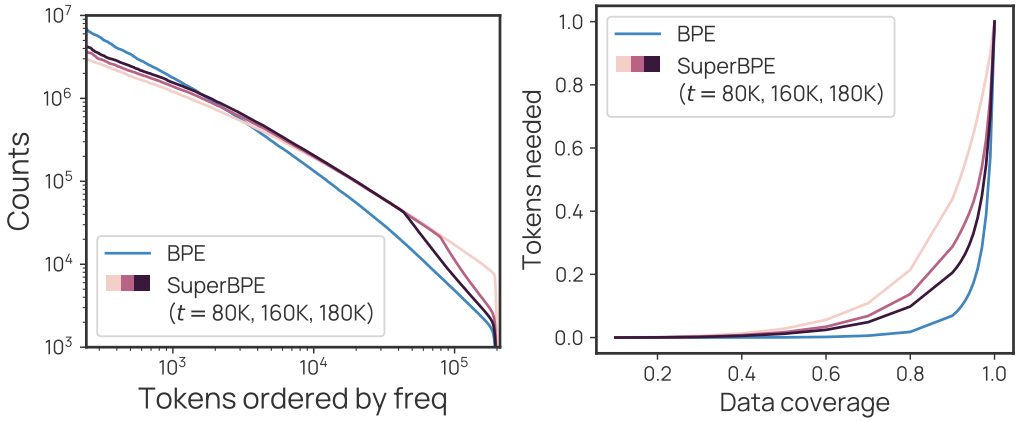
### C.2 Analysis of token frequencies in encoding

We also analyze token frequency statistics under BPE versus SuperBPE tokenizers. Figure 9a shows the relation between token rank (in frequency) and frequency. While tokens in BPE demonstrate a standard Zipfian relation, the slope of SuperBPE curves have a more shallow slope, meaning that the rate of decay in token frequency is smaller. The smaller proportion of tokens with very low counts may reduce prevalence and severity of glitch tokens (Rumbelow & Watkins, 2023; Land & Bartolo, 2024).

Figure 9b shows the minimum number of tokens from the vocabulary needed to cover any given proportion of data. For BPE, the relation is striking—only 57% of tokens are needed to encode 99% of the data! The remaining tokens make up a long tail of infrequent tokens. In contrast, SuperBPE tokenizers make better use of the vocabulary. For $t = 80$k and $t = 180$k, this statistic is 90% and 70% of tokens, respectively.

### C.3 Distributional Distortion at the Prompt Boundary

Prior work (Lundberg, 2023; Phan et al., 2024) has shown that LMs using BPE tokenizers may produce distorted generations due to the forced partition in tokenization between a prompt and its completion. This issue stems from the fact that users typically desire completions conditioned on a text prompt. The natural approach to obtaining such completions is to take the prompt, tokenize it with the proper tokenizer, and then sample a completion of the resulting token sequence from the LM.




(a) Token frequency distribution

(b) Data covering

Figure 9: (Left) Token counts when ordered by frequency. The rate of decay in token frequency is smaller. (Right) The minimum number of tokens needed to cover a given proportion of the data. SuperBPE tokenizers make better use of the vocabulary, while BPE tokenizers have a long tail of infrequent tokens.

For a simple example of how this can go wrong, consider a tokenizer with base vocabulary of `A` and `B` and a single merge forming the token `AB`. Let's suppose we trained a model using this tokenizer on the strings "AA", "AB", and "BB" with equal proportions. If we condition on the text prefix "A", there are two equally probable continuations: "A" and "B". However, `A` is the only valid completion of the token prefix `A`, since the token `B` never follows the token `A` during training. In other words, the prompt-completion pair `(A,B)` is canonically tokenized using a token that crosses the boundary between the prompt and the completion.

While this problem is shared by all BPE tokenizers, it can be partially mitigated by pretokenization: if the prompt and the completion are separated during the pretokenization step, then it is impossible for a token to cross the boundary. This fix tends to work well for English, where the completion is typically expected to begin with whitespace, so whitespace pretokenization would apply. However, there are many settings where whitespace pretokenization cannot fix the underlying issue, including natural languages that do not use whitespace to separate words (like Chinese and Japanese), programming languages, and constrained generation (Lundberg, 2023; Ribeiro, 2023).

Several fixes for this issue have been proposed: at training time, token merges can be randomly dropped (Provilkov et al., 2020; Sims et al., 2025; DeepSeek-AI et al., 2025) to expose LMs to the internal makeup of tokens; at inference time, options include token healing (Lundberg, 2023), algorithmic correction (Phan et al., 2024), and enumeration of all relevant segmentations of the prompt (Vieira et al., 2024). We leave a detailed comparison of these techniques to future work.

Additionally, the issue does not apply at all to models that separate the user's input from the model's response using special tokens, as is typical for chat models.

## D Other Related Work

Please see Mielke et al. (2021) for a survey of subword tokenization.

**Pretokenization** Pretokenization defines how the text is split in order to prevent certain pairs of tokens from being merged. GPT-2 (Radford et al., 2019) introduced a regular expression (regex) which defines the pretokenization pattern. These regex strings have gained complexity over time; GPT-3.5 limits the number of digits in numerical tokens to 3, and allows single punctuation to be merged with the start of words (presumably to accommodate code, as it allows `.get` to be a single token). Prior work has shown that, for

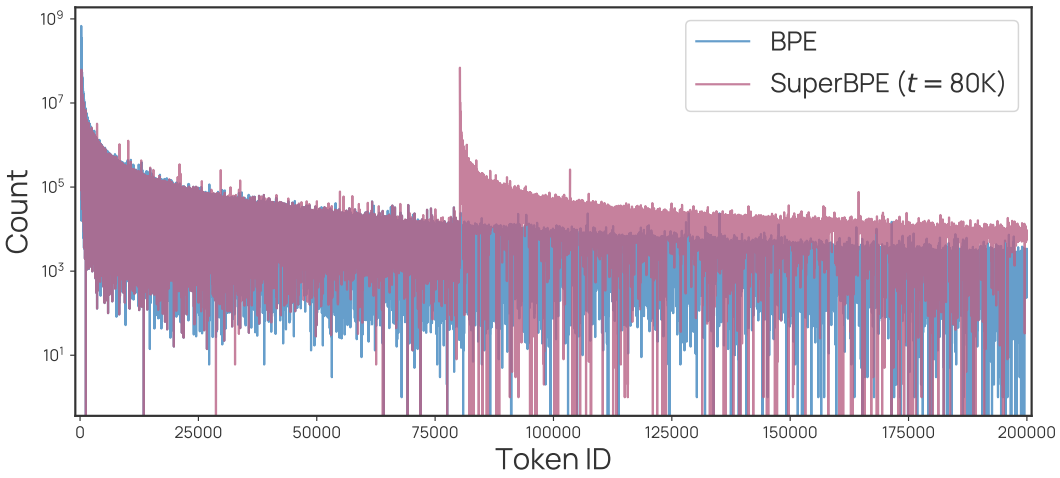



Figure 10: Token counts when ordered by token ID, which reflects the order in which tokens were learned in tokenizer training.

instance, digit pretokenization choices (Nogueira et al., 2021; Thawani et al., 2021; Singh & Strouse, 2024) can significantly impact arithmetic performance. It is also likely that pretokenization affects different languages differently (Velayuthan & Sarveswaran, 2025; Ahia et al., 2023), due to natural statistics of the average word length, which acts as an upper bound on encoding efficiency in that language under subword tokenization. Nonetheless, the effectiveness of many pretokenization choices have not been thoroughly studied.

**$n$-gram language models** Our work is loosely related to $n$-gram LMs, which incorporate $n$-gram statistics into the next-word prediction (Brants et al., 2007; Liu et al., 2024).

**Internal representation of semantic units** Previous work has showed that the early layers of the LM may "aggregate" information over multi-token entities (e.g., [_New, _York]) into the *last* token's (e.g., _York) hidden representation (Meng et al., 2022; Kaplan et al., 2025; Lad et al., 2024). This suggests that LMs naturally learn multi-word representations, and segmentating text into more semantically cohesive units at the input level (e.g., having _New_York as a single token) may simplify this process.

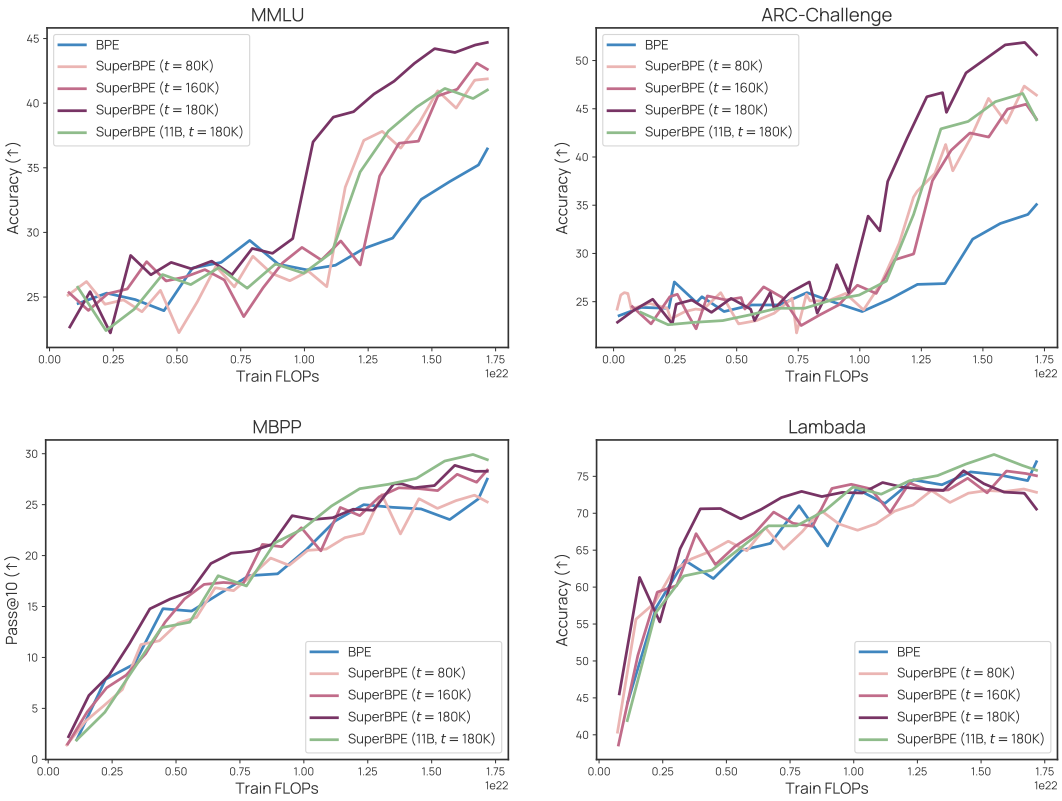



Figure 11: Performance during pretraining for a subset of tasks in our evaluation suite.