Broken Tokens? Your Language Model can Secretly Handle Non-Canonical Tokenizations

Anonymous Authors[1]

Abstract

Modern tokenizers employ deterministic algorithms to map text into a single "canonical" token sequence, yet the same string can be encoded as many non-canonical tokenizations using the language model vocabulary, including tokenizing by character. In this paper, we investigate the robustness of LMs to input encoded with non-canonical tokenizations entirely unseen during training. Surprisingly, when evaluated across 20 benchmarks, we find that instruction-tuned models retain up to 93.4% of their original performance when given a randomly sampled tokenization, and 90.8% with character-level tokenization. We find that overall stronger models tend to be more robust, and that robustness diminishes as the tokenization departs farther from the canonical form. Motivated by these results, we then identify settings where non-canonical tokenization schemes can *improve* performance, finding that character-level segmentation improves string manipulation and code understanding tasks by up to 15%, and right-aligned digit grouping enhances large-number arithmetic by over 33%. Finally, we investigate the source of this robustness, finding that it arises in the instruction-tuning phase. We show that while both base and post-trained models grasp the semantics of non-canonical tokenizations (perceiving them as containing misspellings), base models try to mimic the imagined mistakes and degenerate into nonsensical output, while post-trained models are committed to fluent responses. Overall, our findings suggest that models are less tied to their tokenizer than previously believed, and highlight the promise of intervening on tokenization at inference time to boost performance.

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

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

1. Introduction

Tokenizers segment text into a sequence of discrete tokens in the language model's (LM) vocabulary. Most of today's LMs use deterministic subword tokenization, which produces a single canonical token sequence for a given piece of text, and further, for each whitespace-delimited word. One commonly discussed limitation of this approach is that, by mapping byte strings to symbolic token IDs, the orthographic makeup of tokens is obscured to the LM (Provilkov et al., 2020; Edman et al., 2024). This can be especially harmful for LM understanding of numbers (Nogueira et al., 2021; Thawani et al., 2021; Singh & Strouse, 2024) and morphologically rich languages (Arnett & Bergen, 2025; Hofmann et al., 2021), and has motivated efforts to model text directly at the byte level (Clark et al., 2022; Xue et al., 2022; Wang et al., 2024b; Tay et al., 2022; Yu et al., 2023; Nawrot et al., 2023; Pagnoni et al., 2024; Ahia et al., 2024; Limisiewicz et al., 2024).

To shed light on this topic, in this work we investigate whether LMs can adapt *at inference time*, without any additional training, to a different tokenization scheme than the one they were trained with. While the tokenizer deterministically outputs a *canonical tokenization* of any text into tokens (usually by applying an ordered list of merge rules), *non-canonical tokenizations* of the same text using the same vocabulary are generally possible (see example in Figure 1). Here, we evaluate how LMs trained with deterministic tokenizers behave when given non-canonical tokenizations of text. Surprisingly, we find that *instruction-tuned LMs across many model families are extremely robust* to non-canonical tokenizations (section 2). For example, when evaluated across 20 benchmarks, `Qwen-2.5-7B-Instruct` retains 93.4% of its original performance when presented with a random non-canonical tokenization, and 90.8% when presented with character-level tokens (see Figure 1). Thus, far from not understanding the makeup of their tokens, LMs are able to compose token sequences in entirely new ways at inference time (Kaplan et al., 2025).

This leads to an intriguing possibility: if LMs can process non-canonical tokenizations, can we use different tokenization schemes at inference time to improve performance? For instance, prior work has found that better

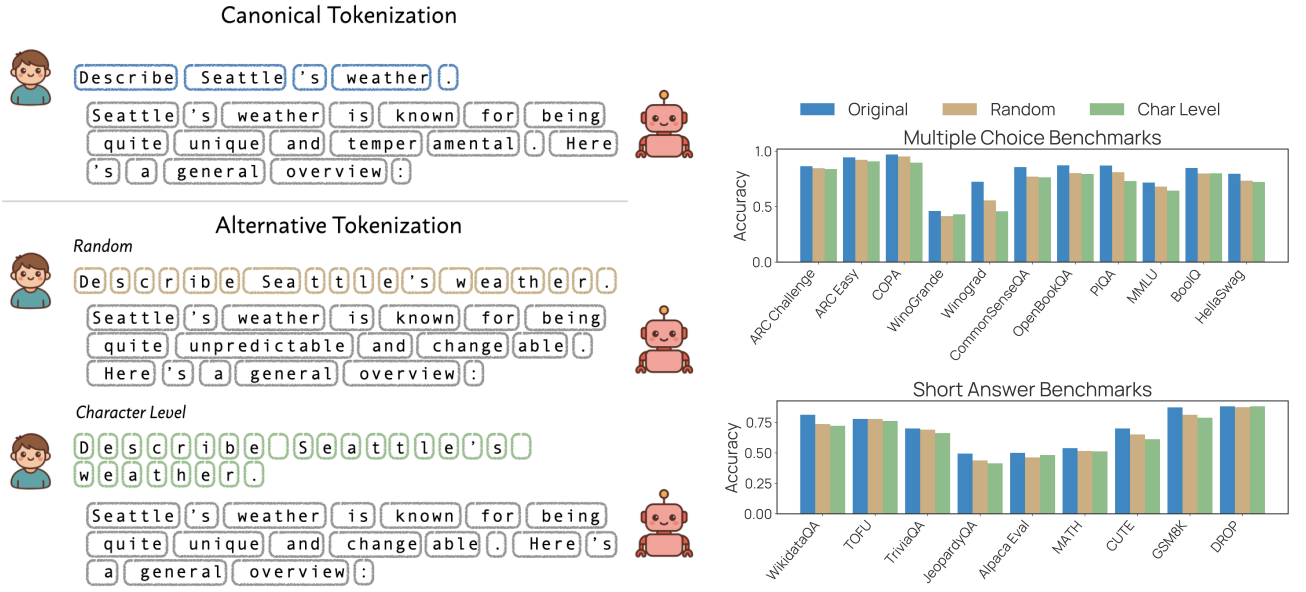

Figure 1. **Left:** An example of how `Llama-3.1-8B-Instruct` responds when given **canonically tokenized** input, versus a **random tokenization** and **character-level tokenization**. The responses are surprisingly similar, demonstrating their ability to handle non-canonical tokenizations. In particular, LMs generally respond with correctly tokenized output regardless of the tokenization scheme used for the context. **Right:** Performance of `Qwen-2.5-7B-Instruct` across various benchmarks and segmentation strategies. The model preserves much of its original performance when presented with non-canonical tokenizations. Note that even when presented with non-canonical tokenizations, models produce outputs that are canonically tokenized.

segmentation of large numbers can improve accuracy on arithmetic (Singh & Strouse, 2024; Sathe et al., 2025). Indeed, we identify several settings where non-canonical tokenization schemes dramatically improve performance for `Llama-3.1-Instruct` (section 3). Character-level tokenization brings up to +15% improvement on string manipulation and code understanding tasks, perhaps by granting LMs more direct access to orthographic cues. Meanwhile, right-aligned digit groups, which provide a consistent grouping of digits by powers of a thousand, improves arithmetic on large numbers by +33%. These performance gains are achieved without any finetuning, pointing to the promise of tokenization as a means of inference-time control.

Finally, we investigate the origin of model robustness to non-canonical tokenizations (section 4). Across multiple model families, we find that pretrained-only LMs consistently fail to produce fluent continuations given non-canonically tokenized context. By studying models at different stages of post-training, we identify that robustness arises during the supervised instruction-tuning (SFT) phase (subsection 4.1). We then ablate differences between pretraining and SFT procedures and find that the separation of the instruction and response as distinct turns of conversation is key (subsection 4.2). From here, we provide evidence for a plausible explanation: while both base and post-trained models retain the semantics of non-canonical tokenizations, they also perceive them as containing misspellings (subsection 4.2). Base models attempt to mimic the imagined mistakes and degenerate into nonsense, whereas post-trained models are not bound by the style of the instruction and thus able to produce fluent responses.

Overall, despite being trained with deterministic tokenization, instruction-tuned LMs readily accommodate new tokenizations at inference time, suggesting that LMs are less constrained by their tokenizer than previously believed (Minixhofer et al., 2024). Moreover, in settings where different representations of text are beneficial, we can intervene on tokenization at inference time for performance gains. We hope our work sheds new light on the discussion of strengths and limitations of tokenization, and points to the possibility of dynamically finding the optimal representation of text after pretraining.

2. Language Models are Robust to Non-Canonical Tokenizations

In our main experiments, we evaluate the robustness of LMs to non-canonical tokenizations by comparing their performance on downstream tasks when given different tokenizations of the input.

Table 1. **Evaluated across many benchmarks, models are surprisingly robust to non-canonical tokenizations of the context.** We show the absolute drop in performance when given a randomly sampled non-canonical tokenization (**Rand** Δ) and character-level tokenization (**Char** Δ), relative to the canonical (**Canon**) tokenization. We also summarize the model's ability to retain performance across benchmarks and tokenization strategies (bottom).

	QWEN-2.5-7B-INSTRUCT			LLAMA-3.1-8B-INSTRUCT			OLMO-2-7B-INSTRUCT		
Benchmark	Canon	Rand Δ	Char Δ	Canon	Rand Δ	Char Δ	Canon	Rand Δ	Char Δ
Multiple choice (MC)									
ARC-C	86.4	–1.80	–2.60	76.2	–14.10	–22.40	77.0	–37.40	–44.60
ARC-E	94.4	–2.20	–3.60	91.3	–12.60	–21.50	85.4	–37.00	–49.00
COPA	97.0	–1.80	–7.40	97.2	–9.60	–14.80	93.8	–21.40	–33.60
Winogrande	46.0	–4.60	–3.00	59.6	+2.00	–5.00	58.6	–7.80	–7.00
Winograd	72.4	–16.80	–26.60	74.4	–9.40	–13.00	72.4	–8.60	–19.00
CSQA	85.6	–8.60	–9.20	77.6	–11.40	–20.00	75.4	–31.60	–40.00
OpenbookQA	87.2	–7.00	–7.80	82.0	–13.80	–20.20	76.2	–30.80	–40.80
PIQA	87.0	–6.00	–14.00	84.0	–12.60	–18.40	78.2	–17.40	–25.00
MMLU	71.7	–3.70	–7.30	68.2	–11.60	–24.00	59.5	–16.30	–29.10
BoolQ	84.8	–5.00	–4.80	86.2	–19.20	–17.20	71.0	–4.00	–9.20
HellaSwag	79.6	–6.20	–7.40	68.6	–14.20	–23.40	68.0	–26.80	–39.80
Short answer (SA)									
WikidataQA	81.2	–7.60	–9.00	78.6	–12.40	–18.00	73.2	–28.80	–32.20
TOFU	77.8	+0.00	–1.70	82.1	+1.70	+0.80	82.9	–12.80	–23.90
TriviaQA	70.0	–1.00	–3.80	76.6	–9.80	–13.60	70.0	–22.20	–34.80
JeopardyQA	49.4	–5.60	–8.00	43.6	–2.20	–10.20	42.6	–21.60	–24.20
AlpacaEval	50.0	–3.70	–1.80	50.0	–5.70	–7.50	50.0	–2.10	–11.30
MATH	53.9	–2.40	–2.70	32.0	–4.20	–9.70	22.7	–5.20	–9.20
CUTE	70.0	–4.90	–8.80	68.0	–11.10	–15.30	55.3	–10.20	–5.70
GSM8K	87.3	–6.10	–8.50	82.0	–11.70	–16.00	73.9	–23.10	–35.80
DROP	88.2	–0.80	+0.00	88.8	–0.60	–5.00	77.0	–5.60	–7.60
Avg MC Retention (%)		$92.4_{\pm 5.97}$	$89.2_{\pm 9.33}$		$85.6_{\pm 6.86}$	$76.8_{\pm 7.99}$	$71.2_{\pm 14.7}$	$59.2_{\pm 17.8}$	
Avg SA Retention (%)		$94.6_{\pm 3.97}$	$92.7_{\pm 5.35}$		$90.3_{\pm 6.78}$	$82.7_{\pm 9.65}$	$75.4_{\pm 15.1}$	$65.4_{\pm 17.4}$	
Avg Overall Retention (%)		$93.4_{\pm 5.15}$	$90.8_{\pm 7.81}$		$87.7_{\pm 7.05}$	$79.4_{\pm 9.05}$	$73.1_{\pm 14.7}$	$62.0_{\pm 17.5}$	

2.1. Background

Most LMs today, and all the models we study, use the *Byte-Pair Encoding* (BPE) (Sennrich et al., 2016) algorithm for tokenization. The BPE tokenizer is learned by splitting a corpus of text into bytes, which form the initial vocabulary, then iteratively merging the most frequent pair of tokens into a new token that is added to the vocabulary. To encode a new text, it is split into bytes, and the learned merges are applied in the same order. As a result, a BPE tokenizer always produces the same token sequence for the same text. Further, because BPE tokens do not cross whitespace boundaries, the same whitespace-delimited word is always represented with the same token or token sequence.

A natural observation is that given a tokenizer vocabulary, there exist many token sequences that decode to the same text. For instance, *cat* could be tokenized as [`cat`], [, `cat`], [, `c`, `at`], [, `c`, `a`, `t`], etc. In general, the number of non-canonical tokenizations grows exponentially with the length of the text. Many previous works have argued that the probability of a string should be calculated as the sum of probabilities of all possible tokenizations (Cao & Rimell, 2021; Chirkova et al., 2023; Geh et al., 2024). However, less attention has been paid to how non-canonical tokenizations affect LMs in generative settings.

2.2. Setup

We consider two non-canonical tokenization schemes: (1) *Random tokenization* produces a tokenization (uniformly at random) from the set of tokenizations more granular than the canonical one. This can be achieved by recursively splitting individual tokens into a valid pair of tokens, similarly to (Sims et al., 2025); the pseudocode and proof of correctness is provided in Appendix A. (2) *Character-level tokenization* decomposes the string into character tokens, i.e., using no subword token from the vocabulary. For text containing only English letters and punctuation (where each character is exactly one byte), this produces the most granular possible tokenization.

We consider three models, `Llama-3.1-8B-Instruct` (Meta, 2024),

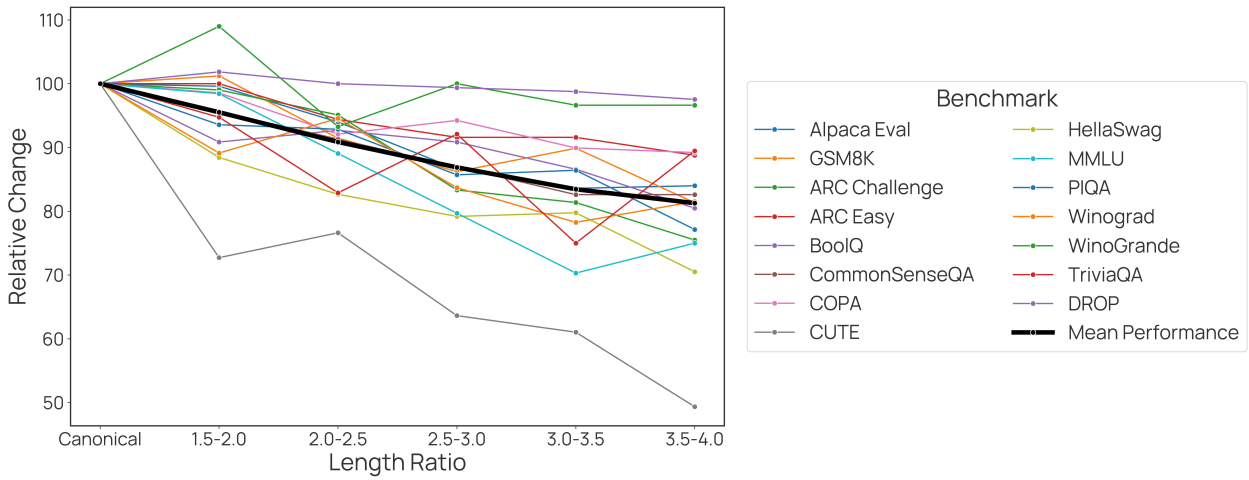

Figure 2. **Model performance generally declines as the tokenization becomes more granular.** We achieve variation in tokenization length using different values of p in BPE-dropout, and group tokenizations into buckets based on how many times longer it is than the canonical tokenization.

`OLMO2-7B-Instruct` (OLMo et al., 2024), and `Qwen-2.5-7B-Instruct` (Qwen, 2025), which we evaluate on 20 benchmarks shown in Table 1. Please see subsection B.1 for further description of the datasets and evaluation setup.

2.3. Results

Results in Table 1 show that while random tokenization consistently leads to worse performance compared to the canonical tokenization, the effect is small. On average across benchmarks, `Qwen-2.5` retains 93.4% of its performance when given random tokenization, followed by `Llama-3.1` at 87.7% and `Olmo-2` at 73.1%. The performance drops further with character-level tokenization, with retention of 90.8%, 79.4%, and 62.0% for the three models, respectively. This ranking of models in terms of retention is consistent with their ranking in absolute accuracy (under canonical tokenization), suggesting that stronger models are generally more robust to non-canonical tokenization strategies.

We also observe that all models retain performance better on short answer (SA) benchmarks (where the model generates an output in free-form) compared to multiple choice (MC) benchmarks (where the model is instructed to directly output the correct answer choice). We observe that LMs consistently *produce* correct token sequences even when conditioning on non-canonical tokenizations, and therefore hypothesize that, in the SA setting, models benefit from eventually conditioning on recent correctly-tokenized context.

2.4. Analysis: How does granularity of the tokenization affect robustness?

We next study whether tokenization fine-grainedness correlates in general with model robustness. We measure the fine-grainedness of a given non-canonical tokenization by how many times longer it is (in tokens) than the canonical tokenization, which we call the "length ratio." Finer-grained tokenizations have higher ratios, while coarser ones have ratios closer to 1. We produce tokenizations with diverse length ratios by applying BPE dropout (Provilkov et al., 2020) with $p \in [0.1, 0.2, ..., 0.9]$, which controls the probability with which each merge is dropped. (High p leads to finer-grained segmentations, and $p = 0.0$ corresponds to conventional BPE.)

Figure 2 shows the relationship between the length ratio and the average performance retention relative to canonical tokenization, with finer-grained tokenization generally leading to worse performance. When performance retention is averaged across tasks, the negative correlation is statistically significant under Kendall's τ with $p = 0.003$.

3. Can non-canonical tokenizations *improve* model performance?

If LMs can process non-canonical tokenizations, this points to the exciting possibility that tokenization schemes can be modified solely at inference-time. This would be useful if, in certain settings, there exists a better representation of text than what the tokenizer produces. In this section, we develop a suite of tasks that intuitively require understanding of the orthography of the text, and show that

Table 2. Examples from tasks we construct where non-canonical tokenizations lead to improved performance for `Llama-3.1-7B-Instruct` (section 3).

Counting characters: `Count the number of the letter 'r' in the word strawberry.`
Acronyms: `Come up with a sequence of words where the first letters would form this acronym: isman`
Common Morphemes: `What is the common morpheme among these words: alloenzyme, azyme, azymite, enzyme? A. vom- B. maxim- C. zym- D. log-`

Codeline Description: `What does the following code do:`
`{code block here}`
`A. Counts paths from a point to reach Origin`
`B. Program to check if a matrix is symmetric`
`C. Longest subsequence from an array of pairs having first element increasing and second element decreasing .`
`D. Count the number of strings in an array whose distinct characters are less than equal to M`

Arithmetic: `8492079913 + 4877278482 =`

`Llama-3.1-8B-Instruct` performs better under non-canonical tokenization schemes.

3.1. Tasks

Please see Table 5 for an example question in each task and Table B.2 for further details on dataset construction. For all tasks except Arithmetic, we use character-level tokenization.

Counting Characters This task asks the model to count the number of occurrences of a given letter (e.g., *r*) in 10-character tokens in `Llama-3.1`'s vocabulary, and contains 331 samples.

Acronyms This task asks models to generate a list of words whose first letters form a given acronym. We construct 3594 5-letter acronyms by sampling each letter uniformly at random from the alphabet.

Common Morphemes To test whether models understand the morphological makeup of words, we construct a task that asks models to identify the morpheme shared in common by a given set of words, among four multiple choice options. This task is built on top of the BIG-Bench task (bench authors, 2023) with the same name, by adding more examples from a public morpheme dataset.[1] Additionally, we modify the answer choices to contain the morphemes themselves (e.g., *ambul-*) rather than the meaning of the morphemes (e.g., *walk*). We sample three incorrect responses for each question from the other morphemes in the dataset. This task contains 1443 examples.

[1] https://github.com/colingoldberg/morphemes/tree/master/data

Code Description For a more real-world application, we create a task where each example provides a code snippet and asks the model to identify, from four options, the function of the code in natural language. Again we are inspired by a similar BIG-Bench task (bench authors, 2023) called Codeline Description, but to increase the difficulty of the examples we use more complex code snippets from XLCoST (Zhu et al., 2022). To collect incorrect answers, we sample three other code descriptions from the dataset. This task contains 4800 samples across 6 programming languages.

Arithmetic Prior work has suggested that arithmetic is difficult for LMs in part due to poor segmentation of digits (Nogueira et al., 2021; Thawani et al., 2021). We curate a simple arithmetic dataset by constructing addition and subtraction tasks for 10 digit numbers. Here, we use a different segmentation strategy. The `Llama-3.1` tokenizer segments numbers into groups of three left-to-right (e.g., *1000000* is encoded as ["`100`", "`000`", "`0`"]), due to the pre-tokenization regular expression looking for matches greedily from the left. Inspired by (Singh & Strouse, 2024), we instead segment digits into groups of three right-to-left (e.g., ["`1`", "`000`", "`000`"]). This task contains 1000 addition and subtraction questions in total.

3.2. Results

Shown in Table 3, in all the tasks we construct, the non-canonical tokenization strategy leads to substantially better performance compared to the canonical tokenization. In particular, we observe a +15.0% improvement on code description and +33.7% on arithmetic. Our results show that the tokenization scheme used in training is not necessarily the optimal one at inference-time, and replacing them with intuitively meaningful tokenizations can bring substantial

Table 3. **On several tasks, `Llama-3.1-8B-Instruct` achieves substantially *better* performance when using a non-canonical tokenization scheme**. For the first four tasks, the input is tokenized at the character level; for Arithmetic, we segment digits into groups of three digits from right to left.

Task	Canonical	Alternative	Δ
Counting Characters	67.9	**78.9**	+11.0
Acronyms	49.7	**56.7**	+7.00
Common Morphemes	94.7	**98.3**	+3.60
Code Description	68.1	**83.1**	+15.0
Arithmetic	36.5	**70.2**	+33.7

performance gains. We leave automatically identifying the optimal tokenization as a promising direction for future work.

4. Investigating the Source of Robustness

Thus far, our experiments have used post-trained "instruct" models. In this section, we find that pretrained-only models are actually unable to produce fluent continuations of usually tokenized context (subsection 4.1), and perform ablations to better understand the conditions enabling robustness (subsection 4.2).

4.1. When does robustness appear in model training?

We first quantify the robustness of models at different stages of the model development pipeline, by using the `Olmo2` and `Tulu3` model families which include the base, SFT, DPO, and final instruct models. For simplicity, we focus on AlpacaEval and use character-level tokenization. For base models, we construct the prompt by placing the instruction in a question-answer template (`Question: {instruction} Answer:`). We define three simple measures of generation quality.

Spelling We measure the proportion of (whitespace-delimited) words in the generation that can be found in a collection of the top 10,000 most common English words.[2]

Grammaticality We use LanguageTool's grammar checker[3] to count the number of grammatical mistakes, which we normalize by the number of words in the generation and subtract from 1 to produce a grammaticality score where higher is better.

Win rate To measure overall generation quality, we use `alpaca_eval_gpt4` as an LM judge in the AlpacaEval framework and report the win rate of the generation given alternative against canonical tokenizations of the context. Unlike the previous two metrics, this measures not only the quality of the generation but also its relevance to the context.

Shown in Figure 3, the base models of `Olmo2` and `Llama-3.1` are both unable to produce sensible output conditioned on character-level tokenizations of context, scoring at best 0.236 on spelling and 0.207 on grammaticality. Qualitatively, generations often involve odd character substitutions and repetitions (e.g., `Yoou`, `haviin`). However, despite the output being extremely difficult to parse, it sometimes reflects an understanding of the prompt. For example:

```
Question:  I like to host guests at my
home from time to time [...]
Can you give me a recipe for Canjeero?
Answer:  I aam glade tio hear tio hear
tio hear tio hear that yoou enjoy
haviin gauests at yoour hoome an tio
keeep tio keeep tio keeep
```

In contrast, the post-trained models are more robust across all three metrics, with *much of the improvement coming from the SFT stage alone*.

4.2. Why do instruction-tuned models become robust?

We first replicate the finding from subsection 4.1 that SFT yields robustness to non-canonical tokenizations by fine-tuning the `Llama-3.2-1B` base model on the `Tulu 3 SFT Personas Instruction Following` dataset. Then, we perform the following interventions on the SFT training data and procedure to shed light on the possible source.

Gradient over full sequence SFT on instruction-response pairs conventionally uses a loss mask over the instruction tokens, so that only the response tokens contribute to the loss. We remove this loss mask and instead compute gradients over the entire instruction and response.

[2] https://github.com/first20hours/google-10000-english

[3] https://github.com/languagetool-org/languagetool

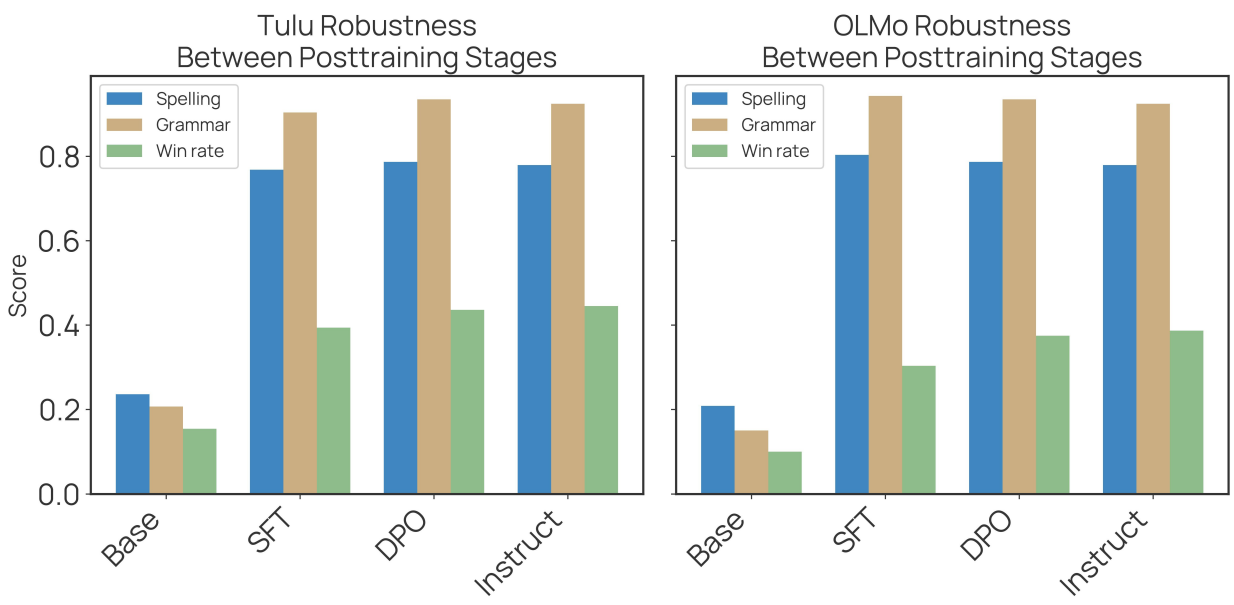

Figure 3. **Pretrained-only models completely fail to generate coherent output conditioned on non-canonical tokenizations of context; robustness is gained in the SFT stage.** We evaluate the spelling, grammaticality, and AlpacaEval win rate of model generations. Note that since `Tulu3` uses `Llama-3` as the base model, its base scores are computed using `Llama-3`'s base model scores.

Question/answer template We replace the chat template with a simple question-answer template, `Question: {instruction} Answer: {response}`, both for training and evaluation.

Removing the chat template We remove the chat template by concatenating the instruction and response without any special formatting. In evaluation, we again provide the instruction alone.

Removing the instruction After SFT training, the LM's goal is no longer to continue a given text prefix, but rather to generate a response to the given instruction. To ablate the nature of the data itself, we take only the responses from the SFT data, and randomly split each into a new "prompt" and "response," which we format with the SFT template.[4] At test time, we similarly provide an incomplete response within "instruction" tags. Since the purpose of the passage is generally inferrable from the first few words of the gold response ("*Sure, here's a recipe for Kubdari...*"), we are able to evaluate generated responses under the same AlpacaEval framework.

Our results are shown in Figure 4. We replicate the finding that SFT (**No ablation**) leads the model to be able to handle non-canonical tokenizations. This persists when computing gradients over the entire instruction and response (**Full gradient**) so that the training procedure matches regular pretraining. Replacing the original chat template with a simple question-answer template (**QA template**) also maintains model robustness. However, the usage of a template is crucial — when directly concatenating the instruction and response (**Removing chat template**), the model fails to produce coherent generations, with the spelling score dropping from 0.786 in the no ablation setting to 0.0698. Inserting the chat template into pretraining-style data (**Removing the instruction**) also does not yield robustness, with a spelling and grammaticality scores remaining low at 0.181 and 0.158, respectively. Overall, these findings suggest that in order for the LM to generate fluent continuations given non-canonical tokenizations, the context and expected continuation need to represent separate turns of dialogue, and additionally, be demarcated with a special template.

4.3. Disentangling understanding from generation

One plausible explanation for our findings thus far is that both base and instruction-tuned models grasp the semantics of non-canonical tokenizations, yet falsely perceive them as containing misspellings. While base LMs attempt to faithfully continue these mistakes and degenerate into nonsensical output, instruction-tuned models are trained to provide a fluent response regardless of the instruction. To

[4] We match the instruction length distribution by counting the number of tokens n in the original instruction, and formatting the first n tokens of the response as the new "instruction."

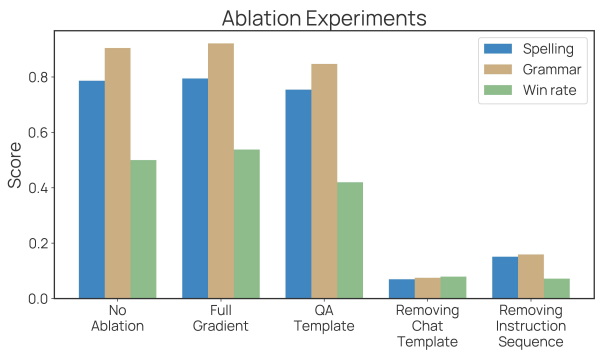

Figure 4. Ablations on the SFT training data and procedure indicate that the separation of the context and expected continuation — as different turns of dialogue demarcated with a special token — is key to robustness to non-canonical tokenizations.

test this hypothesis, we construct two simple tests:

1. **Word Repeat**: To determine if a model perceives the meaning of a word with non-canonical tokenization, we prompt the model to repeat a given word (while correcting any typos).

2. **Identifying Misspellings**: To determine if a model perceives a misspelling, we ask it to identify the word with a misspelling among two options: a (correctly tokenized) misspelling of a word and an non-canonical tokenization of that word (correctly spelled).

Results are shown in Table 4. Consistent with our hypothesis, we find that both the base and instruct models from the `Llama-3.1-8B` family score highly ($> 90\%$) on Word Repeat. This means that the base model, despite its poor performance in subsection 4.1, actually recognizes the correct form of non-canonical tokenizations as well as its post-trained counterpart. In addition, both models perform at random when asked to distinguish non-canonical tokenization from true misspellings. In other words, the instruct model produces fluent responses (section 2) while interpreting the instruction as heavily misspelled! While instruct models evidently overcome this, the base model likely attempts to mimic the (perceived) idiosyncratic surface form, thus producing nonsensical outputs.

5. Related Work

Robustness to Tokenization The extent to which LMs are limited by their tokenization is a topic of much debate, with the story evolving as LMs become larger and more capable. It is commonly argued that tokenization obscures orthographic information about tokens from the LM, leading to unexpected failures (Edman et al., 2024; Chai et al., 2024;

Table 4. Both base and instruct models from the `Llama-3.1` family recognize words represented with non-canonical tokenizations (performing well on **Word Repeat**), but incorrectly perceive that there are misspellings (performing at random on **Identifying Misspellings**).

	Word Repeat	Identifying Misspellings
`Llama-3.1-8B`	90.8	48.2
`Llama-3.1-8B-Instruct`	92.0	55.8

Wang et al., 2024a). As a result, there have been many efforts towards linguistically-informed tokenization that make derivational, compound, and morphological boundaries within words explicit (Klein & Tsarfaty, 2020; Hofmann et al., 2021; 2022; Yehezkel & Pinter, 2023; Bauwens & Delobelle, 2024). Similarly, BPE-dropout (Provilkov et al., 2020) and related methods (Sims et al., 2025) introduce variation in how a given string is tokenized to make models more robust to rare, misspelled, and unseen words.

However, other evidence suggests that LMs naturally overcome these limitations. For instance, token embeddings have been found to robustly encode character-level information, especially in larger models (Kaushal & Mahowald, 2022; Itzhak & Levy, 2022). This may be because word variants that do not share tokens in common (consider e.g., `[dictionary]` and `[diction, aries]`, as tokenized by GPT-2) incentivize the model to learn spelling as a general solution to understanding their relations (Kaushal & Mahowald, 2022). Other works argue that LMs maintain an implicit vocabulary, and can compose arbitrary token sequences (including non-canonical ones) into useful higher-level representations (Feucht et al., 2024; Kaplan et al., 2025). Even in domains like biomedical text where terms are highly agglutinative, using tokenizers that segment on meaningful components does not lead to improved models (Jimenez Gutierrez et al., 2023). Recent works have even found that coarser *superword* tokenization (Liu et al., 2025; Schmidt et al., 2025), which capture common word sequences in single tokens, can be a promising path forward.

Our work informs this conversation by showing that LMs can effectively leverage character-level knowledge of their tokens, and glean potential benefits of improved representation at inference time.

Non-Canonical Tokenizations It has long been recognized that there are many possible ways to segment string into tokens with a fixed vocabulary (Church, 2020), which in principle should be considered in the calculation of a string's likelihood (Cao & Rimell, 2021; Chirkova et al., 2023; Geh et al., 2024). In contemporaneous work, Geh et al. (2025) also show that LMs retain semantic understanding of non-canonical tokenizations. However, they only

study this over a small set of 15 examples, as their main focus is to show that non-canonical tokenizations can be constructed adversarially to trigger unsafe completions. In contrast, we provide a more systematic study of LM robustness using benchmark evaluations and perform experiments to understand the source.

Somewhat relatedly, other works have provided algorithms for sampling at the character- or byte-level from tokenizer-based LMs (Phan et al., 2024; Vieira et al., 2024; Athiwaratkun et al., 2024). Together, these directions suggest that despite being trained with one deterministic tokenization scheme, LMs can both condition on and produce token sequences over a different (sub)vocabulary.

6. Conclusion

We show that instruction-tuned language models are surprisingly robust to token sequences not seen in model training. In fact, in certain domains, such as arithmetic or code, more meaningful tokenizations can be swapped-in at inference time for improved performance. We investigate the source of this robustness, and find that instruction-tuning is key to language models producing fluent continuations to non-canonical tokenizations. Our work suggests that LMs are not necessarily tied to tokenizer they were trained with, and highlights the potential of finding more optimal representations of text after pretraining.

References

Ahia, O., Kumar, S., Gonen, H., Hofmann, V., Limisiewicz, T., Tsvetkov, Y., and Smith, N. A. 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.

Arnett, C. and Bergen, B. Why do language models perform worse for morphologically complex languages? In *Proceedings of the 31st International Conference on Computational Linguistics*, pp. 6607–6623, 2025.

Athiwaratkun, B., Wang, S., Shang, M., Tian, Y., Wang, Z., Gonugondla, S. K., Gouda, S. K., Kwiatkowski, R., Nallapati, R., Bhatia, P., and Xiang, B. Token alignment via character matching for subword completion. In Ku, L.-W., Martins, A., and Srikumar, V. (eds.), *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 15725–15738, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.929. URL https://aclanthology.org/2024.findings-acl.929.

Bauwens, T. and Delobelle, P. BPE-knockout: Pruning pre-existing BPE tokenisers with backwards-compatible morphological semi-supervision. In Duh, K., Gomez, H., and Bethard, S. (eds.), *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 5810–5832, Mexico City, Mexico, June 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.naacl-long.324. URL https://aclanthology.org/2024.naacl-long.324.

bench authors, B. 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.

Bisk, Y., Zellers, R., Bras, R. L., Gao, J., and Choi, Y. Piqa: Reasoning about physical commonsense in natural language. In *Thirty-Fourth AAAI Conference on Artificial Intelligence*, 2020.

Cao, K. and Rimell, L. You should evaluate your language model on marginal likelihood over tokenisations. In Moens, M.-F., Huang, X., Specia, L., and Yih, S. W.-t. (eds.), *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 2104–2114, Online and Punta Cana, Dominican

Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.161. URL https://aclanthology.org/2021.emnlp-main.161.

Chai, Y., Fang, Y., Peng, Q., and Li, X. Tokenization falling short: On subword robustness in large language models. In Al-Onaizan, Y., Bansal, M., and Chen, Y.-N. (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 1582–1599, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-emnlp.86. URL https://aclanthology.org/2024.findings-emnlp.86.

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

Church, K. W. Emerging trends: Subwords, seriously? *Natural Language Engineering*, 26(3):375–382, 2020. doi: 10.1017/S1351324920000145.

Clark, C., Lee, K., Chang, M.-W., Kwiatkowski, T., Collins, M., and Toutanova, K. BoolQ: Exploring the surprising difficulty of natural yes/no questions. In Burstein, J., Doran, C., and Solorio, T. (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.

Clark, J. H., Garrette, D., Turc, I., and Wieting, J. 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.

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

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., Hesse, C., and Schulman, J. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168.

Dua, D., Wang, Y., Dasigi, P., Stanovsky, G., Singh, S., and Gardner, M. DROP: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In Burstein, J., Doran, C., and Solorio, T. (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.

Dubois, Y., Li, X., Taori, R., Zhang, T., Gulrajani, I., Ba, J., Guestrin, C., Liang, P., and Hashimoto, T. B. Alpacafarm: A simulation framework for methods that learn from human feedback, 2023.

Edman, L., Schmid, H., and Fraser, A. CUTE: Measuring LLMs' understanding of their tokens. In Al-Onaizan, Y., Bansal, M., and Chen, Y.-N. (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.

Feucht, S., Atkinson, D., Wallace, B. C., and Bau, D. Token erasure as a footprint of implicit vocabulary items in LLMs. In Al-Onaizan, Y., Bansal, M., and Chen, Y.-N. (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 9727–9739, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.543. URL https://aclanthology.org/2024.emnlp-main.543.

Geh, R., Zhang, H., Ahmed, K., Wang, B., and Van Den Broeck, G. Where is the signal in tokenization space? In Al-Onaizan, Y., Bansal, M., and Chen, Y.-N. (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 3966–3979, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.230. URL https://aclanthology.org/2024.emnlp-main.230.

Geh, R. L., Shao, Z., and den Broeck, G. V. Adversarial tokenization, 2025. URL https://arxiv.org/abs/2503.02174.

Hendrycks, D., Burns, C., Basart, S., Zou, A., Mazeika, M., Song, D., and Steinhardt, J. Measuring massive multitask language understanding. In *International Conference*

on Learning Representations, 2021a. URL https://openreview.net/forum?id=d7KBjmI3GmQ.

Hendrycks, D., Burns, C., Kadavath, S., Arora, A., Basart, S., Tang, E., Song, D., and Steinhardt, J. Measuring mathematical problem solving with the MATH dataset. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021b. URL https://openreview.net/forum?id=7Bywt2mQsCe.

Hofmann, V., Pierrehumbert, J., and Schütze, H. Superbizarre is not superb: Derivational morphology improves BERT's interpretation of complex words. In Zong, C., Xia, F., Li, W., and Navigli, R. (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.

Hofmann, V., Schuetze, H., and Pierrehumbert, J. An embarrassingly simple method to mitigate undesirable properties of pretrained language model tokenizers. In Muresan, S., Nakov, P., and Villavicencio, A. (eds.), *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 385–393, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-short.43. URL https://aclanthology.org/2022.acl-short.43.

Itzhak, I. and Levy, O. Models in a spelling bee: Language models implicitly learn the character composition of tokens. In Carpuat, M., de Marneffe, M.-C., and Meza Ruiz, I. V. (eds.), *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 5061–5068, Seattle, United States, July 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.naacl-main.373. URL https://aclanthology.org/2022.naacl-main.373.

Jimenez Gutierrez, B., Sun, H., and Su, Y. Biomedical language models are robust to sub-optimal tokenization. In Demner-fushman, D., Ananiadou, S., and Cohen, K. (eds.), *The 22nd Workshop on Biomedical Natural Language Processing and BioNLP Shared Tasks*, pp. 350–362, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.bionlp-1.32. URL https://aclanthology.org/2023.bionlp-1.32.

Joshi, M., Choi, E., Weld, D. S., and Zettlemoyer, L. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension, 2017. URL https://arxiv.org/abs/1705.03551.

Kaggle. 200,000+ jeopardy! questions, 2019. URL https://www.kaggle.com/datasets/tunguz/200000-jeopardy-questions.

Kaplan, G., Oren, M., Reif, Y., and Schwartz, R. 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.

Kaushal, A. and Mahowald, K. What do tokens know about their characters and how do they know it? In Carpuat, M., de Marneffe, M.-C., and Meza Ruiz, I. V. (eds.), *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 2487–2507, Seattle, United States, July 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.naacl-main.179. URL https://aclanthology.org/2022.naacl-main.179.

Klein, S. and Tsarfaty, R. Getting the ##life out of living: How adequate are word-pieces for modelling complex morphology? In Nicolai, G., Gorman, K., and Cotterell, R. (eds.), *Proceedings of the 17th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology*, pp. 204–209, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.sigmorphon-1.24. URL https://aclanthology.org/2020.sigmorphon-1.24.

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

Limisiewicz, T., Blevins, T., Gonen, H., Ahia, O., and Zettlemoyer, L. MYTE: Morphology-driven byte encoding for better and fairer multilingual language modeling. In Ku, L.-W., Martins, A., and Srikumar, V. (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15059–15076, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.804. URL https://aclanthology.org/2024.acl-long.804.

Liu, A., Hayase, J., Hofmann, V., Oh, S., Smith, N. A., and Choi, Y. SuperBPE: Space travel for language models, 2025. URL https://arxiv.org/abs/2503.13423.

Maini, P., Feng, Z., Schwarzschild, A., Lipton, Z. C., and Kolter, J. Z. Tofu: A task of fictitious unlearning for llms, 2024. URL https://arxiv.org/abs/2401.06121.

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

Mihaylov, T., Clark, P., Khot, T., and Sabharwal, A. Can a suit of armor conduct electricity? a new dataset for open book question answering, 2018. URL https://arxiv.org/abs/1809.02789.

Minixhofer, B., Ponti, E., and Vulić, I. Zero-shot tokenizer transfer. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=RwBObRsIzC.

Nawrot, P., Chorowski, J., Lancucki, A., and Ponti, E. M. Efficient transformers with dynamic token pooling. In Rogers, A., Boyd-Graber, J., and Okazaki, N. (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.

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

OLMo, T., Walsh, P., Soldaini, L., Groeneveld, D., Lo, K., Arora, S., Bhagia, A., Gu, Y., Huang, S., Jordan, M., Lambert, N., Schwenk, D., Tafjord, O., Anderson, T., Atkinson, D., Brahman, F., Clark, C., Dasigi, P., Dziri, N., Guerquin, M., Ivison, H., Koh, P. W., Liu, J., Malik, S., Merrill, W., Miranda, L. J. V., Morrison, J., Murray, T., Nam, C., Pyatkin, V., Rangapur, A., Schmitz, M., Skjonsberg, S., Wadden, D., Wilhelm, C., Wilson, M., Zettlemoyer, L., Farhadi, A., Smith, N. A., and Hajishirzi, H. 2 olmo 2 furious, 2024. URL https://arxiv.org/abs/2501.00656.

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

Phan, B., Havasi, M., Muckley, M., and Ullrich, K. Understanding and mitigating tokenization bias in language models, 2024. URL https://arxiv.org/abs/2406.16829.

Provilkov, I., Emelianenko, D., and Voita, E. BPE-dropout: Simple and effective subword regularization. In Jurafsky, D., Chai, J., Schluter, N., and Tetreault, J. (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.

Qwen. Qwen2.5 technical report, 2025. URL https://arxiv.org/abs/2412.15115.

Roemmele, M., Bejan, C. A., and Gordon, A. S. Choice of Plausible Alternatives: An Evaluation of Commonsense Causal Reasoning. In *AAAI Spring Symposium on Logical Formalizations of Commonsense Reasoning*, Stanford University, March 2011. URL http://ict.usc.edu/pubs/Choice%20of%20Plausible%20Alternatives-%20An%20Evaluation%20of%20Commonsense%20Causal%20Reasoning.pdf.

Sakaguchi, K., Bras, R. L., Bhagavatula, C., and Choi, Y. 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.

Sathe, A., Aggarwal, D., and Sitaram, S. Improving consistency in LLM inference using probabilistic tokenization. In Chiruzzo, L., Ritter, A., and Wang, L. (eds.), *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 4766–4778, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. ISBN 979-8-89176-195-7. URL https://aclanthology.org/2025.findings-naacl.268/.

Schmidt, C. W., Reddy, V., Tanner, C., and Pinter, Y. Boundless byte pair encoding: Breaking the pre-tokenization barrier, 2025. URL https://arxiv.org/abs/2504.00178.

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

Sims, A., Lu, C., Kaleb, K., Foerster, J. N., and Teh, Y. W. 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.

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

Talmor, A., Herzig, J., Lourie, N., and Berant, J. CommonsenseQA: A question answering challenge targeting commonsense knowledge. In Burstein, J., Doran, C., and Solorio, T. (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.

Tay, Y., Tran, V. Q., Ruder, S., Gupta, J., Chung, H. W., Bahri, D., Qin, Z., Baumgartner, S., Yu, C., and Metzler, D. Charformer: Fast character transformers via gradient-based subword tokenization. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=JtBRnrlOEFN.

Thawani, A., Pujara, J., Ilievski, F., and Szekely, P. Representing numbers in NLP: a survey and a vision. In Toutanova, K., Rumshisky, A., Zettlemoyer, L., Hakkani-Tur, D., Beltagy, I., Bethard, S., Cotterell, R., Chakraborty, T., and Zhou, Y. (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.

Vieira, T., LeBrun, B., Giulianelli, M., Gastaldi, J. L., DuSell, B., Terilla, J., O'Donnell, T. J., and Cotterell, R. From language models over tokens to language models over characters, 2024. URL https://arxiv.org/abs/2412.03719.

Wang, D., Li, Y., Jiang, J., Ding, Z., Jiang, G., Liang, J., and Yang, D. Tokenization matters! degrading large language models through challenging their tokenization, 2024a. URL https://arxiv.org/abs/2405.17067.

Wang, J., Gangavarapu, T., Yan, J. N., and Rush, A. M. Mambabyte: Token-free selective state space model. In *First Conference on Language Modeling*, 2024b. URL https://openreview.net/forum?id=X1xNsuKssb.

Xue, L., Barua, A., Constant, N., Al-Rfou, R., Narang, S., Kale, M., Roberts, A., and Raffel, C. 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.

Yehezkel, S. and Pinter, Y. Incorporating context into subword vocabularies. In Vlachos, A. and Augenstein, I. (eds.), *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pp. 623–635, Dubrovnik, Croatia, May 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.eacl-main.45. URL https://aclanthology.org/2023.eacl-main.45.

Yu, L., Simig, D., Flaherty, C., Aghajanyan, A., Zettlemoyer, L., and Lewis, M. 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.

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

Zhu, M., Jain, A., Suresh, K., Ravindran, R., Tipirneni, S., and Reddy, C. K. Xlcost: A benchmark dataset for cross-lingual code intelligence, 2022. URL https://arxiv.org/abs/2206.08474.

A. Random Non-Canonical Tokenization

In this section, we provide our algorithm for producing a random non-canonical tokenization, and a proof that each non-canonical tokenization that is *more fine-grained* than the canonical one has equal probability of being output.

A.1. Algorithm

We will split each token in a canonical tokenization into smaller tokens (that each exists in the tokenizer's vocabulary). We formulate our problem as: Given a valid token t, and a set of vocabulary $\mathcal{V}$, construct a sequence of tokens seq using tokens that exist in $\mathcal{V}$ and form t when concatenated together. We produce seq using a recursive algorithm. Since there can be many possible seq for each t, we need to randomly choose one and guarantee that each possible seq is chosen with equal probability. We achieve this by considering recursion as producing a tree. Each path down the tree corresponds to one possible way to segment t. Each node of the tree represents a segmentation state where we have chosen some number of sub-tokens. At each node, we weigh the choice of which child node to visit by the number of leaves in the sub-tree that is rooted at each child node. This guarantees that each path down the tree is chosen with equal probability since the number of paths down a tree is equal to the number of leaf nodes in that tree. The pseudocode for the algorithm is in 1.

Algorithm 1 Random Token Segmentation

```
function COUNTSEGMENTS(start) {Cached (using memoization)}
    if start = |token| then
        return 1 {Reached end; valid segmentation}
    end if
    total ← 0
    for end ← start + 1 to |token| do
        substring ← token[start : end]
        if substring ∈ vocabulary then
            total ← total + COUNTSEGMENTS(end)
        end if
    end for
    return total
end function
function BUILDSEGMENTS(start)
    if start = |token| then
        return ∅ {Empty segmentation}
    end if
    validSegments ← []
    weights ← []
    for end ← start + 1 to |token| do
        substring ← token[start : end]
        if substring ∈ vocabulary then
            segCount ← COUNTSEGMENTS(end)
            if segCount > 0 then
                Append substring to validSegments
                Append segCount to weights
            end if
        end if
    end for
    if validSegments is empty then
        return ∅
    end if
    chosenSegment ← weightedRandomChoice(validSegments, weights)
    return [chosenSegment] ∥ BUILDSEGMENTS(start + |chosenSegment|) {Concatenate chosen segment with
segmentation of remaining token}
end function
procedure SEGMENTTOKEN(token, vocabulary)
    if COUNTSEGMENTS(0) = 0 then
        return ∅ {No valid segmentation exists}
    else
        return BUILDSEGMENTS(0)
    end if
end procedure=0
```

A.2. Proof

Goal: To prove that the random segmentation algorithm chooses one valid segmentation from all possible valid segmentations with uniform probability.

Notation: Let $W(i)$ denote the number of valid segmentation completions (i.e., the number of leaves in the recursive tree) for the substring starting at index i. In particular, $W(|token|) = 1$. Note that $W(i)$ is calculated by the memoized recursive function $countSegments(i)$, which calculates the number of leaves of the subtree rooted at i.

Base Case: Consider the node corresponding to $i = |token|$ (the end of the token). Here, there is exactly one valid segmentation (the empty segmentation), so the algorithm returns it with probability 1. That is, every segmentation (in this case, the only one) is chosen with probability

$$\frac{1}{W(|token|)} = \frac{1}{1} = 1.$$

Thus, the base case holds.

Inductive Hypothesis: Assume that for any node corresponding to an index j with $j > i$ (i.e., deeper in the recursion tree), every complete segmentation (leaf) in the subtree rooted at j is chosen with probability

$$\frac{1}{W(j)}.$$

Inductive Step: Now consider a node corresponding to index i (with $i < |token|$). Suppose that from i there are k valid branches corresponding to choosing substrings that end at indices $j_1, j_2, \ldots, j_k$, where for each j we have $i < j \leq |token|$ and the substring $token[i : j]$ is in the vocabulary. By definition,

$$W(i) = \sum_{r=1}^{k} W(j_r).$$

The algorithm selects the branch from i to a specific child j with probability

$$P(i \to j) = \frac{W(j)}{W(i)}.$$

Once branch $i \to j$ is chosen, by the inductive hypothesis every complete segmentation (leaf) in the subtree rooted at j is chosen with probability

$$\frac{1}{W(j)}.$$

Thus, the probability $P(S)$ of obtaining a particular complete segmentation $\mathcal{V}$ that starts at i by first taking the branch $i \to j$ and then following a specific path in the subtree rooted at j is

$$P(S) = \frac{W(j)}{W(i)} \cdot \frac{1}{W(j)} = \frac{1}{W(i)}.$$

Since the factor $W(j)$ cancels, the probability $P(S)$ is independent of the particular child j chosen.

Conclusion: By the inductive step, every complete segmentation (leaf) in the subtree rooted at any index i is chosen with probability $\frac{1}{W(i)}$. In particular, when $i = 0$ (the start of the token), every valid segmentation of the entire token is selected with uniform probability $\frac{1}{W(0)}$. This completes the proof.

B. Evaluation Details

B.1. General benchmarks

For short-answer benchmarks, the system prompt is:

```
You are a helpful assistant.
```

For multiple-choice benchmarks, the system prompt is:

```
You are a helpful assistant. For the following multiple choice questions,
return the answer only, without any additional reasoning or explanation.
```

MATH MATH is a dataset composed of fairly difficult, competition level math problems (Hendrycks et al., 2021b). The test set is composed of short answer problem that describe some scenario and asks the model to output a mathematically correct answer.

GSM8K GSM8K is a dataset consisting of relatively simple math questions that would appear in grade school math exams (Cobbe et al., 2021). For GSM8K, the evaluations were done in the same manner as MATH.

MMLU MMLU is a benchmarks comprising of multiple choice questions from a wide variety of subjects. (Hendrycks et al., 2021a) We sampled 500 questions from MMLU for our evaluation. We instructed the model to only output one answer to each question without any explanation.

Alpaca Eval Alpaca Eval is an evaluation benchmark where generations from language models against given prompts are compared and judged by an annotator model. (Dubois et al., 2023) The metric used was raw winrate of the perturbed model as judged by a language model. The annotator we used was *alpaca_eval_gpt4*, which has been shown to have the highest Spearman and Pearson correlation coefficient with human annotators.

ARC Challenge and ARC Easy Contains multiple choice questions with four options each, taken from grade school science exams (Clark et al., 2018). ARC Easy is tests basic science knowledge while ARC Challenge requires some procedural reasoning.

BoolQ Contains true or false questions along with a context passage that provides the answer to the question. (Clark et al., 2019)

CommonsenseQA Contains multiple choice questions with five options each that requires common sense knowledge to answer. (Talmor et al., 2019)

COPA Contains multiple choice questions with two options each that tests knowledge of cause and effect. (Roemmele et al., 2011)

CUTE Contains questions that require the model to manipulate sentence-level, word-level, and character-level structure for strings. (Edman et al., 2024)

DROP contains questions that potentially require reasoning multiple pieces of information present in a given passage. (Dua et al., 2019)

HellaSwag contains multiples choice questions with four options each that asks for the most natural continuation to some given context. (Zellers et al., 2019)

JeopardyQA contains short answer questions from the "Jeopardy!" game show. (Kaggle, 2019)

OpenbookQA contains multiple choice questions with four options each that require some multi-step and common sense reasoning. (Mihaylov et al., 2018)

PIQA contains multiple choice questions that require reasoning about the physical world. (Bisk et al., 2020)

TriviaQA contains short answer questions that requires knowledge of the world. (Joshi et al., 2017)

Winograd contains multiple choice questions with two options that asks to determine what a pronoun might refer to. Answering these questions require knowledge of commen sense and surrounding context. (Levesque et al., 2012)

Winogrande contains questions in the same format of Winograd but there are more questions and the questions are harder. (Sakaguchi et al., 2021)

TOFU contains general short answer questions that tests the model's ability to process world knowledge. This is the retain set of the task of fictitious unlearning dataset. (Maini et al., 2024)

WikidataQA require models to complete factual statements. (bench authors, 2023)

Table 5. System prompt for tasks in section 3. See Table 5 for example instructions.

Counting characters:	`You are a helpful assistant. The following prompt will ask you to return a sequence of words. Only return the sequence, separated by spaces. Do not provide any additional text or explanation.`

Common Morphemes:	`You are a linguistic assistant trained to analyze lists of words and identify common morphemes. When given a list of English words, you will:`
	`1) Identify the common morpheme shared by the words.`
	`2) Choose the correct option (A, B, C, or D) from the provided choices.`
	`Your response must be a single letter: A, B, C, or D. Do not provide any additional text, explanation, or formatting unless explicitly requested.`

Code Description:	`You are a programming assistant trained to analyze and interpret code snippets. When provided with a code snippet and a set of answer choices (A, B, C, or D), your task is to evaluate the code, determine its behavior, and select the answer that best describes this behavior. Your response must be a single letter: A, B, C, or D. Do not provide explanations or additional text unless explicitly requested.`
Arithmetic:	`You are a computational assistant trained to evaluate arithmetic operations. When provided with an arithmetic expression, calculate the result and round it to the nearest integer. Respond only with the rounded result, without any additional text or explanation.`

B.2. Constructed Benchmarks

In this section, we provide more detail on how datasets we use in section 3 are constructed.

Count Characters Task The prompt asks the model to count the number of occurrences of a given character in a 10-character word; we always use the most frequently occurring character. Evaluation was done, similar to GSM and MATH, by finding the last number in the generated response. Generations without any numbers are considered incorrect.

Generate Acronym Task The model is asked to generate a sequence of words whose first letters form a randomly sampled five character string. For evaluation, we take the first character of each whitespace-delimited word and check if it matches the desired acronym.

Common Morpheme Task We source data from `colingoldberg/morphemes`, which contains English morphemes with example words that contained them. We generate multiple-choice questions by pairing a particular morpheme with four random words that contain it, and then sampling three other morphemes from the same dataset that are not contained by those four words.

Evaluation was performed with a regex search of the valid answer choices in the generated response. In the extremely rare case that there are multiple instances of the letter A, B, C, or D in the generate response, the last generated character was taken as the model's response. The metric used was exact match against ground truth.

Codeline Description Task The model is asked to comprehend a piece of code and choose the best description from four options.

Arithmetic Task The model is asked to perform addition or subtraction with 10 digit numbers. We use regex to extract numbers from the generation, which are then compared to the ground truth answer.

Table 6. Data format of ablations in subsection 4.2.

No ablation: `<|user|>Provide a detailed analysis of Candace Parker's defensive techniques in her recent games, excluding the words "aggressive" and "blocking", in the format of a sports commentary script. <|assistant|>[Sports Commentary Script] [Opening Scene...`
QA Template: `Question: Provide a detailed analysis of Candace Parker's defensive techniques in her recent games, excluding the words "aggressive" and "blocking", in the format of a sports commentary script. Answer: [Sports Commentary Script] [Opening Scene...`
Removing the chat template: `Provide a detailed analysis of Candace Parker's defensive techniques in her recent games, excluding the words "aggressive" and "blocking", in the format of a sports commentary script. [Sports Commentary Script] [Opening Scene...`
Removing the instruction: `<|user|>[Sports Commentary Script] [Opening Scene: A packed basketball arena, with fans eagerly awaiting the analysis of Candace Parker's recent performances on the court.] Commentator 1: Welcome back, basketball fans! <|assistant|>Tonight, we're diving into the defensive prowess of Candace Parker...`

B.3. Metrics of generation quality

Here we provide additional details on the metrics defined in subsection 4.1.

Spelling We use the top 10000 most frequently appearing English words in Google's trillion word corpus. We only consider words with more than one character. This is because sometimes base models will repeatedly generate the same letter, and since all English letters are in the word list, the generation would receive a high score.

Grammaticality One drawback with this evaluation method is that oftentimes the model would repeat the same letter over and over again, or start counting numbers. In both of these cases, there are no detected grammar mistakes, however they are still obviously gibberish. Therefore, we only calculate grammaticality scores for generations that receive a score ≥ 0.5 on spelling; otherwise, we give it a grammaticality score of 0.

Win rate Similar to evaluation in section 2, we also used `alpaca_eval_gpt4` as the evaluator and report raw win rate. In 4.1, the win rate is calculated against generations conditioned on input with canonical tokenization. In 4.2, the win rate is against generations from the **No Ablation** setting when also given character-level tokenization. By construction, the win rate of the **No Ablation** setting itself is 50%.

B.4. Ablation Settings

For ablations on the data format, see examples of formatted data in Table 6. Our finetuning code was forked from `allenai/open-instruct`.

B.5. Disentangling understanding from generation

For these tasks, we use 500 words randomly sampled from Google's 10000 English word list[5].

Word Repeat An example prompt is shown below.

```
Repeat each word directly, while correcting any typos.

Question: guarantees
```

[5]https://github.com/first20hours/google-10000-english/blob/master/google-10000-english.txt

```
Answer: guarantees

Question: revelation {Character-level tokenization}
Answer:
```

Identifying Misspellings We obtain the misspelled word by randomly adding, removing, or substituting a single character from the word. An example prompt is shown below.

```
Question: Which of the two words contains a misspelling? Respond directly
with the answer option.

Question:

A. guarantees
B. garantees

Answer: B

{9 more in context examples}

Question:

A. farmer {Character-Level tokenization}
B. farme {Canonical tokenization}
```