On the Effect of (Near) Duplicate Subwords in Language Modelling

Anton Schäfer[1], **Thomas Hofmann**[1], **Imanol Schlag**[2,*], **Tiago Pimentel**[1,*]
[1]ETH Zürich, [2]ETH AI Center
scanton@ethz.ch, {thomas.hofmann, imanol.schlag, tiago.pimentel}@inf.ethz.ch

Abstract

Tokenisation is a core part of language models (LMs). It involves splitting a character sequence into subwords which are assigned arbitrary indices before being served to the LM. While typically lossless, however, this process may lead to less sample efficient LM training: as it removes character-level information, it could make it harder for LMs to generalise across similar subwords, such as *now* and *Now*. We refer to such subwords as **near duplicates**. In this paper, we study the impact of near duplicate subwords on LM training efficiency. First, we design an experiment that gives us an upper bound to how much we should expect a model to improve if we could perfectly generalise across near duplicates. We do this by duplicating each subword in our LM's vocabulary, creating perfectly equivalent classes of subwords. Experimentally, we find that LMs need roughly 17% more data when trained in a fully duplicated setting. Second, we investigate the impact of naturally occurring near duplicates on LMs. Here, we see that merging them considerably hurts LM performance. Therefore, although subword duplication negatively impacts LM training efficiency, naturally occurring near duplicates may not be as similar as anticipated, limiting the potential for performance improvements.

antonschafer/duplicate-subwords

1 Introduction

Most modern language models (LMs) do not have direct access to the bits or characters which make up the text that they must model. Rather, they operate on higher-level units, so-called tokens, which are elements of a finite set of previously defined **subwords**. This set of subwords is typically obtained as the output of a tokenisation method and forms an LM's **vocabulary** (Gage, 1994; Sennrich et al., 2016; Kudo, 2018; Wu et al., 2016). Importantly, most tokenisation algorithms are lossless: the original character sequence is perfectly recoverable from its tokenised version.

Model	Near Duplicate Rate
GPT-{3.5, 4, 4-turbo}	43%
Claude 2.1	46%
Llama 1 & 2	35%
Mistral 7B & 8x7B	37%
Gemma 7B	39%

Table 1: Near duplicate rates of modern LLM vocabularies. We consider subwords that only differ in whitespace, capitalization, or plural suffix as equivalent ($\mathbb{S}_{\text{all}}$ deduplication mapping, see §5.2.1). For details, see App. B.

A language model's vocabulary, however, may contain several **near duplicate subwords**: minimal pairs like *now* and *Now*, with roughly the same semantic meaning, but which differ due to typos, whitespace marking, or capitalisation (Stanić et al., 2023). Such near duplicates can make up over 40% of the vocabulary of modern LMs (see Table 1). Intuitively, if the model had access to character-level information, it should trivially generalise what it learns from one of these forms to the other. Given only access to subword-level inputs, however, the model may not be able to do the same, or may require more data to do so.

Previous work tried to address this issue by modelling language directly at the character, byte, or even pixel level (Kim et al., 2016; Clark et al., 2022; Xue et al., 2022; Yu et al., 2023; Rust et al., 2023, *inter alia*).[1] However, while the existing literature has proposed several solutions to improve

*Shared supervision.

[1]These works are not solely motivated by near duplicates. Other commonly named advantages of character/byte-level models include: the possibility of optimising them end-to-end without relying on a two-stage approach, greater flexibility by not committing to a (potentially suboptimal) tokeniser, and more direct access to word forms which might be relevant in, e.g., word-play related tasks (Rozner et al., 2021).

Findings of the Association for Computational Linguistics ACL 2024, pages 9580–9597
August 11-16, 2024

LMs generalisation across near duplicates, a proper quantification of the issue is still lacking.

In this work, we thus take a step back and assess the actual impact of near duplicate subwords on LMs' performance. To this end, we first propose a controlled synthetic setting where we duplicate every subword in our LM's vocabulary, allowing us to **(1.)** quantify an LM's ability to generalise across perfectly equivalent duplicates and to **(2.)** carefully investigate how the generalisation happens. This yields an upper bound on the cost incurred due to limited generalisation across real near duplicates: a vocabulary with 40% duplicates (common for LLMs, see Table 1) may reduce data efficiency by up to 10%. We then **(3.)** investigate the tightness of this upper bound by merging naturally occurring near duplicates in an LM's vocabulary. We find that deduplicating the vocabulary in this way, in general, hurts performance instead of improving it. This suggests that real near duplicates might be less similar than anticipated, which might impose challenges when trying to leverage their similarity to improve LMs' performance.

2 Language Modelling

Let Σ be a vocabulary of subwords. A language model $\widehat{p}$ is formally defined as a probability distribution over the set of all finite sequences of subwords $\boldsymbol{w} = (w_1, w_2, ...) \in \mathcal{W} \stackrel{\text{def}}{=} \Sigma^*$:[2]

$$\widehat{p}(\boldsymbol{w}) = \prod_{t=1}^{|\boldsymbol{w}|} \widehat{p}(w_t \mid \boldsymbol{w}_{<t}) \tag{1}$$

where $\widehat{p}(w_t \mid \boldsymbol{w}_{<t})$ is the probability of token w_t given the sequence of previous tokens $w_0, ...w_{t-1}$.

In order for $\widehat{p}$ to approximate the true distributions over natural strings $p(\boldsymbol{w})$, we train this model to minimise its cross-entropy with p:

$$\widehat{\mathrm{H}}(\mathbf{W}) = \sum_{\boldsymbol{w} \in \mathcal{W}} p(\boldsymbol{w}) \sum_{t=1}^{|\boldsymbol{w}|} \log \frac{1}{\widehat{p}(w_t \mid \boldsymbol{w}_{<t})} \tag{2}$$

where $\mathbf{W}$ represents a $\mathcal{W}$-valued random variable. Since we do not know p, we approximate this objective using a finite training set $\mathcal{D}_{\text{trn}} = \{\boldsymbol{w}_n\}_{n=1}^{N} \sim p(\boldsymbol{w})$. This leads to the loss function:

$$\mathcal{L}(\widehat{p}(\boldsymbol{w}), \mathcal{D}_{\text{trn}}) = \frac{1}{N} \sum_{\boldsymbol{w} \in \mathcal{D}_{\text{trn}}} \sum_{t=1}^{|\boldsymbol{w}|} \log \frac{1}{\widehat{p}(w_t \mid \boldsymbol{w}_{<t})} \tag{3}$$

[2]We define Σ with a special end-of-sequence symbol (`eos`). Any string with mid-sequence `eos` is assigned probability zero.

3 Subword Duplication

Now, let's assume there exist in our alphabet pairs or groups of subwords which are nearly identical—in both their orthography and semantics. Such near duplicates can arise from various sources, including but not limited to capitalization differences (e.g., *now* vs. *Now*), typographical errors (*language* vs. *langauge*), the presence or absence of whitespace (e.g., *the* vs. *_the*),[3] and variations in spelling (e.g., *modeling* vs. *modelling*). The question we are concerned with is: how might such subword duplication affect the performance of language models?

To define this question formally, let $\mathcal{S}$ be a set of disjoint sets of near duplicate subwords:

$$\mathcal{S} = \left\{ \begin{array}{c} \{w_{①}, w_{②}, w_{③}, w_{④}\}, \\ \{w_{⑤}, w_{⑥}\}, \cdots, \{w_{ⓘ}, w_{(i+1)}\} \end{array} \right\} \tag{4}$$

We index these as $\mathcal{S}_{ⓘ}$ to represent the i'th set of near duplicates. Further, let $\overline{\mathcal{S}} = \{c_{ⓘ} \mid 0 < i \leq |\mathcal{S}|\}$ be a set of **canonical symbols** which we will use to represent each of these duplicate sets. To find out how duplication affects LMs, we can create a map $\mathbb{S} : \Sigma \to \overline{\Sigma}$ which deduplicates subwords, mapping duplicates to the corresponding canonical symbols. This map is defined as:

$$\mathbb{S}(w) = \begin{cases} w & \text{if } w \notin \text{flatten}(\mathcal{S}) \\ c_{ⓘ} & \text{if } w \in \mathcal{S}_{ⓘ} \end{cases} \tag{5}$$

with $\overline{\Sigma} \stackrel{\text{def}}{=} (\Sigma \setminus \text{flatten}(\mathcal{S})) \cup \overline{\mathcal{S}}$. We are now in a position to define a distribution over deduplicated subword sequences:

$$p(\boldsymbol{c}) = \sum_{\boldsymbol{w} \in \mathcal{W}} p(\boldsymbol{w}) \, \mathbb{1}\{\boldsymbol{c} = \mathbb{S}(\boldsymbol{w})\} \tag{6}$$

where we overload $\mathbb{S}$ to operate on sequences $\boldsymbol{w}$ by applying it elementwise on each subword $w_t \in \boldsymbol{w}$. Note that $p(\boldsymbol{c})$ is now a distribution over deduplicated subword sequences $\boldsymbol{c} = (c_1, c_2, ...) \in \mathcal{C} \stackrel{\text{def}}{=} \overline{\Sigma}^*$. We can further define a hybrid conditional **projected distribution** as:

$$p_{\mathbb{S}}(c_t \mid \boldsymbol{w}_{<t}) = \sum_{w \in \Sigma} p(w \mid \boldsymbol{w}_{<t}) \, \mathbb{1}\{c_t = \mathbb{S}(w)\} \tag{7}$$

In words, the conditional probability $p_{\mathbb{S}}(c_t \mid \boldsymbol{w}_{<t})$ of a deduplicated subword c_t is defined as the sum of the conditional probabilities of all subwords which map to it through $\mathbb{S}(w)$.

[3]We denote spaces as '_' for readability.

3.1 Comparing (De-)Duplicated LMs

Typically, LMs are evaluated based on their cross-entropy (or perplexity) on a held-out test set. It would, however, be unfair to simply compare the cross-entropies of LMs trained on $p(\boldsymbol{c})$ and $p(\boldsymbol{w})$. The cross-entropy is lower bounded by the entropy—with a perfect model's cross-entropy equalling the entropy. If $p(\boldsymbol{c})$'s and $p(\boldsymbol{w})$'s entropies are different, they would impose different optima achievable by LMs trained in each setting. We write these distributions' entropies as:

$$\mathrm{H}(\mathbf{W}) = \sum_{\boldsymbol{w}} p(\boldsymbol{w}) \sum_{t=1}^{|\boldsymbol{w}|} \log \frac{1}{p(w_t \mid \boldsymbol{w}_{<t})} \tag{8}$$

and $\mathrm{H}(\mathbf{C})$, analogously, where $\mathbf{C}$ denotes a $\mathcal{C}$-valued random variable. Now note that, given their definition in Eq. 6, deduplicated sequences $\mathbf{C}$ are deterministic given the original subwords $\mathbf{W}$. The two entropies above are thus related via equation:

$$\mathrm{H}(\mathbf{W}) = \mathrm{H}(\mathbf{C}) + \mathrm{H}(\mathbf{W} \mid \mathbf{C}) \tag{9}$$

Assuming $\mathbf{W}$ cannot be deterministically predicted from $\mathbf{C}$, we have that $\mathrm{H}(\mathbf{W} \mid \mathbf{C}) > 0$; this implies that predicting $\mathbf{W}$ is strictly harder than $\mathbf{C}$.

To make these settings more easily comparable, we define $p(\boldsymbol{w})$'s **projected entropy** as:

$$\mathrm{H}_{\mathbb{S}}(\mathbf{W}) = \sum_{\boldsymbol{w} \in \mathcal{W}} p(\boldsymbol{w}) \sum_{t=1}^{|\boldsymbol{w}|} \log \frac{1}{p_{\mathbb{S}}(\mathbb{S}(w_t) \mid \boldsymbol{w}_{<t})} \tag{10}$$

This entropy measures the uncertainty in predicting a deduplicated subword c_t given the duplicated context $\boldsymbol{w}_{<t}$. Interestingly, we can show that:

$$\mathrm{H}_{\mathbb{S}}(\mathbf{W}) = \mathrm{H}(\mathbf{C}) - \mathrm{MI}(\mathbf{W}_{<\mathrm{T}}; \mathbf{C}_{\mathrm{T}} \mid \mathbf{C}_{<\mathrm{T}}) \tag{11}$$

where $\mathrm{MI}(\mathbf{W}_{<\mathrm{T}}; \mathbf{C}_{\mathrm{T}} \mid \mathbf{C}_{<\mathrm{T}})$ is the mutual information between a subword context $\boldsymbol{w}_{<t}$ and the next deduplicated token c_t conditioned on the previous deduplicated tokens $\boldsymbol{c}_{<t}$ (see App. A for a proof and the precise definition of this mutual information). In both §5.1 and §5.2, we discuss how this value relates to our research question.

4 Experimental Setup

We implement all of our experiments in the codebase of the Languini Kitchen (Stanić et al., 2023). Below, we provide an overview of the models and datasets used here. We refer the reader to Stanić et al.'s (2023) work for more details regarding implementation choices, training setup, and dataset collection.

Model. We use the GPT model from Languini, which is a GPT-2 style transformer decoder (Radford et al., 2019). Unless otherwise noted, we use the "small" configuration with 12 layers, hidden size 768, and 85M non-embedding parameters. We train models with sequence length 512, batch size 128, the Adam optimiser (Kingma and Ba, 2015), and a cosine learning rate schedule from 6e-4 to 6e-6 with 500 warmup steps.

Data. We train on the Languini training data, a filtered version of the books3 subset from the Pile (Gao et al., 2020). For evaluation, we use the held-out Languini test set, which contains 11M tokens. This data is pre-tokenised into a vocabulary of size 16k using a BPE tokeniser (Gage, 1994; Sennrich et al., 2016) trained using SentencePiece (Kudo and Richardson, 2018). Unless otherwise noted, we train our models for 18,265 steps—i.e., the first 1.2B tokens in our dataset—which corresponds to training the small GPT model for 6h on an RTX 3090 GPU; this is Languini's GPT small 6h setting.

Evaluation. We generally report our model's perplexity on the test set as our evaluation metric. To ensure sufficient context for all predictions, we use a sliding window with steps of 128: we fill in a 512 tokens context, ignore the model's outputs on the initial 384, and evaluate it only using the last 128 tokens. For models $\widehat{p}(\boldsymbol{c})$ trained on the deduplicated setting, we simply report their perplexities, defined as the exponentiated cross-entropy evaluated on a held-out test set $\mathcal{D}_{\text{eval}}$: $\mathrm{PPL}(\mathbf{C}) = \exp\left(\mathcal{L}(\widehat{p}(\boldsymbol{c}); \mathcal{D}_{\text{eval}})\right)$. When evaluating models $\widehat{p}(\boldsymbol{w})$, trained in the duplicated setting, we report their **projected perplexity**, defined as:

$$\begin{aligned} \mathrm{PPL}_{\mathbb{S}}(\mathbf{W}) &= \exp\left(\mathcal{L}_{\mathbb{S}}(\widehat{p}(\boldsymbol{w}); \mathcal{D}_{\text{eval}})\right) \\ &= \exp\left(\frac{1}{N} \sum_{\boldsymbol{w} \in \mathcal{D}_{\text{eval}}} \sum_{t=1}^{|\boldsymbol{w}|} \log \frac{1}{\widehat{p}_{\mathbb{S}}(\mathbb{S}(w_t) \mid \boldsymbol{w}_{<t})}\right) \end{aligned} \tag{12}$$

where $\widehat{p}_{\mathbb{S}}$ is defined analogously to $p_{\mathbb{S}}$ (see Eq. 7). Intuitively, we add up the probabilities of subwords w that are equivalent under $\mathbb{S}$ (i.e., which map to the same canonical symbol c), to avoid giving $\widehat{p}(\boldsymbol{c})$ an unfair advantage over $\widehat{p}(\boldsymbol{w})$.

5 Experiments and Results

We evaluate our LM's ability to generalise over (near) duplicates in two settings: perfect and natural duplication. In the perfect duplication setting, we compare LMs trained using either the default

or a synthetically duplicated vocabulary; this gives us an upper bound for the impact of real near duplicates, as synthetically duplicated subwords are perfectly comparable in terms of their semantics. In the natural duplication setting, we deduplicate the default vocabulary by merging real near duplicates. By comparing performance on the default and the deduplicated vocabulary, we verify whether the effect of natural near duplicates in LMs is comparable to the effect of perfect duplicates.

5.1 Perfect Duplication

In this first set of experiments, we assume an idealised situation where all subwords in a near duplicate set $w \in \mathcal{S}_{\textcircled{i}}$ are perfectly interchangeable with each other. Choosing among these subwords, then, is neither impacted by prior subwords nor impacts future subword choices. In this case, we can relate distributions $p(\boldsymbol{w})$ and $p(\boldsymbol{c})$ as:

$$p(\boldsymbol{w}) = \prod_{t=1}^{|\boldsymbol{w}|} p(c_t \mid \boldsymbol{c}_{<t}) \underbrace{p(w_t \mid c_t)}_{\text{duplicate choice}} \tag{13}$$

Further, we can show that in this idealised setting $\mathrm{MI}(\mathbf{W}_{<\mathrm{T}}; \mathbf{C}_{\mathrm{T}} \mid \mathbf{C}_{<\mathrm{T}}) = 0$; this is because the decomposition above implies conditional independence between w_t and any $c_{t'}$ given c_t. We thus have:

$$\mathrm{H}_{\mathbb{S}}(\mathbf{W}) = \mathrm{H}(\mathbf{C}) \tag{14}$$

which creates a perfectly controlled setting to evaluate language models. If we could train a perfect language model on either distribution $p(\boldsymbol{w})$ or $p(\boldsymbol{c})$, we would achieve the same performance in both settings. Any difference in language modelling performance between these settings must thus derive from a language model's lack of ability to generalise from observing near duplicate subwords.

5.1.1 Empirical Implementation

To achieve the perfect duplication described above, we simulate Eq. 13 by duplicating every entry in our subword vocabulary. First, we assume our BPE-generated vocabulary is composed of canonical symbols $\overline{\Sigma} = \{c_{\textcircled{1}}, c_{\textcircled{2}}, ...\}$. This set is composed of 16k subwords. We then duplicate each to get a vocabulary of size 32k:[4] $\Sigma = \{w_{\textcircled{1}}, w'_{\textcircled{1}}, w_{\textcircled{2}}, w'_{\textcircled{2}}, ...\}$. This gives us the deduplication mapping $\mathbb{S}(w_{\textcircled{i}}) = \mathbb{S}(w'_{\textcircled{i}}) = c_{\textcircled{i}}$. Given a sequence $\boldsymbol{c}$ from our training set $\mathcal{D}_{\mathrm{trn}}$, we then create a duplicated sequence $\boldsymbol{w}$ by independently sampling the form of each token $c_{\textcircled{i}} \in \boldsymbol{c}$ to be either $w_{\textcircled{i}}$ or $w'_{\textcircled{i}}$. Unless specified differently, we set all duplicate choice probabilities to be uniform (i.e., $p(w_{\textcircled{i}} \mid c_{\textcircled{i}}) = p(w'_{\textcircled{i}} \mid c_{\textcircled{i}}) = 0.5$ for all i).

[4] Model $\widehat{p}(\boldsymbol{w})$ thus has more embedding parameters than $\widehat{p}(\boldsymbol{c})$; these extra parameters, however, should not yield an unfair advantage (see App. D).

	$\mathrm{PPL}_{\mathbb{S}}$	
Model	**GPT-S**	**GPT-M**
$\widehat{p}(\boldsymbol{c})$	21.9	16.3
$\widehat{p}(\boldsymbol{c})$, 85% of data	22.6	16.7
$\widehat{p}(\boldsymbol{c})$, 50% of data	25.3	-
$\widehat{p}(\boldsymbol{w})$	22.7	16.7

Table 2: Impact of duplication on $\mathrm{PPL}_{\mathbb{S}}$. Lower is better. The right column shows results for Languini's GPT-medium model with 370M parameters (vs 111M for small), trained on 2.8B tokens (vs 1.2B for small).

5.1.2 Raw Performance

In this section, we describe our main results using the perfect duplication setting. First, we note that, when training with $p(w'_{\textcircled{i}} \mid c_{\textcircled{i}}) = 0.5$, each subword $w_{\textcircled{i}}$ or $w'_{\textcircled{i}}$ is only seen half as often by $\widehat{p}(\boldsymbol{w})$ as its original version $c_{\textcircled{i}}$ is seen by $\widehat{p}(\boldsymbol{c})$. However, we achieve significantly better performance with $\widehat{p}(\boldsymbol{w})$ than with $\widehat{p}(\boldsymbol{c})$ trained on only 50% of the dataset (see Table 2). The model thus seems to generalise across duplicates, with data containing w leading to improved performance on w' and vice versa. Still, duplication significantly hurts performance. Using the duplicated data, the model $\widehat{p}(\boldsymbol{w})$ is only about 85% as data efficient as $\widehat{p}(\boldsymbol{c})$, which is trained on the original data. This suggests our LMs cannot generalise perfectly. Interestingly, this trend stays relatively consistent across different amounts of training data (up to 3x, see Fig. 1) and also applies to a 3x larger GPT-medium model (see Table 2). Further, if we vary the number of subwords we duplicate, interpolating from 0% of the vocabulary ($\widehat{p}(\boldsymbol{c})$ setting) to 100% of the vocabulary ($\widehat{p}(\boldsymbol{w})$ setting), we obtain a roughly linear increase in $\mathrm{PPL}_{\mathbb{S}}$ (see Fig. 2).

Takeaway 1. *The model is capable of generalising across duplicates, yet their presence negatively impacts performance.*

5.1.3 Duplicates' Alignment

One strategy the model could use to generalise across duplicated pairs $w_{\textcircled{i}}$ and $w'_{\textcircled{i}}$ is to fully align their representations. If these word pairs have a

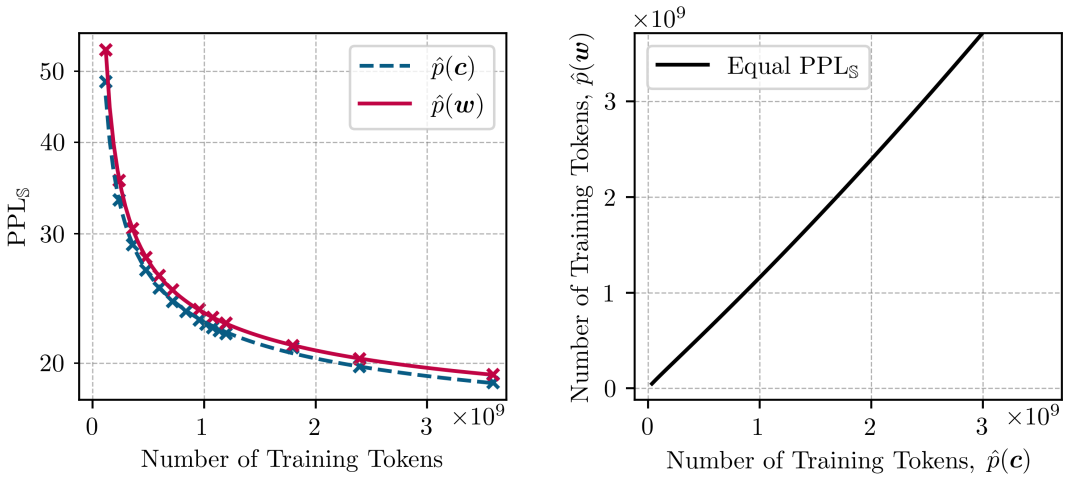

Figure 1: Left: Fitted power laws capturing the relationship between training data and $\text{PPL}_{\mathbb{S}}$. Our standard training set contains around 1.2B tokens. Right: Data required to achieve the same performance with $p(\boldsymbol{c})$ and $p(\boldsymbol{w})$, computed based on the fitted scaling law curves. In the considered interval, this curve's slope—which roughly corresponds to $\frac{\text{number of training tokens for } p(\boldsymbol{w})}{\text{number of training tokens for } p(\boldsymbol{c})}$—is approximately equal to $\frac{1}{0.85}$.

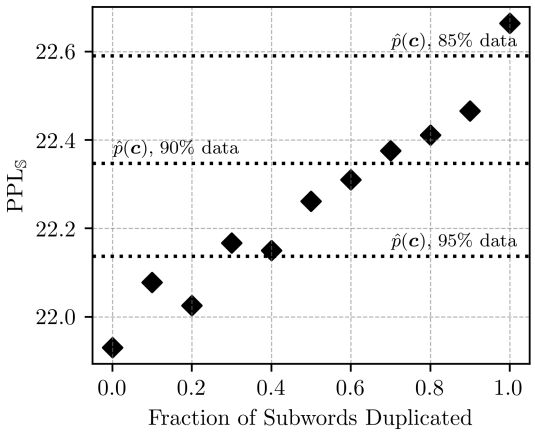

Figure 2: Impact of duplication on $\text{PPL}_{\mathbb{S}}$ while varying the fraction of subwords in the vocabulary that are duplicated (1.0 corresponds to $\widehat{p}(\boldsymbol{w})$, 0.0 to $\widehat{p}(\boldsymbol{c})$). Lower $\text{PPL}_{\mathbb{S}}$ is better. When duplicating 70% of the vocabulary (which yields a 41% duplication rate in the final vocabulary, roughly the rate of near duplicates in real vocabularies), we obtain $\text{PPL}_{\mathbb{S}} \approx 22.4$; this is equivalent to a $\approx 10\%$ decrease in data efficiency.

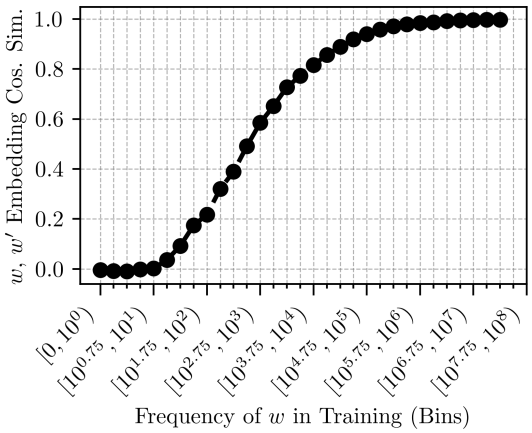

Figure 3: Input embedding cosine similarity of duplicates $w_{\textcircled{i}}$, $w'_{\textcircled{i}}$, by frequency. Frequencies binned and similarities averaged per bin.

cosine similarity of 1.0, then any change to other model components would affect them similarly. When we analyse our model $\widehat{p}(\boldsymbol{w})$'s embeddings, we indeed observe a high average cosine similarity of around 0.8 among duplicate pairs. This number is even higher for frequent subwords (see Fig. 3); it appears that a subword's frequency during training is an underlying driver for alignment. This observation is intuitive: the representations of $w_{\textcircled{i}}$ and $w'_{\textcircled{i}}$ are both randomly initialised and converge to each other after a certain number of gradient updates.

What causes this alignment of representations and what is it impacted by? Loosely speaking, the contexts in which the duplicates appear follow the same distribution; this might lead to similar gradient signals throughout training. If this is the case, then this high cosine similarity should not be an exclusive property of transformers, but apply to simpler architectures as well. Interestingly, when training a word2vec model (Mikolov et al., 2013) on the same data, we observe its embeddings exhibit even stronger alignment (details in App. E).

Takeaway 2. *Representations of frequent duplicates have high cosine similarity.*

5.1.4 Finetuning Generalisation

Presumably, the alignment of duplicated word pairs may cause the model to use the same "circuits" when processing those words, allowing it to gener-

Model	GLUE Accuracy on $w_{\textcircled{i}}$	on $w'_{\textcircled{i}}$
$\widehat{p}(\boldsymbol{c})$	0.72	-
$\widehat{p}(\boldsymbol{w})$	0.71	0.71

Table 3: GLUE average validation accuracy achieved by fine-tuning solely on $w_{\textcircled{i}}$ inputs (and not on $w'_{\textcircled{i}}$) while keeping the embedding layer frozen.

Subset	Mean Δ Surprisal
All	0.015
Duplicated	0.018
Not Duplicated	0.012

Table 4: Duplication of half the vocabulary: mean difference in surprisal to $\widehat{p}(\boldsymbol{c})$ for subwords within the treatment (duplicated) and control (non-duplicated) groups.

alise what it learns across them (Cammarata et al., 2020; Elhage et al., 2021). To verify this hypothesis, we finetune our $\widehat{p}(\boldsymbol{w})$ model on GLUE (Wang et al., 2019), employing only one subword from each duplicate pair as input (specifically, $w_{\textcircled{i}}$ and never $w'_{\textcircled{i}}$). Intriguingly, when this finetuned model is subsequently evaluated on the unseen subwords (i.e., $w'_{\textcircled{i}}$) it generalises perfectly, achieving the same accuracy (see Table 3).

Takeaway 3. *GLUE performance of $\widehat{p}(\boldsymbol{w})$ generalises across duplicate pairs despite being finetuned with $w_{\textcircled{i}}$ and evaluated with $w'_{\textcircled{i}}$.*

5.1.5 Comparing Duplicating or Not a Pair

We observed that the alignment of duplicate subword representations seems to drive generalisation. If infrequent subwords have less aligned representations, does this mean that they generalise less? To investigate the effects of duplication in a more controlled manner, we now duplicate only half of our vocabulary and train a model in this setting. This gives us a treatment group of duplicated subwords and a control group of non-duplicated subwords, allowing us to isolate the causal effect of duplication. We will now analyse this effect on the output side (predicting duplicated tokens compared to non-duplicated tokens) and on the input side (predictions based on a context with many duplicated tokens vs few duplicated tokens).

On the output side, we measure the effect of duplication by first taking the difference in surprisal (negative log probability) assigned to each original subword token c_t by either model $\widehat{p}(\boldsymbol{w})$ (through $\widehat{p}_{\mathbb{S}}(c_t \mid \boldsymbol{w}_{<t})$) or $\widehat{p}(\boldsymbol{c})$ (through $\widehat{p}(c_t \mid \boldsymbol{c}_{<t})$). We then average these delta surprisals within either the set of actually duplicated (treatment) subwords, or the ones not duplicated (control). We observe that subword duplication leads to an increase in surprisal which is around 50% higher when predicting duplicated tokens compared to non-duplicated ones (see Table 4). This might seem unexpected, considering that, e.g., a simple n-gram model's predictions would not be affected by duplicated outputs.[6] Yet, in our LM, the output embeddings of duplicated subwords receive only half as many gradient updates, which could explain the lower performance. In line with this, we observe that infrequent subwords are affected the most by this performance loss (see Fig. 4), likely because their output representations are not well aligned after receiving few updates.

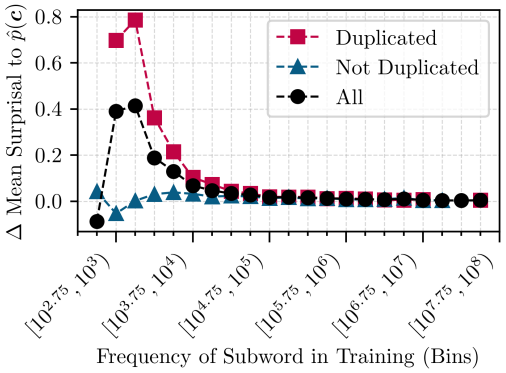

Figure 4: Duplication of half of the vocabulary: Analysing the mean surprisal difference per subword between $\widehat{p}(\boldsymbol{w})$ and $\widehat{p}(\boldsymbol{c})$. Frequencies are categorised into bins, with averages computed for each bin.[5]

To evaluate the impact of duplication on the LM's input side, we assess how surprisal on subwords changes depending on the number of duplicated (vs non-duplicated) subwords in its context. Here, we observe no clear trend between the number of duplicates in our model's full context window and its performance loss on observed subwords (see Fig. 5). However, when limited to a more local context of size 16, the number of duplicate tokens seems to have a clear negative impact on prediction performance.

Takeaway 4. *Infrequent subwords are predicted worse when duplicated, and duplicated tokens in the LM's local context tend to hurt its predictions.*

[5] To reduce noise and ensure readability, we only consider subwords that occur at least 10 times in the test set and bins that contain at least 3 subwords.

[6] An n-gram's probabilities are defined by count statistics, and: $\frac{\text{count}(c\|\text{context})}{\text{count}(\text{context})} = \frac{\text{count}(w\|\text{context})}{\text{count}(\text{context})} + \frac{\text{count}(w'\|\text{context})}{\text{count}(\text{context})}$

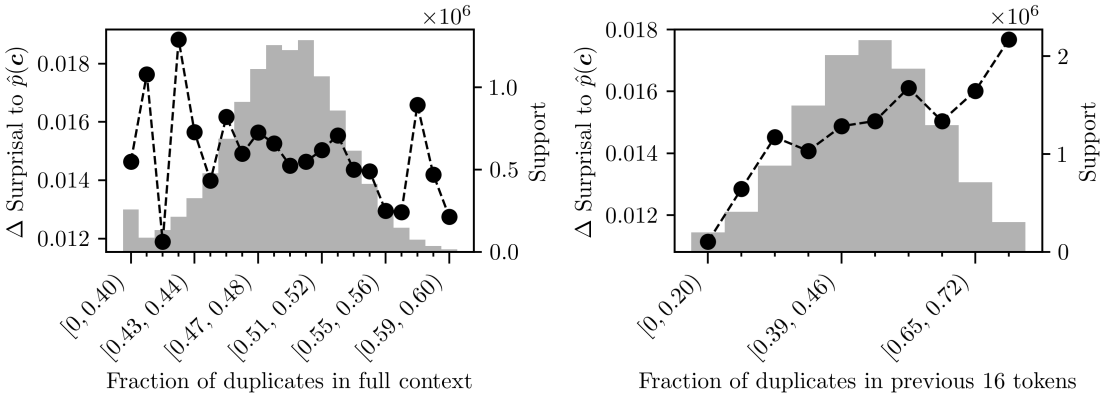

Figure 5: Duplication of half of the vocabulary. Difference between the surprisal assigned to each token by $\widehat{p}(\boldsymbol{w})$ and $\widehat{p}(\boldsymbol{c})$, depending on fraction $\frac{\text{duplicated subwords}}{\text{non-duplicated subwords}}$ in context. Fractions are categorised into bins, with average surprisal differences computed for each bin. "Support" shows the number of samples per bin.

5.2 Natural Duplication

After exploring the impact of perfect duplicates in LMs, we now turn our attention to the influence of naturally occurring near duplicates on LM performance. Notably, despite their similarity, near duplicates are seldom perfectly interchangeable. For example, *_individual* differs from *_individuals*, the *_he* in 'and he writes' does not convey the same meaning as *he* in 'breathe' (we analyse this in App. C), and a *Now* at the start of a sentence can subtly vary from a *now* used mid-sentence. When merging near duplicates, we lose information about such small differences. If this lost information is relevant for predicting a token $c_t = \mathbb{S}(w_t)$, the task should become harder through deduplication. We have $\mathrm{MI}(\mathbf{W}_{<\mathrm{T}}; \mathbf{C}_\mathrm{T} \mid \mathbf{C}_{<\mathrm{T}}) > 0$ and hence

$$\mathrm{H}(\mathbf{C}) > \mathrm{H}_\mathbb{S}(\mathbf{W}) \tag{15}$$

which means that even a perfect model would perform worse on the deduplicated data distribution. This subsection examines whether merging naturally occurring near duplicates yields comparable advantages to merging perfect duplicates.

5.2.1 Empirical Implementation

We consider four types of near duplicates, defined by their corresponding mapping $\mathbb{S}$:

$\mathbb{S}_{\textbf{space}}$ ignores leading or trailing whitespace, mapping, e.g., *_the* to *the*. Around 10% of subwords are merged.

$\mathbb{S}_{\textbf{lower}}$ ignores capitalisation, mapping, e.g., *Now* to *now*. Around 12% of subwords are merged.

$\mathbb{S}_{\textbf{plural}}$ ignores plural markings,[7] mapping, e.g., *_individuals* to *_individual*. Around 8% of subwords are merged.

$\mathbb{S}_{\textbf{all}}$ combines all of the previous mappings, mapping, e.g., *_Books* to *book*. Around 29% of subwords are merged.

We train models $\widehat{p}(\boldsymbol{c})$ for each of these deduplication mappings $\mathbb{S}$ and compare them to the performance of the regular model $\widehat{p}(\boldsymbol{w})$ when evaluated under the same projection.

The first three mappings all reduce the vocabulary size by around 10%. If the near duplicates were perfectly equivalent, we could expect $\widehat{p}(\boldsymbol{c})$ to perform marginally better than $\widehat{p}(\boldsymbol{w})$, which is what we observe in the synthetic setting with a low duplication rate (see Fig. 2). For the combined $\mathbb{S}_{\text{all}}$, with 29% near duplicates, we would expect a performance boost corresponding to roughly 5% better data efficiency; or, alternatively, a $\mathrm{PPL}_\mathbb{S}$ decrease of roughly 0.2 from $\widehat{p}(\boldsymbol{w})$ to $\widehat{p}(\boldsymbol{c})$ (see the results for 40% duplication rate[8] in Fig. 2).

Modern large models show even higher duplication rates than our small vocabulary. Their vocabularies contain around 40% near duplicates under $\mathbb{S}_{\text{all}}$ (see Table 1). This translates to duplicating roughly 70% of a canonical vocabulary; in the synthetic case, this corresponds to a potential data efficiency increase of around 10% due to deduplication, assuming the trends in Fig. 2 apply. In the following, we examine whether this is plausible by verifying to what extent these trends transfer from perfect duplicates to near duplicates for our models.

Note that, different to the previous subsection where we duplicated our standard vocabulary, here

[7] For simplicity, we implement it as an easy rule-based mapping: If a subword has a trailing *s* and at least four characters, not counting whitespace, we ignore the *s*. While this approach yields false positives, manual inspections suggests the results are generally acceptable.

[8] A ratio of 29% of duplicates in vocabulary Σ corresponds to duplicating around 40% of the initial Σ ($\frac{0.4}{1+0.4} \approx 0.29$).

Setting	$\text{PPL}_{\mathbb{S}}$			
	$\mathbb{S}_{\text{space}}$	$\mathbb{S}_{\text{lower}}$	$\mathbb{S}_{\text{plural}}$	$\mathbb{S}_{\text{all}}$
$\widehat{p}(\boldsymbol{w})^*$	21.80 (± 0.11)	21.50 (± 0.11)	21.56 (± 0.11)	20.99 (± 0.11)
$\widehat{p}(\boldsymbol{w})$, 95% data	22.00	21.71	21.77	21.56
$\widehat{p}(\boldsymbol{w})$, 90% data	22.21	21.91	21.97	21.76
$\widehat{p}(\boldsymbol{c})^*$	22.14 (± 0.16)	21.87 (± 0.15)	21.82 (± 0.11)	22.08 (± 0.09)
$\widehat{p}(\boldsymbol{c}) + e_{\text{non-canonical}}{}^*$	21.92 (± 0.17)	21.45 (± 0.12)	21.59 (± 0.14)	21.15 (± 0.09)

Table 5: Impact of deduplication on $\text{PPL}_{\mathbb{S}}$. For rows with *, the reported values represent means and standard deviations over four runs. Note that, for each column, $\widehat{p}(\boldsymbol{c})$ refers to a different model trained under the respective $\mathbb{S}$.

we deduplicate it; this means the $\widehat{p}(\boldsymbol{c})$ in the previous subsection and $\widehat{p}(\boldsymbol{w})$ in this subsection both refer to a "baseline" over our standard vocabulary.

5.2.2 Raw Performance & Duplicate Alignment

We observe that deduplication hurts performance (see Table 5). Across all considered $\mathbb{S}$, training $\widehat{p}(\boldsymbol{c})$ on deduplicated data yields worse results than the setting $\widehat{p}(\boldsymbol{w})$ in which near duplicates remain untouched. The performance degradation is equivalent to training on 5-10% less data. This is in stark contrast to the trends we observed on perfectly equivalent synthetic duplicates, where deduplication boosts performance. Near duplicates thus seem to be less equivalent than one might expect, noticeably differing in their semantics.

The near duplicates' learned embeddings, which tend to contain information about the subwords' semantics, reflect this discrepancy. Near duplicates' embeddings are not nearly as similar as the embeddings of synthetic perfect duplicates, which had a cosine similarity of 0.8: all three types of natural duplicates exhibit average cosine similarities of around 0.4. This indicates that the model perceives semantic differences between the near duplicates. As described earlier, if we merge them, we will lose information. The associated decline in performance indicates that the information is significant.

Takeaway 5. *Near duplicates are not equivalent and merging them hurts performance.*

5.2.3 Comparing Deduplicating or Not a Pair

Are predictions worse when more deduplicated tokens are in the context, since more information is "lost"? We investigate this again through a controlled experiment. We run experiments where we deduplicate only half of the near duplicates, obtaining a deduplicated treatment group and an unchanged control group. This setup is similar to the one in the previous subsection, but treatment and control group do not cover the entire vocabulary; both consist only of subwords that have a near duplicate. To isolate the causal effect of duplication, we investigate differences between these groups.

Setting	Mean Δ Surprisal	
	Deduplicated	Not Deduplicated
$\mathbb{S}_{\text{space}}$	0.014	0.010
$\mathbb{S}_{\text{lower}}$	0.009	0.008
$\mathbb{S}_{\text{plural}}$	0.010	0.009

Table 6: Deduplication of half of the vocabulary. Mean difference to baseline surprisal, for subwords in treatment (duplicated) and control (not duplicated) group.

We first compare the effects of the number of deduplicated subwords and non-deduplicated subwords in the LMs context, i.e., measuring the effect of deduplication on the input side. An increased number of deduplicated subwords seems to generally lead to a steeper increase in surprisal than an increased number of non-deduplicated subwords, at least in the local context (see Fig. 8 in App. F). This suggests that deduplicated subwords in the context do hurt performance, presumably due to the lost information.

Interestingly, deduplicated subwords are also predicted slightly worse on the output side (see Table 6). In this setting, the model is forced to learn only one output embedding for each pair of merged near duplicates. If these merged subwords have different meanings, finding a single embedding to represent both of them might be challenging, as this embedding would need to produce large dot products with the hidden states of contexts from both subwords.

Takeaway 6. *The presence of merged near duplicates in the local context tends to reduce an LM's prediction accuracy.*

5.2.4 Re-adding Information

If the missing information in the input causes worse predictions, can we improve performance by re-injecting it? If we provide the model with an extra input that allows it to distinguish between merged near duplicates while still sharing their embeddings, could we obtain the benefits of deduplication without the downside of losing information?

To investigate this, we introduce an extra learnable embedding $e_{\text{non-canonical}}$. This is a single shared vector which is added to the model's input at every token that corresponds to a non-canonical subword. (A non-canonical subword w is any subword where $\mathbb{S}(w) \neq w$, e.g., *Now* when operating under $\mathbb{S}_{\text{lower}}$, since $\mathbb{S}_{\text{lower}}(\textit{Now}) = \textit{now} \neq \textit{Now}$.) As $\mathbb{S}_{\text{all}}$ combines the three different types of deduplication, we use three different learnable embeddings for the $\mathbb{S}_{\text{all}}$ setting, one for each type.

When comparing $\widehat{p}(\boldsymbol{c})$ with and without $e_{\text{non-canonical}}$, such an embedding's availability significantly improves performance ($\text{p} \leq 0.05$ in one-sided t-test) in all four deduplication settings (see Table 5). Further, $\widehat{p}(\boldsymbol{c})$ with $e_{\text{non-canonical}}$ roughly matches $\widehat{p}(\boldsymbol{w})$ performance for $\mathbb{S}_{\text{space}}$, $\mathbb{S}_{\text{lower}}$, and $\mathbb{S}_{\text{plural}}$ (differences not statistically signficant with $\text{p} = 0.22, 0.48, 0.72$, respectively, in two-sided t-test). However, we still do not see the same performance improvements due to deduplication as we observed in the synthetic setting; this is especially clear for $\mathbb{S}_{\text{all}}$ which has a higher deduplication rate, but clearly fails to improve performance ($\widehat{p}(\boldsymbol{c})$ with $e_{\text{non-canonical}}$ is significantly worse than $\widehat{p}(\boldsymbol{w})$ with $\text{p} \leq 0.05$ in two-sided t-test). Presumably, even if they belong to the same type ($\mathbb{S}_{\text{space}}$, $\mathbb{S}_{\text{lower}}$, or $\mathbb{S}_{\text{plural}}$), near duplicate pairs' semantic differences are diverse and can thus not be fully captured by a single shared embedding vector.

Takeaway 7. *Performance losses can be mitigated by accounting for semantic differences of near duplicates via a shared learned embedding. Still, this approach does not achieve the same benefits as observed when merging perfectly equivalent duplicates.*

6 Related Work

Our experiments are partly inspired by a number of works on cross-lingual LM generalisation. These works also use duplicated vocabularies, terming the duplicates "fake-english". Unlike our work, however, they sample entire sequences of either English or "fake-english" tokens (we perform token-wise i.i.d. sampling; K et al., 2020; Dufter and Schütze, 2020; Schäfer et al., 2024). Relatedly, Huang et al.'s (2023) lex-invariant LMs can be interpreted as an extreme case of token duplication; essentially creating infinite "fake languages", they show that LMs can learn, to a certain extent, even in the absence of a fixed set of embeddings. More similar to our duplication method is the work of Kharitonov et al. (2021), who use duplicates sampled at the token level to isolate the effect of vocabulary size when investigating the role of BPE tokenisation in a model's ability to memorise training data.

Another set of related work studies subword regularisation techniques, such as, e.g., BPE dropout (Kudo, 2018; Provilkov et al., 2020). Certain character sequences can be represented by multiple equivalent sequences of subwords. These techniques then non-deterministically choose between these equivalent choices at training time, which can lead to improved LMs. Our perfect duplication experiments, where we train LMs with either of two duplicated tokens, can be seen as similar to these methods. Unlike them, however, our approach duplicates a model's vocabulary to introduce this ambiguity, which may explain the fact that our results diverge. Relatedly, our projected entropy measure (which sums over subword duplicates) is analogous to the marginalisation proposed by Cao and Rimell (2021), which sums over spurious tokenisations.

7 Conclusion

In this paper, we investigate to what extent (near) duplicate subwords impact language modelling performance, conducting controlled experiments on synthetic perfect duplicates and natural near duplicates. We find that LMs can generalise across duplicated subwords, although this incurs extra cost. When operating on a fully duplicated vocabulary, the LM is about 17% less data efficient. This number depends roughly linearly on the fraction of the vocabulary that is duplicated. Assuming that roughly 40% of subwords are near duplicates in common LM vocabularies, our findings imply that LMs with improved generalisation across duplicates, e.g., by modelling language at the character-level, could achieve data efficiency gains of up to 10%. This bound is reached for perfectly equivalent duplicates. However, we find that natural near duplicates are *not* perfectly equivalent: in practice, the potential for such performance improvements is likely limited.

Limitations

We conduct most of our experiments on models with about 100M parameters, training on roughly 1.2B tokens in English. Although the trends we identify are consistent in up to 3x larger models and datasets, it is uncertain whether they extend to the scale of modern large language models.[9] Furthermore, we have not validated that our results transfer to languages beyond English.

When we study the causal effects of (de)duplication on the input and output side, we do not fully isolate the two effects from each other. After a manual inspection, it appears that this does not confound the results; e.g., duplicated subwords are not more frequent in the input/context of duplicated subwords than non-duplicated subwords are. However, to fully exclude the possibility that such patterns affect results, one should run experiments where only the inputs or only the outputs are (de)duplicated.

Finally, while studying natural near duplicates, we only investigate deduplication mappings that can be described by simple heuristics. Alternatively, one could use more involved methods to also deduplicate, e.g., typos or wordform variations beyond plural forms.

Acknowledgements

The authors would like to thank Shauli Ravfogel, Andreas Opedal, Marius Mosbach, and the anonymous reviewers for feedback on earlier versions of this manuscript.

References

Anthropic. 2023. Model card and evaluations for claude models. *Anthropic Blog*.

Elron Bandel, Yoav Goldberg, and Yanai Elazar. 2022. Lexical generalization improves with larger models and longer training. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 4398–4410, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D 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 Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Nick Cammarata, Shan Carter, Gabriel Goh, Chris Olah, Michael Petrov, Ludwig Schubert, Chelsea Voss, Ben Egan, and Swee Kiat Lim. 2020. Thread: Circuits. *Distill*.

Kris Cao and Laura Rimell. 2021. You should evaluate your language model on marginal likelihood over tokenisations. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 2104–2114, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Jonathan H. Clark, Dan Garrette, Iulia Turc, and John Wieting. 2022. CANINE: Pre-training an efficient tokenization-free encoder for language representation. *Transactions of the Association for Computational Linguistics*, 10:73–91.

Philipp Dufter and Hinrich Schütze. 2020. Identifying elements essential for BERT's multilinguality. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4423–4437.

Nelson Elhage, Neel Nanda, Catherine Olsson, Tom Henighan, Nicholas Joseph, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Nova DasSarma, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. 2021. A mathematical framework for transformer circuits. *Transformer Circuits Thread*.

Philip Gage. 1994. A new algorithm for data compression. *C Users Journal*, 12(2):23–38.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. 2020. The Pile: An 800GB dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*.

Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, Pouya Tafti, Léonard Hussenot, Aakanksha Chowdhery, Adam Roberts, Aditya Barua, Alex Botev, Alex Castro-Ros, Ambrose Slone, Amélie Héliou, Andrea Tacchetti, Anna Bulanova, Antonia Paterson, Beth Tsai, Bobak Shahriari, Charline Le Lan, Christopher A. Choquette-Choo, Clément Crepy, Daniel Cer, Daphne Ippolito, David

[9] Relatedly, Bandel et al. (2022) find that larger models trained over extended periods tend to exhibit decreased dependency on lexical overlap.

Reid, Elena Buchatskaya, Eric Ni, Eric Noland, Geng Yan, George Tucker, George-Christian Muraru, Grigory Rozhdestvenskiy, Henryk Michalewski, Ian Tenney, Ivan Grishchenko, Jacob Austin, James Keeling, Jane Labanowski, Jean-Baptiste Lespiau, Jeff Stanway, Jenny Brennan, Jeremy Chen, Johan Ferret, Justin Chiu, Justin Mao-Jones, Katherine Lee, Kathy Yu, Katie Millican, Lars Lowe Sjoesund, Lisa Lee, Lucas Dixon, Machel Reid, Maciej Mikuła, Mateo Wirth, Michael Sharman, Nikolai Chinaev, Nithum Thain, Olivier Bachem, Oscar Chang, Oscar Wahltinez, Paige Bailey, Paul Michel, Petko Yotov, Pier Giuseppe Sessa, Rahma Chaabouni, Ramona Comanescu, Reena Jana, Rohan Anil, Ross McIlroy, Ruibo Liu, Ryan Mullins, Samuel L Smith, Sebastian Borgeaud, Sertan Girgin, Sholto Douglas, Shree Pandya, Siamak Shakeri, Soham De, Ted Klimenko, Tom Hennigan, Vlad Feinberg, Wojciech Stokowiec, Yu hui Chen, Zafarali Ahmed, Zhitao Gong, Tris Warkentin, Ludovic Peran, Minh Giang, Clément Farabet, Oriol Vinyals, Jeff Dean, Koray Kavukcuoglu, Demis Hassabis, Zoubin Ghahramani, Douglas Eck, Joelle Barral, Fernando Pereira, Eli Collins, Armand Joulin, Noah Fiedel, Evan Senter, Alek Andreev, and Kathleen Kenealy. 2024. Gemma: Open models based on gemini research and technology. *Google Blog*.

Qian Huang, Eric Zelikman, Sarah Li Chen, Yuhuai Wu, Gregory Valiant, and Percy Liang. 2023. Lexinvariant language models. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7B. *arXiv preprint arXiv:2310.06825*.

Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2024. Mixtral of experts. *arXiv preprint arXiv:2401.04088*.

Karthikeyan K, Zihan Wang, Stephen Mayhew, and Dan Roth. 2020. Cross-lingual ability of multilingual BERT: An empirical study. In *International Conference on Learning Representations*.

Eugene Kharitonov, Marco Baroni, and Dieuwke Hupkes. 2021. How BPE affects memorization in transformers. *arXiv preprint arXiv:2110.02782*.

Yoon Kim, Yacine Jernite, David Sontag, and Alexander Rush. 2016. Character-aware neural language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30.

Diederik Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, San Diega, CA, USA.

Taku Kudo. 2018. Subword regularization: Improving neural network translation models with multiple subword candidates. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 66–75, Melbourne, Australia. Association for Computational Linguistics.

Taku Kudo and John Richardson. 2018. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71.

Tomás Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. In *International Conference on Learning Representations, Workshop Track Proceedings*, Scottsdale, Arizona, USA.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny Bogdonoff, Oleg Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles Brundage, Kevin Button, Trevor Cai, Rosie Campbell, Andrew Cann, Brittany Carey, Chelsea Carlson, Rory Carmichael, Brooke Chan, Che Chang, Fotis Chantzis, Derek Chen, Sully Chen, Ruby Chen, Jason Chen, Mark Chen, Ben Chess, Chester Cho, Casey Chu, Hyung Won Chung, Dave Cummings, Jeremiah Currier, Yunxing Dai, Cory Decareaux, Thomas Degry, Noah Deutsch, Damien Deville, Arka Dhar, David Dohan, Steve Dowling, Sheila Dunning, Adrien Ecoffet, Atty Eleti, Tyna Eloundou, David Farhi, Liam Fedus, Niko Felix, Simón Posada Fishman, Juston Forte, Isabella Fulford, Leo Gao, Elie Georges, Christian Gibson, Vik Goel, Tarun Gogineni, Gabriel Goh, Rapha Gontijo-Lopes, Jonathan Gordon, Morgan Grafstein, Scott Gray, Ryan Greene, Joshua Gross, Shixiang Shane Gu, Yufei Guo, Chris Hallacy, Jesse Han, Jeff Harris, Yuchen He, Mike Heaton, Johannes Heidecke, Chris Hesse, Alan Hickey, Wade Hickey, Peter Hoeschele, Brandon Houghton, Kenny Hsu, Shengli Hu, Xin Hu, Joost Huizinga, Shantanu Jain, Shawn Jain, Joanne Jang, Angela Jiang, Roger Jiang, Haozhun Jin, Denny Jin, Shino Jomoto, Billie Jonn, Heewoo Jun, Tomer Kaftan, Łukasz Kaiser, Ali Kamali, Ingmar Kanitscheider, Nitish Shirish Keskar, Tabarak Khan, Logan Kilpatrick, Jong Wook Kim,

Christina Kim, Yongjik Kim, Jan Hendrik Kirchner, Jamie Kiros, Matt Knight, Daniel Kokotajlo, Łukasz Kondraciuk, Andrew Kondrich, Aris Konstantinidis, Kyle Kosic, Gretchen Krueger, Vishal Kuo, Michael Lampe, Ikai Lan, Teddy Lee, Jan Leike, Jade Leung, Daniel Levy, Chak Ming Li, Rachel Lim, Molly Lin, Stephanie Lin, Mateusz Litwin, Theresa Lopez, Ryan Lowe, Patricia Lue, Anna Makanju, Kim Malfacini, Sam Manning, Todor Markov, Yaniv Markovski, Bianca Martin, Katie Mayer, Andrew Mayne, Bob McGrew, Scott Mayer McKinney, Christine McLeavey, Paul McMillan, Jake McNeil, David Medina, Aalok Mehta, Jacob Menick, Luke Metz, Andrey Mishchenko, Pamela Mishkin, Vinnie Monaco, Evan Morikawa, Daniel Mossing, Tong Mu, Mira Murati, Oleg Murk, David Mély, Ashvin Nair, Reiichiro Nakano, Rajeev Nayak, Arvind Neelakantan, Richard Ngo, Hyeonwoo Noh, Long Ouyang, Cullen O'Keefe, Jakub Pachocki, Alex Paino, Joe Palermo, Ashley Pantuliano, Giambattista Parascandolo, Joel Parish, Emy Parparita, Alex Passos, Mikhail Pavlov, Andrew Peng, Adam Perelman, Filipe de Avila Belbute Peres, Michael Petrov, Henrique Ponde de Oliveira Pinto, Michael, Pokorny, Michelle Pokrass, Vitchyr H. Pong, Tolly Powell, Alethea Power, Boris Power, Elizabeth Proehl, Raul Puri, Alec Radford, Jack Rae, Aditya Ramesh, Cameron Raymond, Francis Real, Kendra Rimbach, Carl Ross, Bob Rotsted, Henri Roussez, Nick Ryder, Mario Saltarelli, Ted Sanders, Shibani Santurkar, Girish Sastry, Heather Schmidt, David Schnurr, John Schulman, Daniel Selsam, Kyla Sheppard, Toki Sherbakov, Jessica Shieh, Sarah Shoker, Pranav Shyam, Szymon Sidor, Eric Sigler, Maddie Simens, Jordan Sitkin, Katarina Slama, Ian Sohl, Benjamin Sokolowsky, Yang Song, Natalie Staudacher, Felipe Petroski Such, Natalie Summers, Ilya Sutskever, Jie Tang, Nikolas Tezak, Madeleine B. Thompson, Phil Tillet, Amin Tootoonchian, Elizabeth Tseng, Preston Tuggle, Nick Turley, Jerry Tworek, Juan Felipe Cerón Uribe, Andrea Vallone, Arun Vijayvergiya, Chelsea Voss, Carroll Wainwright, Justin Jay Wang, Alvin Wang, Ben Wang, Jonathan Ward, Jason Wei, CJ Weinmann, Akila Welihinda, Peter Welinder, Jiayi Weng, Lilian Weng, Matt Wiethoff, Dave Willner, Clemens Winter, Samuel Wolrich, Hannah Wong, Lauren Workman, Sherwin Wu, Jeff Wu, Michael Wu, Kai Xiao, Tao Xu, Sarah Yoo, Kevin Yu, Qiming Yuan, Wojciech Zaremba, Rowan Zellers, Chong Zhang, Marvin Zhang, Shengjia Zhao, Tianhao Zheng, Juntang Zhuang, William Zhuk, and Barret Zoph. 2023. GPT-4 technical report. *OpenAI Blog*.

Ivan Provilkov, Dmitrii Emelianenko, and Elena Voita. 2020. BPE-dropout: Simple and effective subword regularization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1882–1892, Online. Association for Computational Linguistics.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI Blog*.

Joshua Rozner, Christopher Potts, and Kyle Mahowald. 2021. Decrypting cryptic crosswords: Semantically complex wordplay puzzles as a target for NLP. In *Advances in Neural Information Processing Systems*.

Phillip Rust, Jonas F. Lotz, Emanuele Bugliarello, Elizabeth Salesky, Miryam de Lhoneux, and Desmond Elliott. 2023. Language modelling with pixels. In *International Conference on Learning Representations*.

Anton Schäfer, Shauli Ravfogel, Thomas Hofmann, Tiago Pimentel, and Imanol Schlag. 2024. Language imbalance can boost cross-lingual generalisation. *arXiv preprint arXiv:2404.07982*.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany. Association for Computational Linguistics.

Aleksandar Stanić, Dylan Ashley, Oleg Serikov, Louis Kirsch, Francesco Faccio, Jürgen Schmidhuber, Thomas Hofmann, and Imanol Schlag. 2023. The languini kitchen: Enabling language modelling research at different scales of compute. *arXiv preprint arXiv:2309.11197*.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023a. LLaMA: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023b. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019.

GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *International Conference on Learning Representations*.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

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

Lili Yu, Daniel Simig, Colin Flaherty, Armen Aghajanyan, Luke Zettlemoyer, and Mike Lewis. 2023. MEGABYTE: Predicting million-byte sequences with multiscale transformers. In *Thirty-seventh Conference on Neural Information Processing Systems*.

A Lemma 1 and Proof

Definition 1 (Time-dependent Conditional Entropy and Mutual Information)**.** *We define a time-dependent conditional entropy as:*

$$\mathrm{H}(\mathbf{W}_{<\mathrm{T}} \mid \mathbf{C}_{\leq \mathrm{T}}) \stackrel{\text{def}}{=} \sum_{\boldsymbol{w} \in \mathcal{W}} \sum_{\boldsymbol{c} \in \mathcal{C}} p_{\mathbb{S}}(\boldsymbol{w}, \boldsymbol{c}) \sum_{t=1}^{|\boldsymbol{w}|} \log \frac{1}{p_{\mathbb{S}}(\boldsymbol{w}_{<t} \mid \boldsymbol{c}_{\leq t})} \tag{16}$$

with $\mathrm{H}(\mathbf{W}_{<\mathrm{T}} \mid \mathbf{C}_{<\mathrm{T}})$ *defined analogously. Note that, while* $p_{\mathbb{S}}(\boldsymbol{w})$ *stands for the probability of the single event* $\{\boldsymbol{w}\}$*, we write* $p_{\mathbb{S}}(\boldsymbol{w}_{<t})$ *to represent the probability of the event composed by the union of all sequences with this prefix, i.e.,* $\cup_{\boldsymbol{w}' \in \mathcal{W}} \{\boldsymbol{w}_{<t} \circ \boldsymbol{w}'\}$*, where* $\circ$ *stands for concatenation.*

Accordingly, we define a time-dependent mutual information as:

$$\mathrm{MI}(\mathbf{W}_{<\mathrm{T}}; \mathbf{C}_{\mathrm{T}} \mid \mathbf{C}_{<\mathrm{T}}) = \mathrm{H}(\mathbf{W}_{<\mathrm{T}} \mid \mathbf{C}_{<\mathrm{T}}) - \mathrm{H}(\mathbf{W}_{<\mathrm{T}} \mid \mathbf{C}_{\leq \mathrm{T}}). \tag{17}$$

Lemma 1. *Let* $\mathbb{S} : \Sigma \to \overline{\Sigma}$ *be a deterministic function which maps duplicated subwords* $w \in \Sigma$ *to their deduplicated versions* $c \in \overline{\Sigma}$*. We then have that:*

$$\mathrm{H}_{\mathbb{S}}(\mathbf{W}) = \mathrm{H}(\mathbf{C}) - \mathrm{MI}(\mathbf{W}_{<\mathrm{T}}; \mathbf{C}_{\mathrm{T}} \mid \mathbf{C}_{<\mathrm{T}}) \tag{18}$$

Proof. We start with the definition of $\mathrm{H}_{\mathbb{S}}(\mathbf{W})$ and derive Lemma 1:

$$\begin{aligned}
\mathrm{H}_{\mathbb{S}}(\mathbf{W}) &= \sum_{\boldsymbol{w} \in \mathcal{W}} p(\boldsymbol{w}) \sum_{t=1}^{|\boldsymbol{w}|} \log \frac{1}{p_{\mathbb{S}}(\mathbb{S}(w_t) \mid \boldsymbol{w}_{<t})} && \text{(19a)} \\
&= \sum_{\boldsymbol{w} \in \mathcal{W}} \sum_{\boldsymbol{c} \in \mathcal{C}} p_{\mathbb{S}}(\boldsymbol{w}, \boldsymbol{c}) \sum_{t=1}^{|\boldsymbol{w}|} \log \frac{1}{p_{\mathbb{S}}(c_t \mid \boldsymbol{w}_{<t})} && p_{\mathbb{S}}(\boldsymbol{w}, \boldsymbol{c}) = p(\boldsymbol{w}) \text{ if } \mathbb{S}(\boldsymbol{w}) = \boldsymbol{c} \text{ else } 0 \quad \text{(19b)} \\
&= \sum_{\boldsymbol{w} \in \mathcal{W}} \sum_{\boldsymbol{c} \in \mathcal{C}} p_{\mathbb{S}}(\boldsymbol{w}, \boldsymbol{c}) \sum_{t=1}^{|\boldsymbol{w}|} \log \frac{1}{p_{\mathbb{S}}(c_t \mid \boldsymbol{c}_{<t}, \boldsymbol{w}_{<t})} && p_{\mathbb{S}}(\boldsymbol{c}_{<t} \mid \boldsymbol{w}_{<t}) \text{ is deterministic} \quad \text{(19c)} \\
&= \sum_{\boldsymbol{w} \in \mathcal{W}} \sum_{\boldsymbol{c} \in \mathcal{C}} p_{\mathbb{S}}(\boldsymbol{w}, \boldsymbol{c}) \sum_{t=1}^{|\boldsymbol{w}|} \log \frac{p_{\mathbb{S}}(\boldsymbol{w}_{<t} \mid \boldsymbol{c}_{<t})}{p_{\mathbb{S}}(c_t \mid \boldsymbol{c}_{<t})\, p_{\mathbb{S}}(\boldsymbol{w}_{<t} \mid \boldsymbol{c}_{\leq t})} && \text{Bayes rule} \quad \text{(19d)} \\
&= \sum_{\boldsymbol{w} \in \mathcal{W}} \sum_{\boldsymbol{c} \in \mathcal{C}} p_{\mathbb{S}}(\boldsymbol{w}, \boldsymbol{c}) \sum_{t=1}^{|\boldsymbol{w}|} \left(\log \frac{1}{p(c_t \mid \boldsymbol{c}_{<t})} + \log \frac{p_{\mathbb{S}}(\boldsymbol{w}_{<t} \mid \boldsymbol{c}_{<t})}{p_{\mathbb{S}}(\boldsymbol{w}_{<t} \mid \boldsymbol{c}_{\leq t})} \right) && \text{Split log} \quad \text{(19e)} \\
&= \sum_{\boldsymbol{c} \in \mathcal{C}} p(\boldsymbol{c}) \sum_{t=1}^{|\boldsymbol{c}|} \log \frac{1}{p(c_t \mid \boldsymbol{c}_{<t})} + \sum_{\boldsymbol{w} \in \mathcal{W}} \sum_{\boldsymbol{c} \in \mathcal{C}} p_{\mathbb{S}}(\boldsymbol{w}, \boldsymbol{c}) \sum_{t=1}^{|\boldsymbol{w}|} \log \frac{p_{\mathbb{S}}(\boldsymbol{w}_{<t} \mid \boldsymbol{c}_{<t})}{p_{\mathbb{S}}(\boldsymbol{w}_{<t} \mid \boldsymbol{c}_{\leq t})} && \text{Split sum} \quad \text{(19f)} \\
&= \mathrm{H}(\mathbf{C}) + \sum_{\boldsymbol{w} \in \mathcal{W}} \sum_{\boldsymbol{c} \in \mathcal{C}} p_{\mathbb{S}}(\boldsymbol{w}, \boldsymbol{c}) \sum_{t=1}^{|\boldsymbol{w}|} \log \frac{p_{\mathbb{S}}(\boldsymbol{w}_{<t} \mid \boldsymbol{c}_{<t})}{p_{\mathbb{S}}(\boldsymbol{w}_{<t} \mid \boldsymbol{c}_{\leq t})} && \text{Definition of } \mathrm{H}(\mathbf{C}) \quad \text{(19g)} \\
&= \mathrm{H}(\mathbf{C}) + \mathrm{H}(\mathbf{W}_{<\mathrm{T}} \mid \mathbf{C}_{\leq \mathrm{T}}) - \mathrm{H}(\mathbf{W}_{<\mathrm{T}} \mid \mathbf{C}_{<\mathrm{T}}) && \text{Definitions of conditional entropies} \quad \text{(19h)} \\
&= \mathrm{H}(\mathbf{C}) - \mathrm{MI}(\mathbf{W}_{<\mathrm{T}}; \mathbf{C}_{\mathrm{T}} \mid \mathbf{C}_{<\mathrm{T}}) && \text{Definition of mutual information} \quad \text{(19i)}
\end{aligned}$$

Note that $p_{\mathbb{S}}(\boldsymbol{w}, \boldsymbol{c}) > 0 \implies \mathbb{S}(\boldsymbol{w}) = \boldsymbol{c} \implies |\boldsymbol{w}| = |\boldsymbol{c}|$, which is required for arriving at Eq. (19f). □

B Near Duplicates in Modern LLMs

Table 7 shows the near duplicate rates in vocabularies of modern large language models, namely GPT-3.5 and GPT-4 models (Brown et al., 2020; OpenAI et al., 2023), Claude 2.1 (Anthropic, 2023), Llama (Touvron et al., 2023a,b), Mistral 7B and Mixtral 8x7B (Jiang et al., 2023, 2024), and Gemma (Gemma Team et al., 2024).

		Near Duplicate Rate			
Model	**Vocabulary Size**	$\mathbb{S}_{\text{space}}$	$\mathbb{S}_{\text{lower}}$	$\mathbb{S}_{\text{plural}}$	$\mathbb{S}_{\text{all}}$
GPT-{3.5, 4, 4-turbo}	100k	19%	24%	9%	43%
Claude 2.1	65k	25%	23%	9%	46%
Llama 1 & 2	32k	17%	31%	22%	35%
Mistral 7B & Mixtral 8x7B	32k	15%	32%	23%	37%
Gemma 7B	256k	21%	20%	7%	39%

Table 7: Near duplicate rates of modern LLM vocabularies. Computed as $1 - \frac{|\overline{\Sigma}|}{|\Sigma|}$, i.e., the ratio by which the size of the vocabulary decreases when mapping all subwords to their canonical versions using the respective $\mathbb{S}$. For the definitions of the deduplication mappings $\mathbb{S}$, see §5.2.1.

C Near Duplicates under $\mathbb{S}_{\text{space}}$

Near duplicates under $\mathbb{S}_{\text{lower}}$ and $\mathbb{S}_{\text{plural}}$ are generally very close in meaning. $\mathbb{S}_{\text{lower}}$ pairs often consist of the lowercase and the capitalized version of a subword, where the latter might appear at the beginning of sentences. And (sub)word pairs under $\mathbb{S}_{\text{plural}}$ tend to refer to the same lemma. For $\mathbb{S}_{\text{space}}$, however, the contexts in which subword variants with and without a leading space appear might differ much more: in particular, we observe this with shorter subwords that can appear both as their own word (e.g., *_he* in 'and he writes') and as a substring of a longer unrelated word (e.g., *he* in 'breathe').

To quantify how frequently $\mathbb{S}_{\text{space}}$ near duplicate pairs show such differences, we manually inspect 100 randomly sampled pairs and note whether the subwords convey comparable meaning in their first occurrences in the training data. We find that 53% of the pairs carry highly similar meaning in both occurrences. Examples include use in compound words (e.g., *_writing* in isolation vs *writing* in 'handwriting', or *_held* in isolation vs *held* in 'long-held') or after special characters like newlines, parentheses, or quotation marks (e.g. *_wide* in isolation vs *wide* in '(wide [...])'). Of the remaining 47%, around half appear in contexts that are clearly different (e.g., *he* and *_he* as in example above) while the rest are very short subwords where the conveyed meaning is hard to determine (e.g. *_sche* in 'schemed' vs *sche* in 'Nietzsche').

D Effect of Added Parameters for Duplicates

A model $\widehat{p}(\boldsymbol{w})$ over duplicated subwords has more embedding parameters than its deduplicated counterpart $\widehat{p}(\boldsymbol{c})$. In the perfect duplication setting, however, these extra parameters should not yield an unfair advantage. In particular, assume an equiprobable duplicates setting, where $p(w_{\textcircled{i}} \mid c_{\textcircled{i}}) = p(w'_{\textcircled{i}} \mid c_{\textcircled{i}}) = 0.5$. We can show that, in this setting, if there exists an optimal model $\widehat{p}(\boldsymbol{w})$ (which achieves $\widehat{\mathrm{H}}(\mathbf{W}) = \mathrm{H}(\mathbf{W})$), then there also exists an equivalent model $\widehat{p}(\boldsymbol{c})$. Further, the input and output embeddings for duplicates $w_{\textcircled{i}}$ and $w'_{\textcircled{i}}$ are perfectly interchangeable; the added embeddings thus provide no benefits, as the embedding of any $w'_{\textcircled{i}}$ can be replaced with the embedding of the respective $w_{\textcircled{i}}$ without affecting performance.

To see this, note that for the optimal $\widehat{p}(\boldsymbol{w})$, we have $\widehat{p}(\boldsymbol{w}) = p(\boldsymbol{w})$ almost everywhere. Consequently, by Bayes, $\widehat{p}(w \mid \boldsymbol{w}_{<t}) = p(w \mid \boldsymbol{w}_{<t})$. We thus obtain

$$\widehat{p}(w_{\textcircled{i}} \mid \boldsymbol{w}_{<t}) = \widehat{p}(w'_{\textcircled{i}} \mid \boldsymbol{w}_{<t}) \tag{20}$$

which means that the model makes identical predictions for duplicate subwords. Predictions are thus not affected if we assign all $w'_{\textcircled{i}}$ the output embedding of the respective $w_{\textcircled{i}}$ (or vice-versa).

An analogous argument holds for input embeddings. Let $\boldsymbol{w}_{<t}[w'_{\textcircled{i}} \to w_{\textcircled{i}}]$ denote the token sequence obtained when replacing every occurrence of $w'_{\textcircled{i}}$ in $\boldsymbol{w}_{<t}$ with $w_{\textcircled{i}}$. For the true distribution, we know that $p(w_{\textcircled{i}} \mid \boldsymbol{w}_{<t}) = p(w_{\textcircled{i}} \mid \boldsymbol{w}_{<t}[w'_{\textcircled{i}} \to w_{\textcircled{i}}])$ due to the perfect equivalence, and thus also:

$$\widehat{p}(w_{\textcircled{i}} \mid \boldsymbol{w}_{<t}) = \widehat{p}(w_{\textcircled{i}} \mid \boldsymbol{w}_{<t}[w'_{\textcircled{i}} \to w_{\textcircled{i}}]) \tag{21}$$

This means that $\widehat{p}(\boldsymbol{w})$ makes identical predictions, whether we replace $w'_{\textcircled{i}}$ with $w_{\textcircled{i}}$ in the context or not. If we assign $w'_{\textcircled{i}}$ the input embedding of $w_{\textcircled{i}}$, model performance is not affected. By applying this argument iteratively for all i, we can show that all duplicates' input embeddings can be made identical. If the input embeddings of all duplicates can be made identical, then there exists a model $\widehat{p}(c_{\textcircled{i}} \mid \boldsymbol{c}_{<t})$ which, when given as input $\mathbb{S}(\boldsymbol{w})$, outputs the same probability as $\widehat{p}(w_{\textcircled{i}} \mid \boldsymbol{w}_{<t}) + \widehat{p}(w'_{\textcircled{i}} \mid \boldsymbol{w}_{<t})$. In particular, let $\mathbf{v}_w$ be the output embeddings associated with w. We can show that:

$$\begin{aligned}
\widehat{p}(w_{\textcircled{i}} \mid \boldsymbol{w}_{<t}) + \widehat{p}(w'_{\textcircled{i}} \mid \boldsymbol{w}_{<t}) &= 2\,\widehat{p}(w_{\textcircled{i}} \mid \boldsymbol{w}_{<t}) && \text{(22a)}\\
&= 2\,\frac{\exp(\mathbf{v}_{w_{\textcircled{i}}}\,\mathbf{h})}{\sum_{w \in \Sigma} \exp(\mathbf{v}_w\,\mathbf{h})} && \text{(22b)}\\
&= 2\,\frac{\exp(\mathbf{v}_{w_{\textcircled{i}}}\,\mathbf{h})}{\sum_{j=1}^{|\overline{\Sigma}|} \exp(\mathbf{v}_{w_{\textcircled{j}}}\,\mathbf{h}) + \sum_{j=1}^{|\overline{\Sigma}|} \exp(\mathbf{v}_{w'_{\textcircled{j}}}\,\mathbf{h})} && \text{(22c)}\\
&= \frac{2\,\exp(\mathbf{v}_{w_{\textcircled{i}}}\,\mathbf{h})}{2\sum_{j=1}^{|\overline{\Sigma}|} \exp(\mathbf{v}_{w_{\textcircled{j}}}\,\mathbf{h})} && \text{(22d)}\\
&= \frac{\exp(\mathbf{v}_{c_{\textcircled{i}}}\,\mathbf{h})}{\sum_{j=1}^{|\overline{\Sigma}|} \exp(\mathbf{v}_{c_{\textcircled{j}}}\,\mathbf{h})} && \text{(22e)}\\
&= \widehat{p}(c_{\textcircled{i}} \mid \boldsymbol{c}_{<t}) && \text{(22f)}
\end{aligned}$$

So we can construct a $\widehat{p}(\boldsymbol{c})$ model from $\widehat{p}(\boldsymbol{w})$ by simply making the embeddings of $c_{\textcircled{i}}$ the same as $w_{\textcircled{i}}$. The identical input embeddings preserve the value of the hidden state $\mathbf{h}$ for any context, which, along with the identical output embeddings, ensures identical model outputs.

E Embedding Similarity Results

While in Fig. 6, it appears that a balance between duplicated and non-duplicated subwords (i.e. $p(w'_{\textcircled{i}} \mid c_{\textcircled{i}}) = 1 - p(w_{\textcircled{i}} \mid c_{\textcircled{i}})$ close to 0.5) leads to higher similarity of their embeddings, Fig. 7 shows that the underlying factor driving embedding alignment seems to be subword frequency, or more precisely, how often the rarer version of the subword (here always w' as $p(w'_{\textcircled{i}} \mid c_{\textcircled{i}}) < 0.5$) occurs.

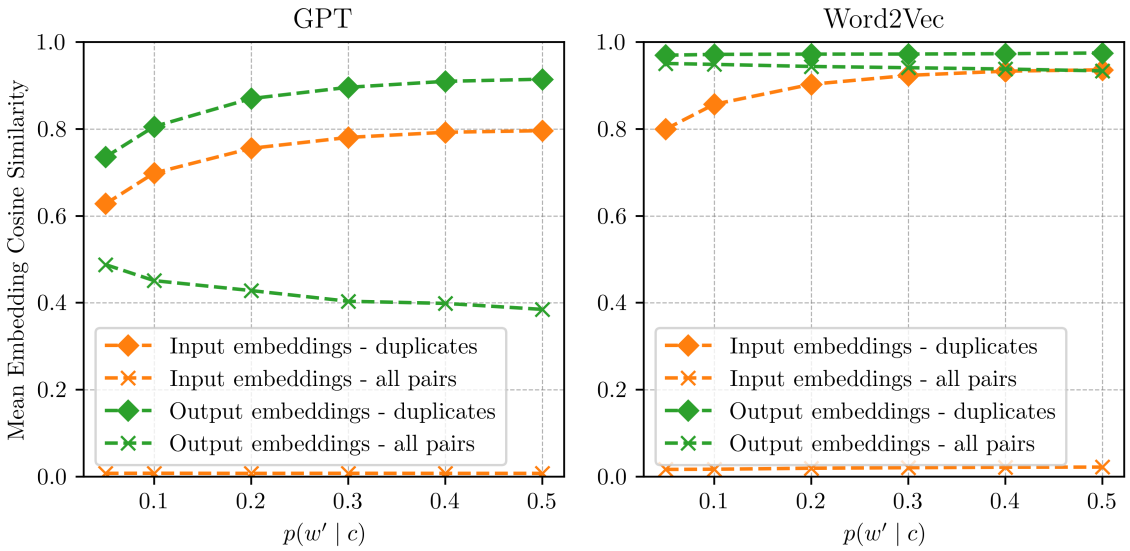

Figure 6: Embedding cosine similarity of duplicates $w_{\textcircled{i}}$, $w'_{\textcircled{i}}$ and random pairs $w_{\textcircled{i}}$, $w_{\textcircled{j}}$ to control for anisotropy. Left: Our GPT model. Right: Word2vec embeddings trained on the same data (computed with Gensim).

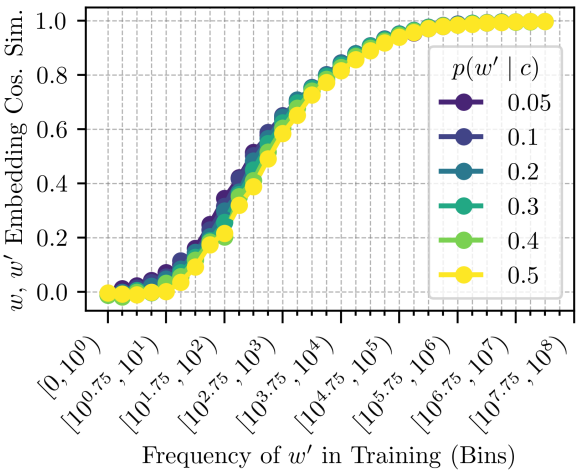

Figure 7: Embedding cosine similarity of duplicates $w_{\textcircled{i}}$, $w'_{\textcircled{i}}$, by frequency of the rarer $w'_{\textcircled{i}}$. Frequencies binned and similarities averaged per bin.

F Effect of Deduplicated Subwords in the Context

An increased number of deduplicated subwords in the context seems to generally lead to a steeper increase in surprisal than an increased number of non-deduplicated subwords, at least in the local context of 16 tokens (see Figure Fig. 8). For the full context, the trend is less clear.

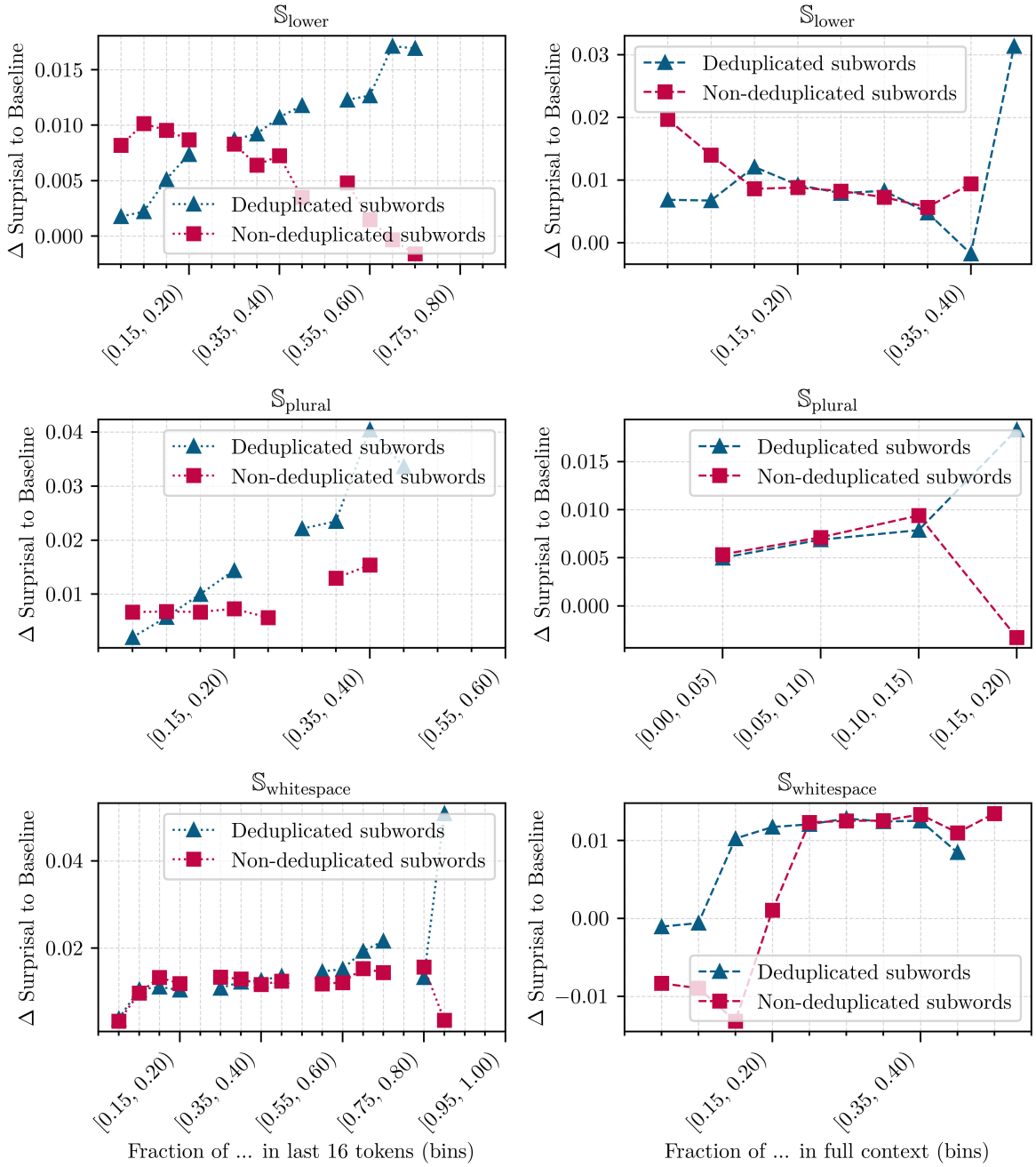

Figure 8: Deduplication of half of the vocabulary. Difference to baseline ($\widehat{p}(\boldsymbol{w})$) surprisal, depending on fraction of (non-)deduplicated subwords in context. Fractions binned and surprisal differences averaged per bin. To reduce noise and ensure readability, we only plot bins that contain at least 1k predicted tokens.