How Do Language Models Acquire Character-Level Information?

Soma Sato **Ryohei Sasano**
Graduate School of Informatics, Nagoya University
sato.soma24@gmail.com sasano@i.nagoya-u.ac.jp

Abstract

Language models (LMs) have been reported to implicitly encode character-level information, despite not being explicitly provided during training. However, the mechanisms underlying this phenomenon remain largely unexplored. To reveal the mechanisms, we analyze how models acquire character-level knowledge by comparing LMs trained under controlled settings, such as specifying the pre-training dataset or tokenizer, with those trained under standard settings. We categorize the contributing factors into those independent of tokenization. Our analysis reveals that merge rules and orthographic constraints constitute primary factors arising from tokenization, whereas semantic associations of substrings and syntactic information function as key factors independent of tokenization.

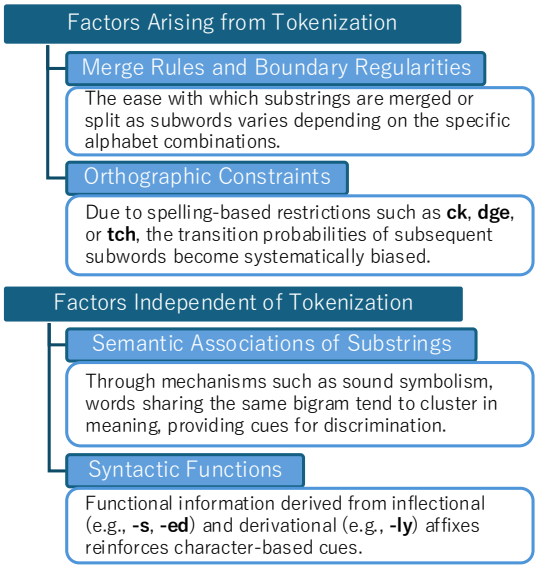

Figure 1: Factors contributing to character-level information acquisition in LMs.

1 Introduction

In recent years, language models (LMs) have made substantial progress in a wide range of natural language processing tasks. These models are thought to acquire advanced understanding and generation capabilities through pre-training on massive text corpora, capturing high-level linguistic information, such as lexical, syntactic, and contextual aspects (Devlin et al., 2019; Radford et al., 2019; Tenney et al., 2019). Meanwhile, the units used in LMs training are subwords, which are segmented by the tokenizer, and finer-grained information at the character level is not explicitly provided during training. It might therefore be assumed that LMs do not retain information about the characters within each subword. Prior studies, however, have shown that information about characters and substrings within subwords is, to some extent, reflected in LMs' internal representations (Edman et al., 2024; Kaushal and Mahowald, 2022; Itzhak and Levy, 2022). Nevertheless, although these works establish that LMs learn character-level information, research that elucidates the mechanisms of this acquisition remains largely unexplored.

To address this open question, we systematically examine the mechanisms by which LMs acquire character-level information. We first manipulate the pretraining data to test whether character-level information can be learned by LMs trained on the manipulated corpora. To evaluate this, we employ probing classifiers, following Kaushal and Mahowald (2022), which take token embeddings as input and are trained to make a binary decision about whether a token contains a given alphabetic character. Building on these results, we categorize the sources of character acquisition into factors arising from tokenization and factors independent of tokenization, as illustrated in Figure 1, and quantify their respective contributions. Our analysis reveals that merge rules and orthographic constraints constitute primary factors arising from tokenization, while semantic associations of substrings and syntactic information serve as key factors independent of tokenization.

Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics
Volume 1: Long Papers, pages 5987–5997

2 Background

The possibility that LMs implicitly retain character-level knowledge has been reported through analyses of surface-level information. Kaushal and Mahowald (2022) demonstrated through probing experiments on subword embeddings that larger models are more likely to predict the presence of specific characters and substrings. Itzhak and Levy (2022) reported that word spellings can be partially reconstructed from LM embedding matrices. Hiraoka and Okazaki (2024) showed that while exact reconstruction of subword composition is limited, information about token length and substrings is reflected in internal representations. Furthermore, the CUTE benchmark by Edman et al. (2024) evaluated LMs on tasks probing understanding of string composition and character- and word-level operations, reporting that models are fragile to character-level edits such as insertion and reordering, but exhibit a certain robustness in recognizing inclusion relations. Analyses comparing morphologically functional substrings (e.g., prefixes, suffixes and stems) with meaningless substrings also showed that the former tend to carry more substring-level information (Ciaccio et al., 2025).

However, these studies stop at showing that LMs encode some information about characters within subwords, and offer only limited analysis of the mechanisms underlying such acquisition. In particular, no systematic analysis has been conducted on how individual factors such as merge rules or orthographic constraints in subword segmentation affect character-level information acquisition. Our work aims to fill this gap by clarifying the concrete mechanisms of character-level information acquisition through systematic analysis.

We quantitatively analyze how subword segmentation itself affects the learning of character-level information. As a prerequisite, we briefly review representative subword tokenization algorithms. In neural language models, subword representations are widely used to address the out-of-vocabulary (OOV) problem arising from fixed-size vocabularies. Byte-pair encoding (BPE) (Sennrich et al., 2016) is a greedy tokenization method based on merge rules of frequent character pairs, and has been shown to improve translation accuracy while enhancing vocabulary generalization. BPE starts from characters and iteratively merges the most frequent adjacent pairs in the training corpus to obtain a sequence of merge rules, which are then greedily applied for tokenization. WordPiece (Schuster and Nakajima, 2012; Wu et al., 2016) incrementally expands the vocabulary by evaluating the contribution of candidate merges to the increase in corpus log-likelihood, and has been widely adopted in Google's translation systems and BERT models. Unlike BPE, it relies on a likelihood-based criterion for vocabulary updates. The unigram language model (Kudo and Richardson, 2018; Kudo, 2018) instead assumes a probabilistic generative model consisting of a subword vocabulary and probabilities for each subword, and optimizes it using the expectation maximization (EM) algorithm (Dempster et al., 1977). Because segmentation is stochastic, this approach is also known to function as a form of regularization and data augmentation during training.

3 Experimental Setup

In this section, we describe the setup for analyzing the factors underlying character-level information acquisition. We first introduce a binary classification task to measure how much character-level information is retained in model representations. We then pretrain small-scale LMs to verify whether character-level knowledge can indeed be acquired through pretraining.

3.1 Probing Task

Following Kaushal and Mahowald (2022), we quantify the extent to which LMs acquire character-level information by training a multilayer perceptron (MLP) on pretrained token embeddings to predict whether a given token contains a particular alphabetic character. However, while longer tokens are more likely to contain a given character, previous work did not account for the effect of token length. For example, the probability that a three-character token contains "e" is much lower than that of a ten-character token. In addition, shorter words tend to be high frequency function words, while longer words are often lower frequency content words, meaning that token length is also correlated with semantic information. Thus, pretrained embeddings inherently encode information about token length, which may influence classification performance. To avoid token length from acting as a confounding factor in classification, we matched the token length distributions between positive and negative examples.

Figure 2 illustrates the overall procedure. In

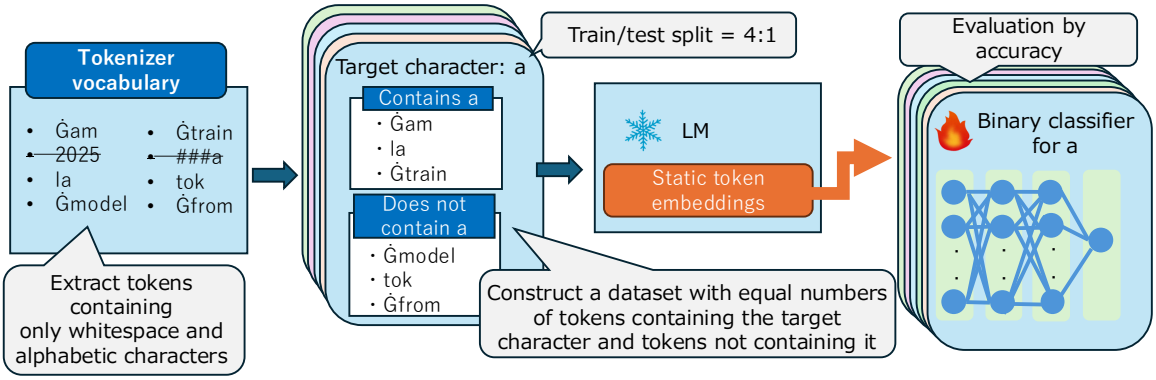

Figure 2: Overview of the probing task, which evaluates whether token embeddings encode character-level information in LMs. Ġ is a symbol used in BPE tokenization to indicate the space preceding a word.

this study, we performed tokenization under the constraint that subwords are not allowed to contain whitespace except at the beginning. We first excluded from the learned tokenizer vocabulary any tokens containing characters other than a leading whitespace symbol (Ġ) or alphabetic letters. For each character α, we constructed a dataset $D(\alpha) = \{(w_1, y_1), \ldots, (w_d, y_d)\}$, where y_i indicates whether token w_i contains α. The numbers of positive and negative examples were balanced. We used a two hidden layer neural network with SELU activation in the first layer, Tanh in the second layer, and a dropout rate of 0.1. Training was performed with a sigmoid cross entropy loss, the Adam optimizer (learning rate 1e-3), and for 3 epochs. We report the micro-averaged accuracy across all alphabetic characters.

3.2 Pretraining Setup

We verify whether character-level knowledge can be acquired through pretraining. Given computational constraints, we adopt two small-scale LMs: BERT-Tiny (Turc et al., 2019) and nanoGPT (Karpathy, 2022). BERT-Tiny is an encoder-only model with 2 encoder blocks, 2 attention heads, and a hidden size of 128, trained on masked language modeling (MLM). In contrast, nanoGPT is a decoder-only model with 12 decoder blocks, 12 attention heads, and a hidden size of 768, trained on next-word prediction.

For the pretraining corpus, we use FineWeb (Penedo et al., 2024), a high quality Web text dataset extracted and filtered from CommonCrawl. Specifically, we employ 10B token sample dataset.[1] We first trained tokenizers on this dataset, applying WordPiece for BERT-Tiny and BPE for nanoGPT.[2]

For BERT-Tiny, we set the input length to 256 tokens, the batch size per GPU to 64, the number of epochs to 1, the MLM mask rate to 0.15, the learning rate to 2e-5, and the gradient clipping value to 1.0. For nanoGPT, we used a micro-batch size of 32 with gradient accumulation of 40, which corresponds to an effective batch size of 1,280, and trained with a sequence length of 1,024. Each iteration processed approximately 1.31M tokens, and we trained for 10,000 iterations, amounting to about 13.1B tokens in total.[3] The learning rate was 6e-4 with weight decay 1e-1, using 2,000 warm-up iterations followed by cosine annealing scheduling. On a single A100 GPU, pretraining took 85 hours for BERT-Tiny and 24 hours for nanoGPT.

3.3 Preliminary Experiments

We applied our pretrained BERT-Tiny and nanoGPT models to the probing task, and compared them against publicly released models: GPT-2 (Radford et al., 2019), BERT-base-uncased (Devlin et al., 2019), and GPT-J (Wang and Komatsuzaki, 2021). The 'matched' column in Table 1 presents the experimental results.[4] While our models exhibit slightly lower accuracies than the large

[1] https://huggingface.co/datasets/HuggingFaceFW/fineweb/viewer/sample-10BT

[2] In this study, the tokenizer is trained such that whitespace characters do not appear at the beginning of subwords.

[3] We trained on 10B tokens and then continued training on an additional 3.1B overlapping tokens from the beginning of the dataset.

[4] Kaushal and Mahowald (2022) report that GPT-J, GPT-2, and BERT-base-uncased achieve higher accuracies in Table 1 of their paper. However, a closer look at Figure 2 in the same paper indicates that these values correspond only to the case where the target character is restricted to z. When all alphabet characters are taken into account, the accuracies decline and become comparable to those observed in the present study.

	Length dist.	
	Matched	Unmatched
Public models		
GPT-J	84.0	88.0
GPT-2	75.5	79.4
BERT-base-uncased	67.1	69.1
Our trained models		
nanoGPT	66.7	69.6
BERT-Tiny	60.8	63.3

Table 1: Comparison of publicly available models and our trained models. 'Matched' indicates the accuracy when the token length distribution between positive and negative examples is matched, while 'Unmatched' indicates the accuracy without matching.

public models, they nonetheless acquire character-level information. Notably, nanoGPT achieves accuracy comparable to BERT-base-uncased, making it a suitable target for our subsequent analyses. A detailed analysis of how accuracy improves throughout the training process is provided in Appendix A. We therefore focus primarily on nanoGPT in the following sections.

The column labeled unmatched in Table 1 shows the performance in the probing task when the length distributions of positive and negative examples are not aligned. This setting, adopted in prior studies, randomly selects positive examples from vocabulary subwords containing the target character and negative examples from those not containing it. As a result, a bias arises whereby longer subwords are more likely to be chosen as positives, leading to slightly higher accuracy compared to the matched setting we adopt.

To further examine the degree to which character-level information is learned depending on subword length, we conducted additional experiments with nanoGPT. We divided the test data into five categories based on subword length: 3 or fewer, 4, 5, 6, and 7 or more. We then measured the classification accuracy for each category. The proportions of these categories in the overall vocabulary were 10.9% (3 or fewer), 15.9% (4), 15.6% (5), 14.9% (6), and 42.7% (7 or more). We then conducted evaluations using only the data included in each category. It should be noted that the original datasets were constructed to match the length distributions between positive and negative examples, ensuring that the numbers of positives and negatives are balanced in every subset. Table 2 presents the results. These results indicate that shorter subwords yield higher probing task accuracy.

Length	≤ 3	4	5	6	$7\leq$	All
Acc.	80.4	71.9	66.7	63.9	62.3	66.7

Table 2: Accuracy of the probing task across groups defined by token length.

4 Analysis via Input String Transformations

In this section, we analyze how different properties of input strings affect subword embeddings on encoding character-level information. Specifically, we investigate how applying various transformations to the input text affects performance on the probing task.

4.1 Design of Input Transformations

The accuracy observed for nanoGPT may partly reflect correlations between word forms and meanings. For example, cognitive effects such as the bouba/kiki phenomenon (Ramachandran and Hubbard, 2001) suggest that associations between form and meaning can systematically influence learning, and such lexical cues may contribute to character acquisition. To test this hypothesis, we constructed transformed versions of the FineWeb dataset.

First, to remove the influence of associations between form and meaning, we performed a transformation called word substitution (WordSub), in which each subword in the tokenizer vocabulary was replaced with a uniquely determined random string. This preserves grammatical structure and sentence-level semantics, but eliminates systematic relations between form and meaning.

Second, to verify that models do not acquire character-level information when trained on data in which the regularities of character sequences other than whitespace are completely eliminated, we applied a transformation called character perturbation (CharPert), in which each alphabetic character was randomly replaced while preserving letter case information. For example, given the sentence "she dreams she is dreaming", CharPert yields a randomized string such as "xwe kmdawa dmj ie dkjrumez". In contrast, WordSub consistently replaces *she* with the same substitute (e.g., **wij**), producing a sentence such as "**wij** gkaswd **wij** da qwmmfans". Table 3 illustrates concrete examples of these transformations.

In addition, we compared the setting using a BPE tokenizer (BPE) with the setting using words as tokens defined by whitespace segmentation without

Method	Example
Original	She plans she will plan later.
WordSub	Wij gkasw wij djaa qwma mfans.
CharPert	Xwe kmdaw dmj iewa dkjs rumez.

Table 3: Examples of input string transformations using WordSub and CharPert.

	FineWeb	WordSub	CharPert
BPE	66.7	53.8	58.2
Word	61.2	50.0	50.0

Table 4: Comparison of accuracy with and without tokenization and the effect of pretraining data differences.

any tokenizer (Word), in order to examine the effect of tokenization on character-level information learning. The original vocabulary size was 50,256, and the target vocabulary consisted of 42,875 tokens in the BPE setting and 42,736 tokens in the Word setting.

4.2 Results

Table 4 summarizes the results. Although we expected the accuracy for the CharPert dataset to be around 50.0%, the BPE setting achieved 58.2%. This indicates that factors arising purely from subword segmentation algorithm contribute to character-level information acquisition, independent of linguistic properties. Next, for the WordSub dataset, accuracies decreased relative to FineWeb by 12.9% (BPE) and 11.2% (Word), confirming that associations between form and meaning play a strong role in learning character-level information. Overall, these results show that both factors arising from tokenization and factors independent of tokenization contribute to character-level information acquisition. In the following sections, we analyze these two perspectives in greater detail.

5 Analysis of Factors Arising from Tokenization

We consider two factors arising from tokenization: the merge rules that construct subwords and the orthographic constraints between subwords within words. In this section, we design tokenizers to isolate these factors and quantitatively assess their impact.

5.1 Effect of Merge Rules

A key difference between settings with and without subword segmentation on the CharPert dataset is whether information about the segmentation procedure is preserved or not. Suppose the BPE merge

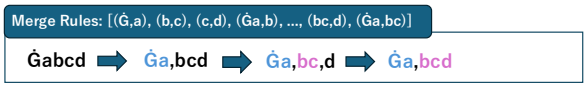

Figure 3: Case where merge (b,c) has higher priority than (c,d).

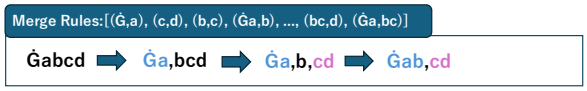

Figure 4: Case where merge (c,d) has higher priority than (b,c).

rules are defined as [(Ġ,a), (b,c), (c,d), (Ġa,b), ..., (bc,d), (Ġa,bc)]. These rules are applied in the listed order of priority. Consider how the token Ġabcd would be segmented. It is first split into characters [Ġ, a, b, c, d]. Applying the first rule (Ġ,a) yields [Ġa, b, c, d]. Applying (b,c) gives [Ġa, bc, d]. Since neither (c,d) nor (Ġa,b) is applicable, (bc,d) applies, resulting in [Ġa, bcd]. The flow of the series of processes is shown in Figure 3. If, alternatively, the merge (c,d) had higher priority than (b,c), we would first apply (c,d) to obtain [Ġa, b, cd] after [Ġa, b, c, d]. We would then apply (Ġa,b) to obtain [Ġab, cd], The flow of the series of processes is shown in Figure 4.

In this way, if the priority of a merge rule is low, the corresponding string is more likely to be split. As a result, the model parameters statistically accumulate the tendency for a subword ending with a particular character to be followed by a subword beginning with certain characters, which influences the acquisition of character-level information.

5.1.1 Analysis Procedure

To evaluate the effect of merge rules on character-level information acquisition by LMs, we use a controlled tokenizer designed under fully specified conditions. In BPE, merge rules depend on training data, making it difficult to disentangle data properties from rule effects. Explicit definition of the merge rules allows us to directly analyze how they help language models acquire character-level information.

Concretely, we restrict the tokenizer vocabulary to combinations of lowercase alphabetic characters with lengths from 1 to 3, both with and without a leading whitespace symbol. The resulting vocabulary size is 36,556.[5] We configure merge rules so

[5]There are 26, 676 (=26^2), and 17,576 (=26^3) combinations of lengths 1, 2, and 3, respectively, totaling 18,278. Doubling this to account for the presence/absence of the leading whitespace gives 36,556.

	1st char	2nd char	3rd char	all
w/ Ġ	49.8	54.4	90.6	62.4
w/o Ġ	73.8	61.2	72.5	

Table 5: Probing task accuracy with a controlled tokenizer. The rightmost column reports the results obtained with all positive examples, whereas the three central columns report results from experiments where the positive examples were partitioned into six groups.

that odd-length strings prefer longer prefixes and even-length strings split evenly. For example, abc is formed from (ab, c), and Ġdef from (Ġd, ef). Although the order of merge rules is shuffled, we define their application to proceed in the following sequence: ⟨Ġ+1 character⟩, ⟨2 characters⟩, ⟨Ġ+2 characters⟩, ⟨3 characters⟩, and ⟨Ġ+3 characters⟩. Here, Ġ+n characters means that the whitespace symbol Ġ is followed by n characters, and ⟨⟩ denotes merge rules.

To remove information arising from the merge rules, we additionally construct a pretraining dataset by applying a random token substitution to the tokenized CharPert corpus, preserving whether tokens include a leading whitespace and their length. For example, a sequence such as "Ġdef, ac, Ġdef, Ġqda" is randomly replaced with "Ġfda, po, Ġgfs, Ġrer." This procedure eliminates statistical patterns induced by the merge rules, allowing us to quantify the contribution of the rules themselves.

Our analysis proceeds as follows. We first tokenize CharPert dataset with the controlled tokenizer, and then create a randomly substituted variant of the tokenized data. We pretrain nanoGPT on each dataset and evaluate both models on the probing task. To analyze merge rule effects in more detail, we further partition positive examples into six groups according to two factors: (i) whether they include a leading whitespace Ġ, and (ii) the position of the target character within the subword, i.e., first, second, or third character. We then report accuracies for each condition.

5.1.2 Results

Table 5 shows the experimental results. Using the controlled tokenizer on the CharPert dataset, the accuracy reached 62.4%, exceeding the 50.0% level even without training a tokenizer on the dataset. In contrast, when we applied random substitution to the tokenized CharPert data, accuracy dropped to 50.1%. This indicates that, in BPE, merge rules themselves impact character-level information acquisition. Table 5 also reports experimental results

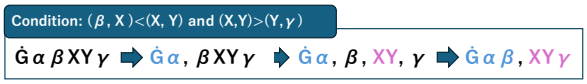

Figure 5: The case where Gαβ, XYγ is segmented as Gαβ and XYγ.

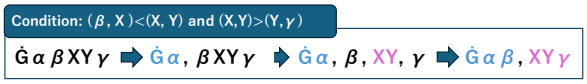

Figure 6: The case where GαβXγδ is segmented as Gαβ and Xγδ.

where the positive examples were partitioned into six groups, defined by the target character position and the presence or absence of the leading whitespace Ġ. The results indicate that accuracy differs substantially across conditions.

We now consider why the accuracies for the first and second characters with a leading whitespace are high (Figure 5). Suppose ĠαβXYγ is segmented as [Ġαβ, XYγ], where X and Y are specific alphabetic characters, and α, β, and γ are arbitrary alphabetic characters. Initially, the sequence is segmented into individual characters: [Ġ, α, β, X, Y, γ]. Next, since processing begins with ⟨Ġ+1 character⟩, the sequence becomes [Ġα, β, X, Y, γ]. For the segmentation [Ġαβ, XYγ] to be obtained, the sequence must first become [Ġα, β, XY, γ]. The necessary condition for this is that m(β, X) < m(X, Y) and m(X, Y) > m(Y, γ). Here, m(X, Y) denotes the strength of the merge between X and Y, which reflects the degree to which the rule is assigned higher priority in the merge rule table. Once the segmentation [Ġα, β, XY, γ] is reached, ⟨Ġ+2 characters⟩ and ⟨3 characters⟩ are subsequently applied, yielding [Ġαβ, XYγ]. In other words, the necessary condition for Ġαβ, XYγ to be obtained is m(β, X) < m(X, Y) and m(X, Y) > m(Y, γ), and whether Ġαβ appears depends on the choice of XY. Information about which combinations are more likely is statistically encoded in the model parameters, and the presence of XY in subword tokens is thus indirectly learned.

As for why the accuracy of the first character without a leading whitespace is higher than that of the second character (Figure 6), consider the case where ĠαβXγδ is segmented as [Ġαβ, Xγδ], where X is a specific alphabetic character and α, β, γ, and δ are arbitrary alphabetic characters. The condition for obtaining [Ġαβ, Xγδ] is m(β, X) < m(X, γ) and m(X, γ) > m(γ, δ), and which Ġαβ appears depends on the choice of X.

The reasons why the third character without a

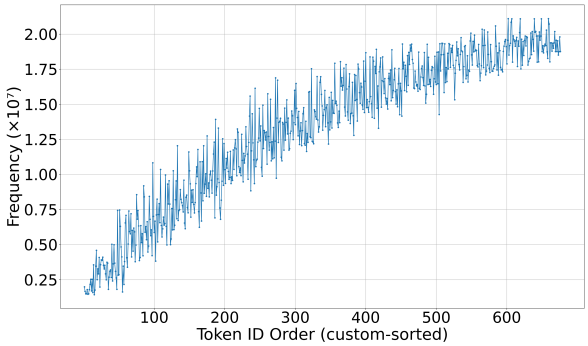

Figure 7: Relationship between merge rule strength and the frequency of occurrences at subword boundaries.

leading whitespace and the third character with a leading whitespace achieve high accuracies are described in Appendix B.

5.1.3 Additional Analysis

We further analyzed the relationship between the strength of merge rules and the frequency of subword boundaries. Under controlled merge rules, certain letter combinations are expected to appear more frequently within the same subword, whereas others are more likely to be separated at subword boundaries. For example, when the letter pair "ab" is frequently merged by the rules, many subwords containing "ab" are generated, while letter pairs with weaker merge strength tend to be split at subword boundaries.

To test this hypothesis, we aggregated the frequencies of pairs consisting of the final character of each subword and the initial character of the subsequent subword from the tokenized CharPert dataset, and analyzed their correlation with the ordering of merge strengths in the merge rules.

The results are shown in Figure 7. We observed that subwords with weaker merge strength, that is, those with larger token ID appearing later in the merge rule table, exhibit higher frequencies of the corresponding character pairs occurring at subword boundaries. These findings empirically support the theoretical explanation that the statistical properties of merge rules directly influence the mechanism by which models acquire character-level information.

5.2 Effect of Orthographic Constraints

Natural languages impose orthographic constraints on the adjacency of substrings within words. English orthography, for example, strongly restricts within word letter sequences. Typical patterns include *ck* rarely appearing word-initially and tending to follow a short vowel (e.g., back, pocket); *dge* (e.g., badge, hedge) and *tch* (e.g., catch, kitchen) after short vowels within stems, contrasted with *ge* and *ch* elsewhere (Hayes et al., 2006). In addition, double consonants (e.g., *bb* in rabbit) exhibit frequency patterns that depend on the preceding vowel spelling and following letters, reflecting letter-combination effects beyond purely phonological correspondences (Cassar and Treiman, 1997).

	w/ Ġ	w/o Ġ	Overall
TCS	50.1	80.3	68.3
BPE	75.3	86.2	80.4

Table 6: Probing task accuracy to analyze the effects of orthographic constraints within words, comparing tokenization methods with and without Ġ.

Hence, subword sequences contain regularities induced by orthographic transition constraints that can be learned during pretraining. The first subword of a word lacks a preceding dependency, making its internal character content harder to infer. On the other hand, for subsequent subwords within a word, orthographic constraints with preceding subwords limit the range of likely patterns, making it possible to infer internal character-level information indirectly from statistical regularities.

5.2.1 Analysis Procedure

To isolate orthographic effects, we design a three-character segmentation tokenizer (TCS). This tokenizer segments words into units of three characters while preserving words of two characters or fewer as they are, thereby performing segmentation independently of the BPE merge rules. For example, "Ġenterprise," where the leading whitespace is ignored for counting, becomes "Ġent, erp, ris, e."

Subwords without a leading whitespace are influenced by orthographic transitions from the immediately preceding subword, whereas subwords with a leading whitespace are not. We therefore evaluate under conditions that include or exclude a leading whitespace to quantify orthographic effects. To align the conditions of the BPE tokenizer with those of the TCS, the evaluation data for the probing task was restricted to vocabulary items of at most three characters. We train both tokenizers and models on FineWeb.

5.2.2 Results

Table 6 reports the results. Under the whitespace condition with the TCS, the accuracy was 50.1%, corresponding to the random baseline when the influences of merge rules and orthographic con-

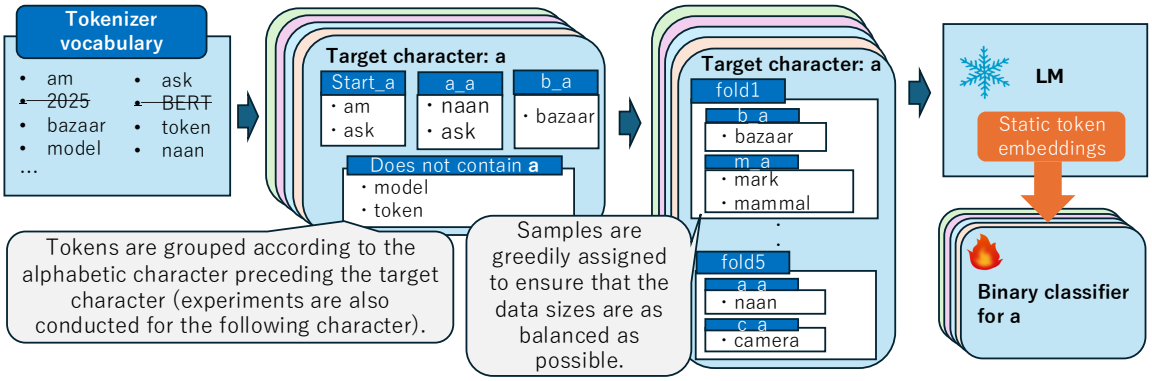

Figure 8: Evaluation Procedure for Semantic Associations of Substrings

straints were removed. In contrast, under the no whitespace condition, the accuracy reached 80.3%, revealing 30.2 point performance difference attributable to orthographic constraints within words. Under the whitespace condition with the BPE tokenizer, the accuracy was 75.3%, showing 25.2 point performance difference attributable to the effect of merge rules compared with the TCS. These results indicate that both merge rules and orthographic constraints contribute substantially to character-level acquisition.

6 Analysis of Factors Independent of Tokenization

We consider two factors independent of tokenization, namely the semantic associations of substrings and syntactic information. In this section, we train language models on the FineWeb dataset without subword segmentation and quantitatively analyze their effects on the probing task.

6.1 Semantic Associations of Substrings

Sound symbolism can induce semantic relatedness among groups of words that share similar phonological patterns. For example, words beginning with *sl* such as slip, slide, and slope are associated with the concept of sliding, while words beginning with *gr* such as grab, grasp, and grip relate to grasping. Such correspondences between spelling and meaning may place words sharing particular character bigrams close to one another in embedding space, thereby providing cues for acquiring character-level information.

6.1.1 Analysis Procedure

In the probing task for a target character α, we group tokens by their immediate context and partition the data so that training and evaluation splits contain disjoint groups, as illustrated in Figure 8. Concretely, we consider two grouping conditions, one based on the character preceding α and the other on the character following α, and measure the effect of semantic associations of substrings as the performance difference from a setting without grouping. In each setting, we partition the data into five groups with approximately equal sizes and perform five-fold cross validation.

	Accuracy	Δ from baseline
Baseline	61.2	–
Bigram (prev.)	57.3	−3.9
Bigram (next)	57.8	−3.4
Lemmatization	58.2	−3.0
Stemming	57.3	−3.9

Table 7: Effects of semantic associations and syntactic functions on character inclusion accuracy, showing performance differences from the baseline.

6.1.2 Results

The rows "Bigram (prev.)" and "Bigram (next)" in Table 7 report the results for the above grouping conditions. Grouping by the preceding character yields 57.3% accuracy, and grouping by the following character yields 57.8%, both showing an approximately 3 point decrease relative to the 61.2% baseline. These drops suggest that words sharing certain character bigrams cluster in embedding space due to semantic associations of substrings, and that bigram patterns function as cues for predicting the presence of the target character.

6.2 Effect of Syntactic Functions

Syntactic information can link particular characters to lexical or syntactic functions. For example, characters contained in affixes with syntactic functions, such as "-s" marking plural (e.g., books, cats), "-ly" marking adverbs (e.g., quickly, slowly), and "-ed"

marking past tense (e.g., walked, played), may facilitate the learning of character-level information through their functional roles. Prior work has reported that such characters with syntactic functions exhibit higher classification performance (Kaushal and Mahowald, 2022). This suggests that inflectional and derivational information are reflected in token embeddings, thereby indirectly providing cues for character-level information.

6.2.1 Analysis Procedure

To analyze the impact of syntactic information, we apply lemmatization and stemming to the pretraining corpus, thereby removing inflectional morphemes through lemmatization and both inflectional and derivational morphemes through stemming, and train models on the processed data. Lemmatization removes inflectional morphemes (e.g., -s: plural, -ed: past, -er: comparative), while stemming further removes derivational morphemes (e.g., -ly: adverbial, -ful: adjectival, -tion: nominal). We quantify the contribution of syntactic cues by comparing performance before and after these removals. We use NLTK's WordNetLemmatizer and the Porter stemmer (Miller, 1992; Porter, 1980). Following Section 3.2, we train a BPE tokenizer and nanoGPT on each processed FineWeb variant and evaluate on the probing task.

6.2.2 Results

The rows "Lemmatization" and "Stemming" in Table 7 summarize the results. Lemmatization yields 58.2% accuracy, 3.0 point decrease relative to the baseline, indicating that inflectional morphemes provide important cues for character acquisition. Stemming yields 57.3% accuracy, 3.9 point decrease, further showing that derivational morphemes also contribute.

7 Conclusion

In this study, we analyzed the factors contributing to character-level information acquisition by applying controlled transformations to input text, and divided these factors into those related to tokenization and those independent of it. Our experiments demonstrated that subword segmentation contributes through merge rules and orthographic constraints within words, while factors independent of segmentation include semantic associations of substring and syntactic information. The findings of this study suggest that LMs do not treat words as opaque tokens, but rather indirectly learn character-level patterns, which are part of their internal structure, from multiple different cues. This could help explain the spelling capabilities and the ability to generate text that leverages the sound of characters that LMs occasionally exhibit. Future work includes a more detailed analysis of the contribution of each factor identified in this study and applying these insights to develop model architectures and pre-training methods that can more robustly process character-level information.

Limitations

In this study, we have identified the main factors underlying the acquisition of character-level information; however, some aspects remain unresolved. One limitation of this work is that we have not fully controlled for the possibility that the pretraining data themselves contain lexical information. For example, the models may acquire character-level information from explicit lexical resources such as word lists or spelling rules that specify which alphabetic sequences constitute which words. Although we attempted to disentangle the factors through input transformations and tokenizer manipulations, the contribution of such lexical information has not been completely eliminated. Furthermore, the nanoGPT model used in this study is relatively small, and it remains unclear what cues contribute to character-level acquisition in larger models. In addition, prior work has reported that character-level information is more easily extracted from low frequency tokens and that rare characters tend to yield high classification performance; however, these aspects were not sufficiently examined in our analysis. Another limitation is that our experiments were conducted exclusively on English text. It remains an open question whether the mechanisms identified in this study generalize to other languages with different orthographic systems, morphological complexity, or writing scripts. Additionally, we used only FineWeb as the pretraining corpus, which may limit the generalizability of our findings. Different pretraining corpora with varying characteristics in terms of domain, genre, or text quality could potentially yield different patterns of character-level information acquisition.

Acknowledgments

This work was partly supported by JSPS KAKENHI Grant Numbers 24H00727.

References

Marie Cassar and Rebecca Treiman. 1997. The beginnings of orthographic knowledge: Children's knowledge of double letters in words. *Journal of Educational Psychology*, pages 631–644.

Cristiano Ciaccio, Marta Sartor, Alessio Miaschi, and Felice Dell'Orletta. 2025. Beyond the spelling miracle: Investigating substring awareness in character-blind language models. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (ACL 2025)*, pages 11361–11372.

A. P. Dempster, N. M. Laird, and D. B. Rubin. 1977. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–38.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (ACL 2019)*, pages 4171–4186.

Lukas Edman, Helmut Schmid, and Alexander Fraser. 2024. CUTE: Measuring llms' understanding of their tokens. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP 2024)*, pages 3017–3026.

Heather Hayes, Rebecca Treiman, and Brett Kessler. 2006. Children use vowels to help them spell consonants. *Journal of Experimental Child Psychology*, pages 27–42.

Tatsuya Hiraoka and Naoaki Okazaki. 2024. Knowledge of pretrained language models on surface information of tokens. *arXiv:2402.09808*.

Itay Itzhak and Omer Levy. 2022. Models in a spelling bee: Language models implicitly learn the character composition of tokens. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL 2022)*, pages 5061–5068.

Andrej Karpathy. 2022. nanogpt: The simplest, fastest repository for training/finetuning medium-sized gpts. https://github.com/karpathy/nanoGPT.

Ayush Kaushal and Kyle Mahowald. 2022. What do tokens know about their characters and how do they know it? In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL 2022)*, pages 2487–2507.

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 (ACL 2018), Volume 1: Long Papers*, pages 66–75.

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 (EMNLP 2018)*, pages 66–71.

George A. Miller. 1992. WordNet: A lexical database for english. In *Speech and Natural Language: Proceedings of a Workshop Held at Harriman*.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben Allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. 2024. The FineWeb datasets: Decanting the web for the finest text data at scale. *arXiv:2406.17557*.

Martin F. Porter. 1980. An algorithm for suffix stripping. *Program-electronic Library and Information Systems*.

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

Vilayanur S. Ramachandran and Edward M. Hubbard. 2001. Synaesthesia—A window into perception, thought and language. *Nature Reviews Neuroscience*, pages 43–53.

Mike Schuster and Kaisuke Nakajima. 2012. Japanese and korean voice search. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2012)*, pages 5149–5152.

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 (ACL 2016)*, pages 1715–1725.

Ian Tenney, Dipanjan Das, and Ellie Pavlick. 2019. BERT rediscovers the classical NLP pipeline. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL 2019)*, pages 4593–4601.

Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Well-read students learn better: On the importance of pre-training compact models. In *arXiv preprint arXiv:1908.08962*.

Ben Wang and Aran Komatsuzaki. 2021. GPT-J-6B: A 6 Billion Parameter Autoregressive Language Model. https://github.com/kingoflolz/mesh-transformer-jax.

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, and 12 others. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

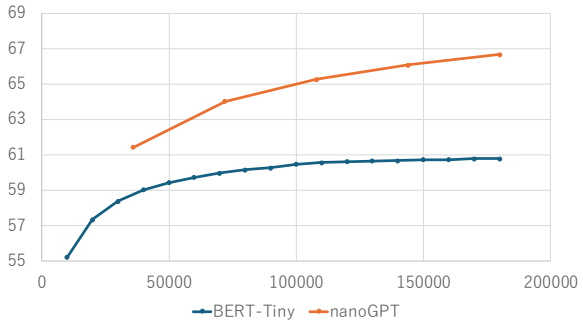

Figure 9: Relationship between pretraining data volume and accuracy on the character inclusion classification task.

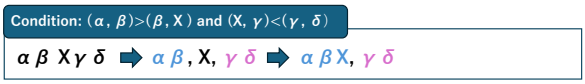

Figure 10: The case where $\alpha\beta$X$\gamma\delta$ is segmented as $\alpha\beta$X and $\gamma\delta$.

A Relationship between Training Data Size and Accuracy

In general, it is known that increases in model size and training data volume lead to improved performance on downstream tasks. In the character inclusion classification task as well, prior research and the results from Section 3 confirm that larger model sizes yield higher accuracy. On the other hand, the relationship between pretraining data volume and character-level information acquisition is not self-evident. Therefore, we measure accuracy at the intermediate checkpoints from Section 3.2 (18 checkpoints for BERT-Tiny and 5 for nanoGPT) and analyze the relationship with data volume. The results are shown in Figure 9. BERT-Tiny steadily improved from 55.2% at the first checkpoint to 60.8% at the final checkpoint, with the growth slowing in the latter half of training. nanoGPT showed a similar trend, rising from 61.4% at the first checkpoint to 66.7% at the final checkpoint. These results indicate that character-level information is acquired from the early stages of training, and its accuracy further improves as the training data volume increases.

B Considerations on Merge Rules

Why the Accuracy of the Third Character without Ġ is High Consider the case where $\alpha\beta X\gamma\delta$ is segmented as [$\alpha\beta X$, $\gamma\delta$], and Figure 10 illustrates this sequence of merge, where X is a specific alphabetic character, and α, β, γ, and δ are arbitrary alphabetic characters. First, the sequence is split into [α, β, X, γ, δ]. Next, when the ⟨2-character⟩ merge is applied, in order for the sequence to become [$\alpha\beta X$, $\gamma\delta$], it must first take the form [$\alpha\beta$, X, $\gamma\delta$]. The necessary conditions for this are m(α, β) > m(β, X) and m(X, γ) < m(γ, δ). After the segmentation [$\alpha\beta$, X, $\gamma\delta$] is obtained, the ⟨3-character⟩ merge is applied, yielding [$\alpha\beta X$, $\gamma\delta$]. In other words, the necessary condition for [$\alpha\beta X$, $\gamma\delta$] to be obtained is m(α, β) > m(β, X) and m(X, γ) < m(γ, δ), and whether $\gamma\delta$ appears depends on the choice of X. Information about which combinations are more likely is statistically encoded in the model parameters, and whether X is included in subwords is indirectly learned.

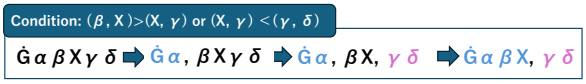

Figure 11: The case where G$\alpha\beta$X$\gamma\delta$ is segmented as G$\alpha\beta$X and $\gamma\delta$.

Why the Accuracy of the Third Character with Ġ is High Consider the case where Ġ$\alpha\beta X\gamma\delta$ is segmented as [Ġ$\alpha\beta X$, $\gamma\delta$], and Figure 11 shows this sequence of merges, where X is a specific alphabetic character, and α, β, γ, and δ are arbitrary alphabetic characters. First, the sequence is split into [Ġ, α, β, X, γ, δ]. Next, since merging begins with ⟨Ġ+1 character⟩, the sequence becomes [Ġα, β, X, γ, δ]. For the segmentation [Ġ$\alpha\beta X$, $\gamma\delta$] to be obtained, it must take the intermediate form [Ġα, βX, $\gamma\delta$]. The necessary condition for this is m(β, X) > m(X, γ) or m(X, γ) < m(γ, δ). After [Ġα, βX, $\gamma\delta$] is merged, the ⟨Ġ+3 character⟩ merge is applied, yielding [Ġ$\alpha\beta X$, $\gamma\delta$]. In other words, the necessary condition for [Ġ$\alpha\beta X$, $\gamma\delta$] to be obtained is m(β, X) > m(X, γ) or m(X, γ) < m(γ, δ). Conversely, when m(X, γ) is stronger than either m(β, X) or m(γ, δ), the segmentation [Ġ$\alpha\beta X$, $\gamma\delta$] does not occur. Thus, when [Ġ$\alpha\beta X$, $\gamma\delta$] segmentation does occur, there exist characters γ that appear only rarely depending on X, and the model learns character-level information based on these tendencies.