Surface Form Consistent BPE Tokenization

Anonymous ACL submission

Abstract

Subword tokenization methods such as Byte-Pair Encoding (BPE) underpin most modern language models for their ability to handle open-vocabulary text. However, conventional BPE conflates surface features—such as casing and whitespace—with lexical content, leading to redundant vocabulary entries and fragmented semantic representations. We introduce Surface Form Consistent BPE (SFC-BPE), a tokenization framework that disentangles surface attributes from lexical cores while preserving full reversibility. SFC-BPE maps all surface variants of a word to a single core token identifier and encodes surface information in lightweight auxiliary streams, allowing us to directly observe the modeling burden that surface forms impose. Intrinsic and extrinsic evaluations across monolingual, multilingual, and code datasets show that SFC-BPE reduces vocabulary size by up to 32% while maintaining superior compression. Models trained with SFC-BPE exhibit modest zero-shot recovery of surface features that typically require fine-tuning. Notably, when language and surface prediction are jointly optimized, the perplexity of the surface stream becomes chaotic, revealing the model's internal struggle to reconcile orthographic variation with lexical meaning. These findings show how SFC-BPE exposes the hidden cost of surface conflation in standard BPE tokenization and provides a framework for analyzing form–meaning entanglement in language models.

1 Introduction

Subword tokenization methods, such as Byte-Pair Encoding (BPE), have become standard in neural language modeling due to their effectiveness in handling open-vocabulary problems. They have proven to work well in large language models trained on many terabytes of text. Yet, from the vantage point of sample efficiency, these methods conflate surface-level information—such as casing and spacing—with the lexical content of tokens, resulting in fragmented and redundant representations. As illustrated in Figure 1, variants such as "␣HARD", "HARD", and "␣hard" are assigned distinct token IDs despite encoding the same lexical item. This detachment of different forms exacerbates the intrinsic limitations of BPE and inflates vocabulary size.

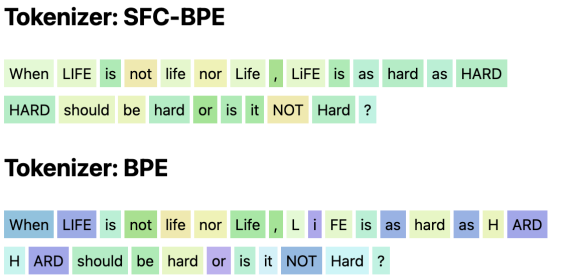


Figure 1: Comparison of SFC-BPE and BPE tokenization. SFC-BPE maps surface variants such as ␣*HARD*, *HARD*, *Hard*, and *hard* to a single lexical unit token, while BPE assigns separate tokens to each of them.

In this work, we propose a principled solution to this issue: Surface Form Consistent Byte-Pair Encoding (SFC-BPE). SFC-BPE decouples surface attributes (case, space) from the underlying character stream by encoding them in auxiliary streams—*case IDs* and *space IDs*—while maintaining a unified core representation of each lexical item. This approach enables the tokenizer to represent all surface variants of a string with a single token ID, thereby preserving lexical consistency while retaining full recoverability of surface information.

While surface information may be inconsequential in some language, they are very important in other languages and are essential to some downstream tasks. Thus, we conduct extensive experiments comparing standard BPE with SFC-BPE across several dimensions: vocabulary size, compression ratio, parity in multilingual settings, lan-

guage modeling performance, and downstream task utility (e.g., Truecasing and Named Entity Recognition).

Empirical results show that SFC-BPE reduces vocabulary size by 32% in monolingual settings (from 32k to 22k) and 28% in multilingual setups (from 32k to 23k), while maintaining higher compression rate, which reflects a better ability to model a language's genuine lexical inventory. Moreover, we demonstrate that language models trained with SFC-BPE recover surface features information with 82-84% accuracy in English, 63-65% accuracy in German, 86-87% accuracy in Java and Python.

Importantly, When language modeling and surface prediction are optimized jointly, the surface stream's perplexity shows unstable, non-convergent dynamics across training steps and languages. This oscillatory behavior suggests that conventional subword tokenization might force the model to repeatedly reallocate capacity between form and meaning. Such interference echoes the negative transfer effects reported in multitask learning (Bingel and Søgaard, 2017; Ruder, 2017) and aligns with recent findings of representational drift in large-scale language models (Liu et al., 2022; Aghajanyan et al., 2023). By separating lexical units from surface form, SFC-BPE reveals how jointly modeling surface information and lexical content poses hidden optimization struggles within modern language models.

These findings highlight the cost of ignoring linguistic structure in tokenization. Our work calls attention to the broader implications of adopting algorithmic tools from non-linguistic domains without adapting them to the structural properties of language.

Our contributions are as follows:

- We identify and formalize the problem of surface conflation in standard BPE tokenization, illustrating its effects on vocabulary size, and model training.
- We propose SFC-BPE, a novel tokenization framework that decouples surface information (case and space) from lexical content while ensuring full reversibility.
- We provide empirical evidence that SFC-BPE allows the tokenizer to include up to 32% more lexical entries within the same vocabulary budget by separating surface variants from their shared lexical core.
- SFC-BPE-trained models recover surface attributes from latent representations with accuracies of 82–84% (EN), 63–65% (DE), and 86–87% (Java/Python), confirming their learnability.

2 Background

As large language models (LLMs) scale in size and application scope, the role of tokenization has become increasingly central to both performance and efficiency (Goldman et al., 2024a; Slagle, 2024). Tokenization, typically implemented via subword segmentation determines the model's vocabulary—influences its representational granularity, and governs core aspects of memory and compute consumption during training and inference (Goldman et al., 2024a; Godey, 2024). Byte-Pair Encoding (BPE; Sennrich et al., 2016), SentencePiece (Kudo and Richardson, 2018), and their many variants have become standard for open-vocabulary modeling. Yet, these methods exhibit fundamental limitations with regard to their treatment of surface-level text variation.

One of the most overlooked inefficiencies in current tokenization approaches is due to detached surface forms, where superficial differences in case (e.g., "Life" vs. "life") and leading space (e.g., "␣life" vs. "life") result in distinct subword sequences and token IDs—despite the underlying lexical item remaining the same. For instance, the words "Hard", "hard", and "HARD" are often assigned entirely separate embeddings as demonstrated in Figure 1. Although these forms carry minimal semantic distinction for human readers, standard tokenizers fail to normalize or abstract away this surface variation.

This conflation leads to two significant issues. First, it bloats the vocabulary size by assigning multiple entries to the same underlying word. Second, it fragments semantic learning by preventing the model from associating related forms of the same lexical item. In large-scale multilingual models—where surface conventions like capitalization vary significantly across scripts, this fragmentation is amplified, degrading generalization and increasing resource demands (Bauwens and Delobelle, 2024; Brown et al., 2020).

Some approaches have sought to mitigate this issue via lossy normalization techniques, such as

lower-casing the input during preprocessing (Kudo and Richardson, 2018). While this reduces redundancy in the tokenizer's vocabulary, it also discards potentially meaningful surface information—such as case distinctions useful for discerning the degree of sentiment, named entity or crucial in style-sensitive generation (Bodapati et al., 2019). Moreover, lossy normalization breaks reversibility, making it unsuitable for high-fidelity generative tasks where surface form preservation is essential.

In contrast, our proposed framework, Surface Form Consistent BPE (SFC-BPE), addresses this problem by explicitly disentangling surface form from the core lexical representation. Rather than conflating them, SFC-BPE retains surface attributes in auxiliary encodings while maintaining a unified core token ID for all variants of a lexical item. This enables the model to benefit from both representation efficiency and surface fidelity—without resorting to lossy simplifications.

3 SFC-BPE Tokenization

The aim of this work is to develop a tokenization strategy that mitigates the inefficiencies caused by surface form conflation in subword tokenization. To this end, we propose Surface Form Consistent BPE (SFC-BPE)—a novel extension of Byte-Pair Encoding that disentangles surface features (e.g., case and space) from lexical content. SFC-BPE encodes all surface variants of a lexical item with a shared token ID, while preserving their surface form in separate, lightweight encoding streams. In this section, we formalize the encoding scheme and describe the training procedure.

3.1 Representation Scheme

In our SFC-BPE encoding scheme, each token is represented as 4-tuple (*Subword ID*, *Case Tag*, *Space Tag*, *Case Mask*), storing four pieces of information separately: an ID for the normalized form of the characters, case information, spacing information and detailed case information. Formally, to construct this, we first partition the set of Unicode characters $\mathcal{U}$ into a set of equivalence classes $\mathcal{U}/\sim$, where $\sim$ is the equivalence relation connecting upper and lower-case characters.[1] A given character[2] sequence $c_1 \cdots c_n$ is then mapped as follows:

$$c_1 \cdots c_n \longmapsto (\psi([c_1 \cdots c_n]_\sim), C, S, M)$$

Here, ψ is a mapping from subwords to numeric subword IDs, $C \in \{0, 1, 2, 3\}$ is a case tag (0: lower case; 1: upper case; 2: title case, 3: mixed), $S \in \{0, 1\}$ is a spacing flag (0: no space, 1: preceding space) and $M \in \{None, [\cdots]\}$. $[\cdots]_\sim$ denotes the equivalence class for every individual character c_i in a token according to the equivalence relation $\sim$ or to a sequence of such equivalence classes when applied to a sequence of characters as input.[3] In principle, the number of case variations for a token grows exponentially with its character length—specifically, a token of character length n has 2^n case variations (either lower or upper). However, most of these variants occur rarely, if at all, in natural text. Thus, we use case mask for full case information when a token has mixed cases otherwise, it is None[4].

This tuple encoding is used for all further processing. For instance, the "HARD" example in Figure 1 has subword ID 1205 and it is represented as (1205, Case=1, Space=0, Mask=None), "␣HARD" is represented as (1205, Case=1, Space=1, Mask=None) while "hARD" is represented as (1205, Case=3, Space=0, Mask=[0,1,1,1]). This approach collapses redundant surface variants into a single token while preserving necessary information for faithful reconstruction.

3.2 Training Procedure

The proposed training procedure described in Algorithm 1 follows the standard Byte Pair Encoding (BPE) algorithm of iteratively merging the most frequent consecutive sequence of two tokens into a single new token. However, there are several notable differences.

The training corpus is modeled using the tuple representation scheme from Section 3.1. Initially, the algorithm starts with individual characters and turns these into vocabulary entries by mapping them to their equivalence classes. The corpus is tokenized accordingly while separately recording the appropriate case information. These initial char-

[1] To ensure unambiguous reconstruction, we consider only bijective mappings and disallow any equivalences beyond the size of 2. For instance, the dotless "ı" common in languages such as Turkish and Crimean Tartar is not included as an additional lower-case form of the letter 'I'.

[2] Like BPE, our method can flexibly be applied either to original Unicode character code points or to individual UTF-8 encoding bytes. For simplicity, throughout this paper we refer to the basic units as characters. To operate on UTF-8 bytes, one considers individual bytes as the set of characters and subsequently accounts for case variations of merged sequences.

[3] When operating on UTF-8 bytes, one considers the equivalence class $[c]_\sim$ for all completed characters c, whereas any remnant bytes are left unchanged

[4] The observed variations in text are mostly lower, upper, and title case. Otherwise we apply mixed.

acters do not have any preceding space; hence the third tuple entry is 0 by default.[5]

The overall training loop adheres to the standard BPE procedure, iteratively replacing the most frequent adjacent pair with a newly minted concatenated token until no further merges are possible. In traditional BPE, the algorithm identifies the pair p exhibiting the highest frequency within the training corpus, and simply assigns a new unique identifier (ID) to this pair, merging all occurrences.

Algorithm 1: SFC-BPE Training

Input: Corpus $\mathcal{C}$, target vocabulary size $V_{\max}$
Output: Learned vocabulary V, merge rules
$V \leftarrow$ initial vocabulary with single characters $[c]_{\sim}$ for c in corpus $\mathcal{C}$
Tokenize input text $\mathcal{C}$ into initial sequences of character token tuples
$S \leftarrow \emptyset$ (empty skip list)
while *vocabulary size* $|V| < V_{\max}$ **do**
  Compute spacing-aware frequency statistics for all adjacent token pairs
  $P \leftarrow$ set of most frequent pairs not in S
  if $P = \emptyset$ *(no valid pair remains)* **then**
    break
  $p \leftarrow$ pick pair from P (breaking ties randomly)
  $T \leftarrow$ GENERATEVARIANTTUPLES(p)
  if $T = \emptyset$ **then**
    $S \leftarrow S \cup \{p\}$ (add to skip list)
    continue
  $V \leftarrow V \cup \{$ entry for canonical form of new merged pair $p\}$
  forall $t = (I, C, S) \in T$ **do**
    Identify all positions matching t in $\mathcal{C}$
    Replace occurrences in $\mathcal{C}$ with t
  Record merge operation for future decoding
return V, merge rules

In contrast, in SFC-BPE, upon identifying pair p, the algorithm relies on a GENERATEVARIANTTUPLES(p) procedure to create tuples for p and its variants. Specifically, this procedure assigns a suitable Subword ID for the merged pair and it sets the Space Tag to 1 if the first token is a space character. It then generates variants for all possible Case Tag settings. It further generates corresponding variants with the opposite of the space flag value (i.e., 1 if p is unspaced, and 0 if p is spaced), again for each possible Case Tag setting. Alternatively, the function may simply return $\emptyset$ in case a merge is deemed impossible—in particular, one can disallow prepending to already spaced tokens and one can disallow merging tokens with inconsistent casing that would not be captured faithfully by the three available case tags.

Replacing the pair of adjacent tokens with a single merged token then proceeds similarly as in BPE, however for multiple variants reflecting different surface form realizations. Additionally, to maintain dictionary efficiency and prevent bloating, the vocabulary is updated with only the Subword ID, leaving out case and space information. Note that case and spacing information are observable at the encoding phase; thus, there is no loss of information.

3.3 Encoding and Decoding

Encoding proceeds by greedily considering maximum length tokens as in BPC, while however accounting for variants with regard to case and spacing. This means that the encoder can also match variant forms that may not have been observed during training. Details of the specific case and spacing variant are recorded. Unlike conventional tokenizers that conflate surface features with semantic content into a single token, SFC-BPE explicitly disentangles these dimensions. It encodes case and space information separately from the core lexical unit, allowing each to be represented and processed independently. This structural separation requires corresponding architectural adaptations in downstream models. Specifically, models must be equipped to handle and integrate these distinct information streams during both training and inference time.

3.4 Double Implementation with equal result

Because the native SFC-BPE training procedure described above is computationally demanding, we implemented two versions: a native merge algorithm and a SentencePiece-based variant. The native approach follows the full SFC-BPE logic but trains slowly, while SentencePiece Kudo and Richardson (2018) provides a faster, optimized alternative with a fixed merge–vocabulary relationship. Despite these differences, both yield the same vocabulary compression. In the Sentence-

[5] The space character as well has no preceding space but itself is a space. We account for this in the merging and decoding logic.

Piece variant, $4,162$ of $32,000$ BPE tokens expand into multiple SFC-BPE tokens, while 14,398 of 32,000 SFC-BPE tokens expand into multiple SentencePiece BPE tokens —an approximately 32% compression, identical to the native result. This confirms that SFC-BPE's efficiency arises from its structural disentanglement of surface and lexical forms, not from implementation details. Detailed sentencepeice variant implementation is in Appendix B

4 SFC-Models

The SFC-BPE tokenization scheme maps tokens to a tuple representation with separate case and space information. Different architectural choices can be pursued to incorporate the additional elements into a model. For instance, one could pass them to the model as additional embeddings, analogously to the positional encoding in Transformer models.

Specifically, we consider SFC-BPE-based language models (SFC-Models) that has embedding representation for each surface feature, projects the feature through a linear layer and adds the feature to the core input embedding. Consequently the models have linear layers to predict surface features from the model's internal contextual representations resulting from the fusion of inputs: SFC-Models further extend standard causal language models by incorporating two auxiliary prediction heads, one for case information and another for space information. These components are trained alongside the primary language model based on the different streams that SFC-BPE provides[6]. We implemented instances of SFC-Models based on Llama-3, GPT-2, and OPT architectures, allowing us to evaluate performance across diverse model families.

5 Experiments

To evaluate the effectiveness of SFC-BPE, we conducted both intrinsic and extrinsic evaluations. Intrinsic evaluations examine tokenizer-specific properties, including vocabulary size, compression efficiency, and multilingual parity. Furthermore, we assess the ability of SFC-Models to predict case and space information from fusion of input streams. Extrinsic evaluations measure the downstream impact of SFC-BPE on language modeling (natural and programming), True casing and Named Entity Recognition. These experiments benchmark the practical utility of SFC-BPE and SFC-Models against conventional BPE-based tokenizers and models.

5.1 Experimental Setup

Datasets We used the C4 (Raffel et al., 2020) corpus to train both tokenizers and language models. Multilingual tokenizers were trained on English, French, German, Spanish, and Turkish subsets. For code generation, we used the Java and Python splits of CodeParrot-clean (Face, 2021). For True casing, we fine-tuned BPE Models on English and German WNT16 (Bojar et al., 2016) dataset. For NER, we finetuned the model families on conll2003 (Tjong Kim Sang and De Meulder, 2003) dataset for English.

Tokenizer Training We trained standard BPE and SFC-BPE tokenizers in two settings: monolingual (English) and multilingual (English, French, German, Spanish, Turkish). We also trained on programming languages Python and Java. Multilingual tokenizers were used only for intrinsic evaluations. The natural and programming language tokenizers were trained on 10000 documents each from C4 and CodeParrot-clean, respectively.[7] Detailed Tokenizer training configuration are in Appendix B

Hyperparameters	Train	Finetune
Batch size	128	128
Max. seq. length	512	512
Gradient acc.	4	4
LR schedule	Cosine	Constant
Optimization	AdamW	AdamW
LR	5×10^{-4}	5×10^{-4}
d_model	256	256
d_ff	1024	1024
n_layer/heads	6/4	6/4
Dropout	0.1	0.1
Max. iterations	-	50k

Table 1: Training and fine-tuning hyperparameters. All models were exposed to 100M tokens while training. Small language sizes were trained to prevent scale from obscuring impact.

Hyperparameters The training and fine-tuning hyperparameters are summarized in Table 1.

[6]Note that the model does not predict Case Masks when Case ID is mixed.

[7]See Appendix C for discussion of (Goldman et al., 2024a) on supporting document for training BPE.

Language	Parity (SFC)	Parity (BPE)	Fertility (SFC)	Fertility (BPE)
English	1.00	1.00	**1.55**	1.56
German	1.16	1.20	**1.76**	1.83
French	1.25	1.25	**1.63**	1.65
Spanish	1.24	1.23	**1.56**	**1.56**
Turkish	1.05	1.06	**2.03**	2.07

Table 2: Comparison of parity and fertility (token-to-word ratio) for SFC-BPE and BPE across five languages. Lower fertility indicates better compression; parity closer to 1 indicates equal token fertility relative to English. Fertility and compression are computed on the Flores dataset to multi-evaluation. SFC and BPE trained on native procedure. Fertility is normalized compression.

Language Modeling and Training Loss For experiments on English, German, Python and Java language modeling, we trained GPT-2 (Brown et al., 2020), Llama-3 (Grattafiori et al., 2024), and OPT (Zhang et al., 2022) models, along with their corresponding SFC-Model variants, each configured with approximately 30 million parameters. This is to prevent model scale from obscuring the struggles of the models. Since SFC-Models have three channels, they were trained on the summation of all losses.

6 Experimental Results

We report the results of the experiments to evaluate both the intrinsic and extrinsic properties of SFC-BPE and SFC-Models.

6.1 Intrinsic Evaluations

In standard byte-pair encoding (BPE), the vocabulary size equals the sum of merge operations and the initial 256 UTF-8 byte symbols. By explicitly separating surface-level variations from lexical content, SFC-Models modifies this relationship, yielding fewer subword IDs. Across both the native and SentencePiece-based implementations, 4,162 SentencePiece tokens expand into multiple SFC-BPE tokens, while 14,398 SFC-BPE tokens expand into multiple SentencePiece tokens—a net difference of 10,236 tokens, or about 32% of the vocabulary. This reduction, consistent with the drop from 33,000 to 22,396 subword IDs in the native training procedure, confirms that the observed compression stems from SFC-BPE's structural disentanglement of surface and lexical forms rather than implementation details.

Token Fertility Fertility reflects the average number of vocabulary items necessary to tokenize words. A fertility as low as 1 would correspond to full coverage of each word. We computed the token fertility of each tokenizer using 200 documents from the validation (dev.) set of FLORES (NLLB-Team et al., 2022). The results in Table 2 reveal that separating the lexical unit from surface forms allows tokenizers to achieve lower fertility rates despite utilizing smaller vocabulary sizes.

Multilingual Parity Parity measures how consistently a tokenizer encodes semantically equivalent sentences across languages. A tokenizer T is said to achieve parity for language A with respect to language B when the ratio of tokens used to encode equivalent sentences in A and B approaches 1. A value closer to 1 indicates fairer cross-lingual treatment. In our evaluation, we use English as the reference language. As shown in Table 2, SFC-BPE achieves closer-to-1 parity in 3 out of 4 languages, despite operating with a smaller vocabulary.

6.2 Extrinsic Evaluation

We trained three language model families namely, Llama3, OPT and GPT2 on English, German, Python and Java. We further finetuned the natural language models for Named Entity recognition and True casing.

Comparable Perplexity For fair perplexity comparision, we adopted Bit per character (Perplexity per character) (Mielke and Eisner, 2020) and ensure equal vocabulary size (naturally imposed by SentencePiece implementation). We report core perplexity per character (BPC) of OPT model on all languages in Figure 3. We computed the BPC of SFC-BPE according as

$$P_{\text{unified}} = \frac{P_{\text{core}} \times P_{\text{case}} \times P_{\text{space}}}{N_{\text{char}}}$$

. It should be noted that different model families exhibited distinct behaviors. Full graphs are in Appendix D

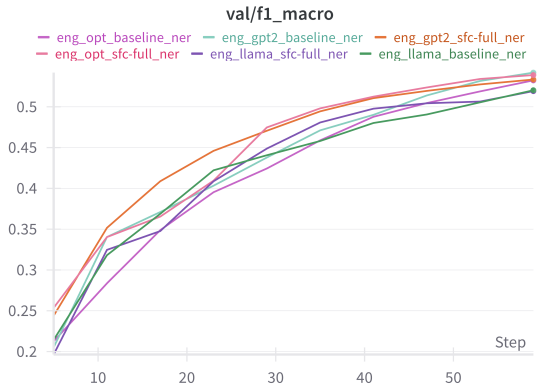


Figure 2: Validation F1-macro scores on the NER task across model families. All models show steady improvement over training. SFC-BPE-trained models perform on par with BPE baselines, particularly in OPT and GPT-2, indicating improved lexical consistency without loss of downstream accuracy.

Model	BPE Truecase Acc.	SFC Case Acc. (Joint)
English (OPT)	0.96	0.83
German (OPT)	0.93	0.67
Python (OPT)	–	0.87
Java (OPT)	–	0.88

Table 3: Comparison of case sensitivity performance between BPE and SFC models. BPE results are from task-specific finetuning, while SFC models perform zero-shot casing prediction during joint training. Programming languages (Python, Java) exhibit consistently high case accuracy.

Named Entity Recognition Across all model families in, the validation F1-macro scores on the NER task show a steady upward trend with training progression, indicating consistent learning stability. Models trained with SFC-BPE perform comparably with their BPE counterparts throughout training. Notably, the OPT and GPT-2 variants with SFC-BPE reach higher F1 scores earlier and maintain a modest advantage in later stages, suggesting improved lexical consistency benefits entity boundary detection. LLaMA-based models exhibit similar convergence for both tokenization methods, reflecting architectural robustness to tokenization differences. Overall, these results indicate that SFC-BPE preserves downstream task performance while offering structural efficiency at the tokenizer level.

Case Sensitivity and Truecasing Performance To evaluate how well models capture surface-level information, we compared truecasing accuracy in BPE-trained models (finetuned for the task) with casing accuracy in SFC-trained models (jointly trained for language modeling and surface prediction). At test time, SFC models were evaluated zero-shot on datasets different from pretraining data and exhibited results consistent with their validation behavior.

While BPE models achieve higher accuracy when explicitly finetuned for truecasing, SFC-BPE models demonstrate emergent zero-shot recovery of casing with competitive performance, despite never being trained specifically for the task. The gap between English and German reflects language-specific casing complexity rather than tokenization weakness, whereas programming languages such as Python and Java show consistently high casing accuracy (0.87–0.88) due to their syntactic regularity and deterministic capitalization rules. Notably, the stable zero-shot transfer of SFC-BPE models across both natural and programming languages underscores their ability to encode surface-level regularities implicitly within latent representations, without additional supervision.

7 Discussion

German and Java While SFC-BPE performs strongly in Java, where camelCase patterns reinforce lexical boundaries, it shows weaker results in German (63–65% accuracy), where capitalization encodes grammatical function rather than style. This contrast underscores a key limitation of surface disentanglement: it excels when orthography is stylistic or redundant but may underperform in languages where casing carries morphological or syntactic meaning.

Joint Modeling of Lexical and Surface Streams Jointly optimizing language modeling and surface reconstruction reveals a distinctive instability across training. As seen in Figures 4, the surface-stream loss and perplexity fluctuate irregularly rather than converging smoothly—an effect consistent across Java and Python despite their syntactic rigidity. This suggests an architectural tension between learning meaning and recovering form.

The observed oscillations mirror the negative transfer reported in multitask learning (Bingel and Søgaard, 2017; Ruder, 2017) and the representational drift seen in large models under multi-objective fine-tuning (Liu et al., 2022; Aghajanyan et al., 2023). In essence, when surface features and lexical content are modeled jointly, the network must repeatedly reallocate capacity between seman-

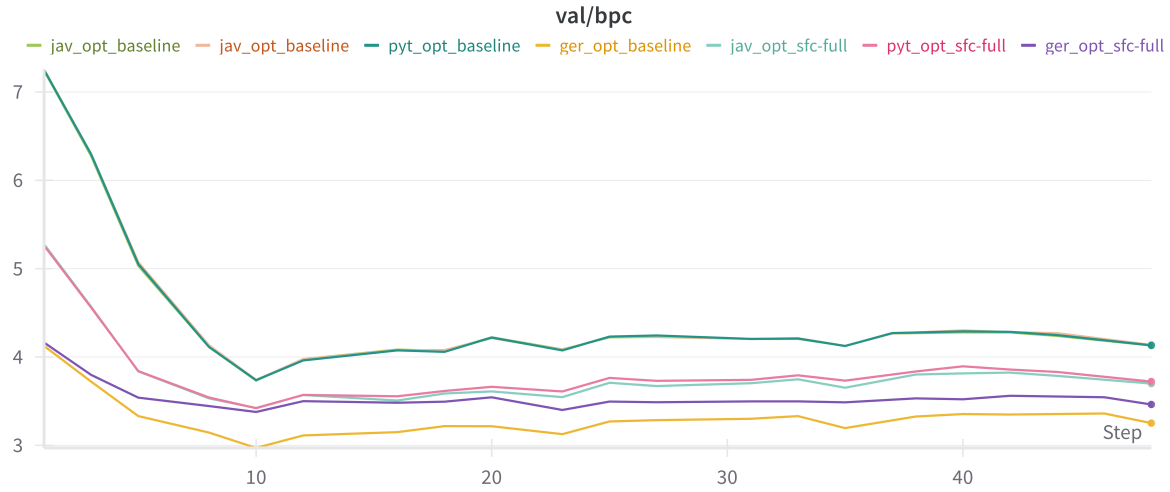


Figure 3: Validation perplexity per character (BPC) for OPT models across languages.

tic prediction and orthographic reconstruction—a process that standard BPE tokenization obscures.

By disentangling lexical units from surface forms, SFC-BPE exposes this hidden optimization burden, transforming what was once internal turbulence into a measurable signal. Remarkably, despite this instability, SFC-BPE models exhibit robust zero-shot surface recovery—82–84% accuracy in English, 63–65% in German, and up to 87% in Java and Python—while achieving a consistent 32% vocabulary compression. Together, these results reveal that surface form is not mere noise but a structured signal: when modeled explicitly yet separately, it illuminates how language models negotiate the balance between efficiency, representation, and generalization.

8 Related Work

Tokenization has re-emerged as a core concern in language modeling due to its effects on efficiency, generalization, and alignment. Bostrom and Durrett (2023) and Mielke et al. (2021) argue that tokenization is more than compression—it is a learned segmentation function that encodes inductive biases. Rust et al. (2021) show that linguistically informed segmentation improves multilingual performance. Recent work has also challenged compression-centric assumptions: Goldman et al. (2024b) find that compression correlates with downstream performance, while Schmidt et al. (2024) emphasize the role of vocabulary construction and pre-tokenization. Ali et al. (2023) highlight that tokenizer choice can lead to up to 68% variation in LLM training compute, underscoring its practical impact.

Our work complements these findings by incorporating simple surface-form cues (e.g., casing and spacing) and demonstrating that models trained with such heuristics require significantly less compute to attain equivalent task performance. This aligns with Liu et al. (2023), who show that expression-aware tokenization improves math reasoning, and Zhou et al. (2024), who demonstrate that surface form variation alone affects problem solvability in LLMs.

9 Conclusion

We presented SFC-BPE, a surface form–aware tokenization framework that disentangles lexical identity from orthographic variation while preserving full reversibility. Across natural and programming languages, SFC-BPE achieved 32% vocabulary compression and comparable or lower training loss than standard BPE. Joint modeling of language and surface streams revealed hidden optimization struggles that conventional tokenization conceals, while SFC-BPE models exhibited robust zero-shot recovery of surface features. These results show that separating form from meaning not only improves vocabulary efficiency but also offers a clearer view of how tokenization shapes learning dynamics in language models.

Limitations

While SFC-BPE presents a promising strategy for disentangling surface form from lexical represen-

tation, several limitations merit discussion. First, the framework introduces auxiliary streams (for casing and spacing) that, while lightweight, require modest architectural adaptations to standard language modeling pipelines. This may limit immediate compatibility with existing pretrained models and off-the-shelf infrastructure.

Second, although we evaluate SFC-BPE across both natural and programming languages, our current experiments remain limited in scope. Languages with highly functional capitalization (e.g., German nouns) or irregular orthographic systems may present additional challenges, particularly where surface cues carry strong grammatical or semantic weight. Moreover, our assumption that surface information can be reliably recovered from context may not hold in low-resource or morphologically complex languages, warranting broader typological evaluation.

Third, our downstream evaluation focuses primarily on language modeling and NER, leaving open how SFC-BPE interacts with other generation or reasoning tasks such as machine translation, summarization, or question answering. We view this as a key direction for future work, especially to explore whether the disentanglement of surface and lexical streams benefits transfer learning or multilingual fine-tuning.

Finally, our analysis shows that jointly modeling language and surface streams can lead to training instability and oscillatory perplexity in the surface channel—indicating interference between orthographic and semantic objectives. While SFC-BPE makes this burden measurable, future research should investigate stabilization strategies or auxiliary loss scheduling to mitigate such effects.

References

Armen Aghajanyan, Sonal Gupta, and Luke Zettlemoyer. 2023. Scaling laws for fine-tuning language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 9980–9995.

Mehdi Ali, Max Fromm, et al. 2023. Tokenizer choice for LLM training: Negligible or crucial? *arXiv preprint arXiv:2310.08754*.

Thomas Bauwens and Pieter Delobelle. 2024. BPE-knockout: Pruning pre-existing BPE tokenisers with backwards-compatible morphological semi-supervision. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 5810–5832, Mexico City, Mexico. Association for Computational Linguistics.

Joachim Bingel and Anders Søgaard. 2017. Identifying beneficial task relations for multi-task learning in deep neural networks. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pages 164–169.

Sravan Bodapati, Hyokun Yun, and Yaser Al-Onaizan. 2019. Robustness to capitalization errors in named entity recognition. In *Proceedings of the 5th Workshop on Noisy User-generated Text (W-NUT 2019)*, pages 237–242, Hong Kong, China. Association for Computational Linguistics.

Ond rej Bojar, Rajen Chatterjee, Christian Federmann, Yvette Graham, Barry Haddow, Matthias Huck, Antonio Jimeno Yepes, Philipp Koehn, Varvara Logacheva, Christof Monz, Matteo Negri, Aurelie Neveol, Mariana Neves, Martin Popel, Matt Post, Raphael Rubino, Carolina Scarton, Lucia Specia, Marco Turchi, Karin Verspoor, and Marcos Zampieri. 2016. Findings of the 2016 conference on machine translation. In *Proceedings of the First Conference on Machine Translation*, pages 131–198, Berlin, Germany. Association for Computational Linguistics.

Kenzie Bostrom and Greg Durrett. 2023. Exploring the limits of tokenization invariance in pre-trained models. In *Findings of ACL*.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. *Preprint*, arXiv:2005.14165.

Hugging Face. 2021. codeparrot/codeparrot-clean. `https://huggingface.co/datasets/codeparrot/codeparrot-clean`. Accessed: 2025-05-16.

Nathan Godey. 2024. *Improving Representations for Language Modeling*. Theses, Sorbonne Universite.

Omer Goldman, Avi Caciularu, Matan Eyal, Kris Cao, Idan Szpektor, and Reut Tsarfaty. 2024a. Unpacking tokenization: Evaluating text compression and its correlation with model performance. *Preprint*, arXiv:2403.06265.

Omer Goldman, Avi Caciularu, Matan Eyal, Kris Cao, Idan Szpektor, and Reut Tsarfaty. 2024b. Unpacking tokenization: Evaluating text compression and its correlation with model performance. *arXiv preprint arXiv:2403.06265*.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, Danny Wyatt, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Francisco Guzmán, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Govind Thattai, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jack Zhang, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Karthik Prasad, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Kushal Lakhotia, Lauren Rantala-Yeary, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Maria Tsimpoukelli, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Ning Zhang, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohan Maheswari, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vítor Albiero, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaofang Wang, Xiaoqing Ellen Tan, Xide Xia, Xinfeng Xie, Xuchao Jia, Xuewei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh, Aayushi Srivastava, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Amos Teo, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Dong, Annie Franco, Anuj Goyal, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Ce Liu, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Cynthia Gao, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkang Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Eric-Tuan Le, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Filippos Kokkinos, Firat Ozgenel, Francesco Caggioni, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Grant Herman, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan, Hakan Inan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Hongyuan Zhan, Ibrahim Damlaj, Igor Molybog, Igor Tufanov, Ilias Leontiadis, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James Kohli, Janice Lam, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U, Karan Saxena, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kiran Jagadeesh, Kun Huang, Kunal Chawla, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabsa, Manav Avalani, Manish Bhatt, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov,

Maya Lathi, Meghan Keneally, Miao Liu, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikhil Mehta, Nikolay Pavlovich Laptev, Ning Dong, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Rangaprabhu Parthasarathy, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Russ Howes, Ruty Rinott, Sachin Mehta, Sachin Siby, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Mahajan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shishir Patil, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Summer Deng, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Koehler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaojian Wu, Xiaolan Wang, Xilun Wu, Xinbo Gao, Yaniv Kleinman, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yu Zhao, Yuchen Hao, Yundi Qian, Yunlu Li, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, Zhiwei Zhao, and Zhiyu Ma. 2024. The Llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Taku Kudo and John Richardson. 2018. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. *Preprint*, arXiv:1808.06226.

Hao Liu, Tao Yu, Zihan Wang, Xing Xie, and Graham Neubig. 2022. Understanding the representational drift of continual learning in language models. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 14215–14228.

Mengzhou Liu, Young Jin Kim, Luke Zettlemoyer, and Hannaneh Hajishirzi. 2023. Expression-aware tokenization for mathematical reasoning. In *ACL*.

Sabrina J Mielke, Julian Eisenschlos, and Sharon Goldwater. 2021. Between words and characters: A brief history of open-vocabulary modeling and tokenization in nlp. In *NAACL*.

Sabrina J. Mielke and Jason Eisner. 2020. Spell once, summon anywhere: A two-level open-vocabulary language model. *Preprint*, arXiv:1804.08205.

NLLB-Team, Marta R. Costa-jussà, James Cross, Maha Elbayad, Angela Fan, Cynthia Gao, Naman Guevara, Kenneth Heafield, Kevin Heffernan, Virginia Kalb, et al. 2022. FLORES-200: Evaluating machine translation in 200 languages. *arXiv preprint arXiv:2207.04672*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text Transformer. In *Proceedings of the 37th International Conference on Machine Learning*. PMLR.

Sebastian Ruder. 2017. An overview of multi-task learning in deep neural networks. In *arXiv preprint arXiv:1706.05098*.

Phillip Rust, Jonas Pfeiffer, Ivan Vulić, Sebastian Ruder, and Iryna Gurevych. 2021. A good tokenizer is what you need: Multilingual tokenization for translation and beyond. In *EMNLP*.

Craig W. Schmidt, Varshini Reddy, Haoran Zhang, Alec Alameddine, Omri Uzan, Yuval Pinter, and Chris Tanner. 2024. Tokenization is more than compression. *arXiv preprint arXiv:2402.18376*.

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.

Kevin Slagle. 2024. SpaceByte: Towards deleting tokenization from large language modeling. *Preprint*, arXiv:2404.14408.

Erik F. Tjong Kim Sang and Fien De Meulder. 2003. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, pages 142–147.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Sid Dewan, Mona Diab, Angela Fan, Faisal Koura, et al. 2022. OPT: open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*.

Yue Zhou, Yada Zhu, Diego Antognini, Yoon Kim, and Yang Zhang. 2024. Paraphrase and solve: Exploring and exploiting the impact of surface form on mathematical reasoning in large language models. In *NAACL*.

A Joint Training

The joint optimization of language modeling and surface reconstruction exposes a striking dynamic instability across training steps. As shown in Figures 4, both the space loss and space perplexity fluctuate irregularly rather than converging smoothly. This instability persists across programming languages (Java and Python) despite their rigid lexical and syntactic structures, indicating that the effect is architectural rather than dataset-specific.

Such oscillatory behavior reflects the underlying interference between lexical and orthographic objectives. When both are optimized jointly, the model must continuously reallocate representational capacity between predicting semantic continuations and reconstructing surface attributes such as casing and spacing. This trade-off parallels the negative transfer phenomena described in multitask learning and the representational drift observed in large-scale models under cross-objective fine-tuning In linguistic terms, it highlights the non-independent yet imperfectly aligned nature of form and meaning, which becomes especially pronounced when modeled under a shared objective.

By disentangling lexical units from surface forms, SFC-BPE makes this burden explicit and measurable. The chaotic fluctuations in the surface stream are not a failure of learning but rather a revealing signature of how standard BPE tokenization forces models to internally resolve orthographic redundancy. Interestingly, despite this instability, SFC-BPE-trained models exhibit robust zero-shot recovery of surface information—achieving 82–84% accuracy in English, 63–65% in German, and up to 87% in Python and Java. This emergent capability suggests that while surface modeling introduces optimization conflict, it also leads to latent alignment between lexical and orthographic representations.

Together, these results form a consistent picture: SFC-BPE compresses redundant vocabulary structure by approximately 32%, exposes the hidden optimization struggles imposed by surface conflation, and still enables models to recover surface features with competitive accuracy—even without explicit supervision. This points to a deeper insight: surface form is not noise but an auxiliary structure that, when properly disentangled, enhances our understanding of how language models balance efficiency, representation, and generalization.

B Sentencepiece Implementation

A Implementation Details of the SFC-BPE Tokenizer

A.1 Overview

The **Surface-Form Consistent Byte Pair Encoding (SFC-BPE)** tokenizer was implemented as a modular, multilingual system designed to disentangle three sources of surface variability in text: (1) lexical content, (2) whitespace structure, and (3) letter casing. The tokenizer ensures *lossless round-trip encoding and decoding* across both natural and programming languages. It builds upon the SentencePiece BPE framework but re-structures its training and encoding stages to preserve explicit surface information while learning subword merges over normalized lexical forms.

A.2 Core Architecture

SFC-BPE consists of three primary components.

Core Lexical Model (PureBPE). A lightweight wrapper around SentencePiece (SPM) trained with the configuration below:

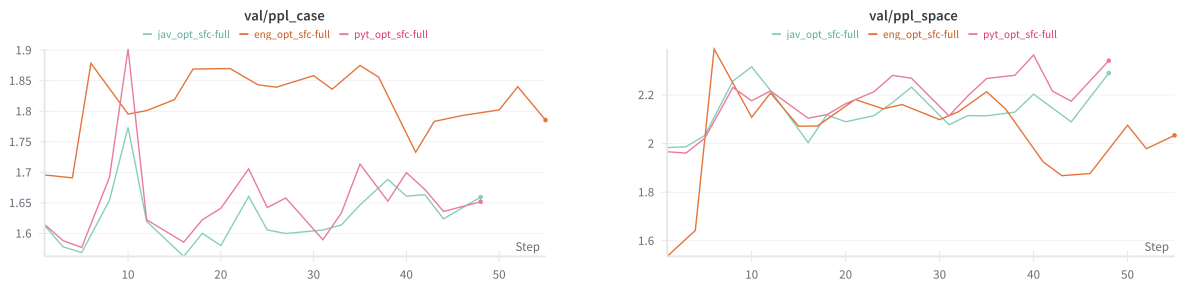
```
spm.SentencePieceTrainer.train(
    input=input_file,
    model_prefix=self.model_prefix,
    vocab_size=self.vocab_size,
    model_type="bpe",
    normalization_rule_name="identity",
    remove_extra_whitespaces=True,
    split_by_whitespace=True,
    hard_vocab_limit=False
)
```

This `PureBPE` module learns merge operations exclusively on contiguous word-like sequences. Whitespace is excluded at this stage to avoid conflating lexical merges with spacing patterns.

Whitespace Model (WhitespaceBPE). Whitespace sequences are modeled by a separate merge-based compressor that learns frequent indentation and spacing patterns. The merges are stored as a small JSON file:

```
{
 "merges": [" ", "  ", "    ", "\t", "\n "]
}
```

During encoding, all whitespace spans are replaced by compact identifiers from this model, allowing the tokenizer to represent multi-space or indentation sequences with a single token. This is



(a) Case perplexity across training steps.

(b) Space perplexity across training steps.

Figure 4: Joint modeling of language and surface streams shows unstable dynamics in the surface channel. Both loss and perplexity fluctuate irregularly, revealing optimization interference between lexical and orthographic objectives.

particularly beneficial for programming languages (e.g., Python) where indentation carries syntactic meaning.

Case Handling. Letter casing is handled via two functions: `detect_case()` and `apply_case()`. The former identifies case categories (`LOWER`, `UPPER`, `TITLE`, or `MIXED`) while the latter restores them during decoding. For mixed-case tokens, a bitmask preserves per-character casing information, ensuring exact reconstruction. Each token in the sequence therefore has aligned identifiers:

- `core_ids` — lexical subwords from `PureBPE`;
- `case_ids` — case category or bitmask;
- `space_ids` — whitespace tokens from `WhitespaceBPE`.

A.3 Training Procedure

Each language corpus (10,000 documents) was pre-processed into plain text with one sentence or code block per line. For every language—English, German, Spanish, French, Turkish, Java, and Python—two models were trained with vocabulary sizes of 8k, 16k, and 32k: (a) a baseline SentencePiece BPE tokenizer, and (b) the proposed SFC-BPE tokenizer.

The SFC-BPE training function jointly trains the core and whitespace models:

```
SFCBPETokenizer.train(corpus=[train_file],
core_vocab=vocab_size)
```

All artifacts are saved under a single directory:

```
core.model
core.vocab
ws_bpe.json
config.json
```

A.4 Encoding and Decoding

The SFC-BPE tokenizer's `encode()` method produces a structured dictionary:

```
{
  "sequence": [core_ids],
  "trailing_ws": "   "
}
```

Decoding reconstructs the original text by: (1) reinserting whitespace tokens from `WhitespaceBPE`, (2) restoring letter case via `apply_case()`, and (3) concatenating all elements in surface order. This guarantees lossless reconstruction:

`decode(encode(text)) == text.`

A.5 Design Rationale

By structurally separating lexical, case, and whitespace channels, SFC-BPE:

- reduces redundant vocabulary slots caused by case and spacing variants;
- improves token parity across languages;
- maintains perfect round-trip fidelity; and
- compresses code indentation efficiently (average one token per indentation span).

These properties enable a fair comparison between SFC-BPE and baseline SentencePiece BPE, isolating the effect of surface-form consistency on downstream performance.

A.6 Baseline SentencePiece Tokenizer

For fair comparison, a baseline **SentencePiece BPE** tokenizer was implemented to match the linguistic segmentation granularity of SFC-BPE while

omitting its case and whitespace modules. To isolate the impact of surface-form consistency, all corpora were first pretokenized using a shared regular expression pattern that separates words, contractions, punctuation, digits, and whitespace:

```
PATTERN = re.compile(
    r"""
      '(?i:[sdmt]|ll|ve|re)
    | \p{L}+
    | \p{N}{1,3}
    | [^\s\p{L}\p{N}]
    | \s+
    """,
    re.VERBOSE
)
```

Each corpus line was tokenized with this pattern and rejoined with spaces before training. This ensures both tokenizers receive identical atomic units as input, isolating differences to the tokenization algorithm itself.

Training Configuration. The `BaselineBPE` class wraps the SentencePiece trainer with parameters suited for raw-space preservation:

```
spm.SentencePieceTrainer.train(
    input=pretokenized_file,
    model_prefix=self.model_prefix,
    vocab_size=self.vocab_size,
    model_type="bpe",
    normalization_rule_name="identity",
    remove_extra_whitespaces=False,
    split_by_whitespace=True,
    hard_vocab_limit=False
)
```

This setup instructs SentencePiece to respect externally provided whitespace-delimited tokens rather than performing its own segmentation. The resulting model learns merges purely over pretokenized items, allowing a direct comparison to SFC-BPE's core channel.

Output Artifacts. Each model directory (e.g., `en_spm_8000`) contains:

```
model.model
model.vocab
train_pretok.txt
```

Behavioral Characteristics. Unlike SFC-BPE, the baseline tokenizer does not explicitly encode casing or structured whitespace. Multiple surface variants of the same word (e.g., "Tree" vs. "tree") or whitespace patterns are stored as separate entries, increasing vocabulary redundancy. However, both models operate on identical lexical boundaries, allowing differences in downstream performance to be attributed solely to SFC-BPE's structural decomposition.

B Detailed Tokenizer Training Configurations

Tokenizer	Merges	Language(s)	Corpus
BPE	32k	English	C4
SFC-BPE	32k	English	C4
BPE	33k	ML	C4
SFC-BPE	33k	ML	C4

Table 4: Training configurations for tokenizers by language and merge count. Trained using the native procedure

C Appendix: On Supporting Data for Tokenizer Compression

A critical factor in the performance of tokenization algorithms such as BPE and SFC-BPE is the composition and scale of the supporting corpus used during vocabulary learning. Recent findings by Goldman et al. (2024b) emphasize that the amount of data used to train a tokenizer significantly affects both its compression ability and the downstream performance of language models. Tokenizers trained on larger and more diverse corpora achieve better compression due to their ability to capture a richer set of statistically frequent subword units. This results in shorter sequences, smaller model input lengths, and ultimately better modeling efficiency.

In the context of **SFC-BPE**, this insight is particularly pertinent. While SFC-BPE introduces a structural refinement by disentangling surface features from lexical identity, it still builds upon core BPE vocabulary learning. The quality of this core vocabulary—how well it compresses and generalizes—depends directly on the representativeness of the support corpus.

Importantly, while SFC-BPE adds auxiliary streams for casing and spacing, the overall compression performance still hinges on the quality of the core token inventory. Our findings align with Goldman et al. (2024b), reinforcing the view that training tokenizers on large, diverse corpora improves both compression and model utility. More-

Table 5: Tokenizer models trained for all languages and vocabulary sizes. Each language was trained with both the baseline SentencePiece BPE (SP-BPE) and the proposed Surface-Form Consistent BPE (SFC-BPE).

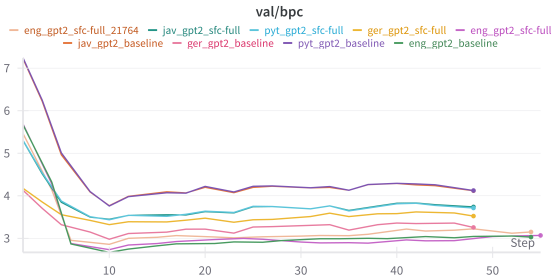
Category	Language	Vocab Sizes (k)	Tokenizers Trained
Natural Languages	English (en)	8 / 16 / 32	SP-BPE, SFC-BPE
	German (de)	8 / 16 / 32	SP-BPE, SFC-BPE
	Spanish (es)	8 / 16 / 32	SP-BPE, SFC-BPE
	French (fr)	8 / 16 / 32	SP-BPE, SFC-BPE
	Turkish (tr)	8 / 16 / 32	SP-BPE, SFC-BPE
Programming Languages	Java	8 / 16 / 32	SP-BPE, SFC-BPE
	Python	8 / 16 / 32	SP-BPE, SFC-BPE
Multilingual Models	NatLang Joint	8 / 16 / 32	SP-BPE, SFC-BPE
	ProgLang Joint	8 / 16 / 32	SP-BPE, SFC-BPE

over, SFC-BPE's compression gains over BPE are most pronounced when both are trained on equally expressive datasets—highlighting that SFC-BPE leverages structural improvements without compromising on statistical token learning.

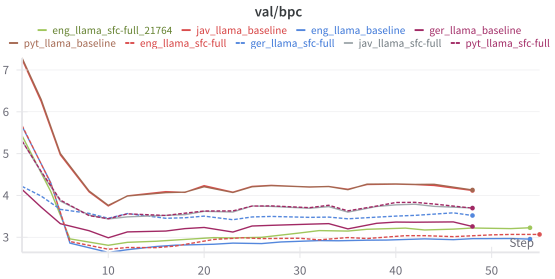
In sum, the effectiveness of any tokenizer, including structure-aware ones like SFC-BPE, is bounded by the scope and quality of the data it is trained on. Supporting corpora remain foundational to achieving competitive compression, especially when pursuing efficient and generalizable language model training.

D Full Language Modeling Report

Figure 5 compares the validation BPC trajectories of baseline and SFC-BPE-trained LLaMA and GPT2 models across languages. Across training, both variants show the expected rapid initial decline followed by gradual stabilization, indicating effective convergence. However, SFC-BPE consistently achieves slightly higher BPC in English, German, and Python.



(a) Validation BPC for GPT2 models across languages.



(b) Validation BPC for LLaMA models across languages.

Figure 5: Comparison of validation bits-per-character (BPC) across model families and languages. Both SFC-BPE and BPE variants exhibit similar convergence trends, but SFC-BPE achieves slightly higher BPC.