EMBYTE : Decomposition and Compression Learning for Small yet Private NLP

Shenglan Li* **Jia Xu*** *
Stevens Institute of Technology
Gateway Academic Center
601 Hudson St, Hoboken, NJ 07030
{sli155, jxu70}@stevens.edu

Mengjiao Zhang †
Microsoft
3940 159th Ave NE
Redmond, WA 98052
mengjiao_123@hotmail.com

Abstract

Recent breakthroughs in natural language processing (NLP) have come with escalating model sizes and computational costs, posing significant challenges for deployment in real-time and resource-constrained environments.

We introduce **EMBYTE** , a novel byte-level tokenization model that achieves substantial embedding compression while preserving NLP accuracy and enhancing privacy. At the core of **EMBYTE** is a new **Decompose-and-Compress** (**DECOMP**) learning strategy that decomposes subwords into fine-grained byte embeddings and then compresses them via neural projection. **DECOMP** enables **EMBYTE** to be shrunk down to any vocabulary size (e.g., 128 or 256), drastically reducing embedding parameter count by up to 94% compared to subword-based models without increasing sequence length or degrading performance. Moreover, **EMBYTE** is resilient to privacy threats such as gradient inversion attacks, due to its byte-level many-to-one mapping structure.

Empirical results on GLUE, machine translation, sentiment analysis, and language modeling tasks show that **EMBYTE** matches or surpasses the performance of significantly larger models, while offering improved efficiency. This makes **EMBYTE** a lightweight and generalizable NLP solution, well-suited for deployment in privacy-sensitive or low-resource environments.

1 Introduction

Recent advances in natural language processing (NLP) have yielded impressive performance across a wide range of tasks. However, these gains come at the cost of massive computational and memory requirements, which make such models expensive, slow, and impractical for deployment on resource-constrained devices. Model compression is therefore essential to enable real-time applications, expand accessibility, and reduce the environmental impact of large-scale AI systems.

To this end, various compression techniques, such as knowledge distillation (Hinton et al., 2015), quantization (Jacob et al., 2018), and parameter sharing (Lan et al., 2019), have enabled compact, efficient models with minimal accuracy loss. Notable examples like ALBERT (Lan et al., 2019), TinyBERT (Jiao et al., 2020), and DistilBERT (Sanh et al., 2019) demonstrate the success of these approaches. However, each presents trade-offs: distillation may miss subtle patterns, quantization can hinder complex reasoning, and parameter sharing may reduce model expressiveness.

Beyond architectural compression, advanced tokenization strategies, such as subword and byte-level representations (Zhang and Xu, 2022), have also improved efficiency and generalization. Subword methods (Sennrich et al., 2015) shrink vocabulary size and handle rare words effectively, especially in morphologically rich languages. Yet, they often lead to longer sequences, fragmented semantics, and reduced domain robustness. Byte-level models, with their fixed 256-symbol vocabulary, offer strong multilingual generalization and language neutrality (Zhang and Xu, 2022), but at the cost of slower training and limited semantic abstraction. Moreover, both approaches face challenges in preserving privacy and providing secure, interpretable representations, highlighting the need for continued innovation in scalable and robust NLP.

To address above challenges, we introduce a novel method, **EMBYTE** , that significantly shrinks model size and provides privacy protections simultaneously. It combines the strengths of byte-based NLP with architectural optimizations to deliver superior performance across key dimensions.

We refer to this procedure as our **Decompose-and-Compress** (**DECOMP**) learning strategy. It follows a divide-and-conquer principle: first, it decomposes subword representations into finer-grained units to reduce model size and provide privacy guarantees; then, it aggregates and com-

*Authors are listed alphabetically.
†Mengjiao Zhang was a PhD student at Stevens Institute of Technology during this work.

Findings of the Association for Computational Linguistics: EMNLP 2025, pages 7182–7201
November 4-9, 2025

Feature	Word-Based	Subword-Based	Byte-Based	EmByte-Based
Train Memory	✗Large embeddings	✓Smaller vocab	✓Tiny (256 vocab)	✓Tiny (256 vocab)
Infer Memory	✗Large softmax	✓Smaller softmax	✓Tiny softmax	✓Tiny softmax
Seq. Length	✓Short	✗Longer	✗✗Very long	✓Short
Embed Size	✗Very large	✓Smaller	✓Minimal	✓Minimal
Softmax Speed	✗Slow	✓Faster	✓Fastest	✓Fastest
Lang. Agnostic	✗No	✗Partially	✓Yes	✓Yes
Noise Robustness	✗Low	✓Moderate	✓High	✓High
Privacy	✗High risk	✓Moderate risk	✓Low risk	✓Low risk
Accuracy	✓High	✓High	✓Language generalization	✓Slightly higher

Table 1: Brief comparison of Word-, Subword-, Byte-, and our EmByte-Based NLP models.

presses these units to reduce the input length for time efficiency.

As illustrated in Figure 1, **EMBYTE** employs a neural network to project the aggregated byte-level embeddings into a reduced-dimensional space. As a result, **EMBYTE** maintains or even shortens input lengths compared to subword-based models, thereby preserving comparable training complexity. This enables us to leverage a smaller vocabulary without incurring the typical trade-off of increased input sequence length.

Specifically, the procedure consists of four steps:

1. **Decompose** each subword by byte encoding.
2. Embed each byte.
3. Aggregate byte embeddings to form the embedding for a subword.
4. **Compress** the subword embedding.

In the following, we provide further intuition on how **EMBYTE** balances efficiency, privacy, and accuracy in a unified framework.

Efficiency. As shown in Table 1, **EMBYTE** inherits the low memory footprint of byte-based models due to its fixed 256-token vocabulary and compact softmax layer, making it ideal for low-resource settings. Unlike conventional byte-level models that suffer from long input sequences, **EMBYTE** introduces a novel *decomposition-compression learning* strategy that significantly shortens sequences while retaining semantic structure. Moreover, by treating the vocabulary size as a tunable hyperparameter, the number of unique embeddings is drastically reduced, lowering the overall parameter count. To further improve computational efficiency, we project aggregated byte-level embeddings into a lower-dimensional space, maintaining input length and runtime comparable to subword-based models.

Privacy. Conventional subword models use a one-to-one mapping between tokens and embeddings, leaving them vulnerable to gradient inversion attacks (GIAs). In contrast, **EMBYTE** performs byte-level decomposition, introducing a one-to-many mapping where each subword corresponds to multiple byte embeddings. This increases ambiguity and makes it significantly harder to reconstruct original tokens from gradients, thereby improving resistance to GIAs and offering stronger privacy guarantees than both subword and byte-based baselines.

Accuracy. Despite its compact design, **EMBYTE** matches or exceeds the accuracy of larger models. The use of smaller vocabulary improves generalization and reduces overfitting. Furthermore, byte-level decomposition captures fine-grained morphological patterns, such as suffixes (e.g., `-s`, `-ed`), which subword tokenizers may fragment or overlook. This leads to richer, more flexible representations that align better with linguistic structure.

Summary. Overall, **EMBYTE** offers a compelling balance of performance, efficiency, and privacy. It outperforms traditional byte-based models in speed and accuracy, provides stronger privacy protections than subword methods, and remains language-agnostic and scalable, making it suitable for deployment in privacy-sensitive and low-resource NLP applications.

In summary, our main contributions are as follows:

1. We propose **EMBYTE** , a byte-level NLP model that decomposes text into fine-grained units through randomized byte mapping and embedding. We demonstrate that our method achieves up to a *94% reduction in embedding parameter size* compared to subword-based models, without sacrificing representational power.
2. We introduce a novel learning paradigm, **DE-**

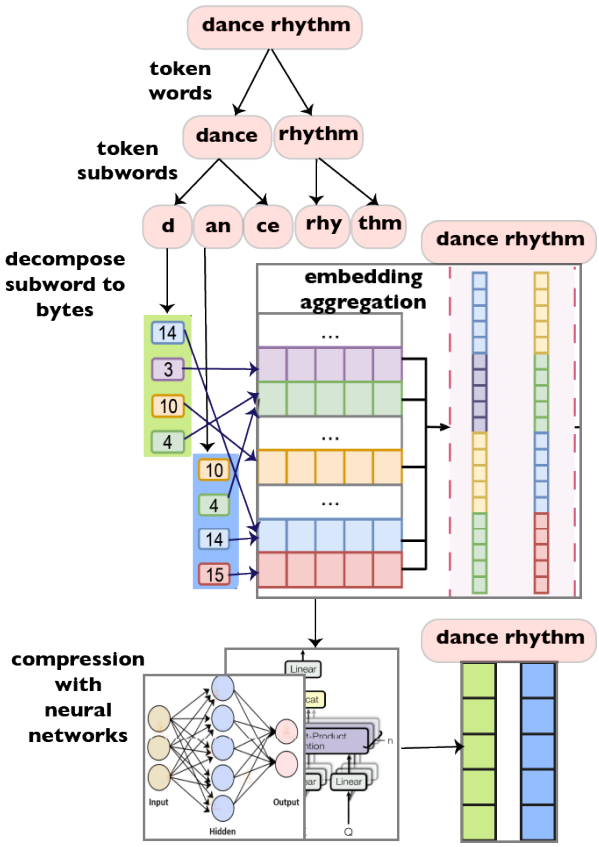


Figure 1: An example of calculating EmByte by DeComp (Decomposition and Compress).

COMP, built on a neural compression strategy that substantially *reduces input sequence length*, enabling faster *training and inference* than conventional byte-based approaches.

3. We show that our method offers strong *privacy guarantees* against gradient-based data leakage attacks, outperforming both byte- and subword-level models in federated learning scenarios.

4. Extensive experiments show that our full system consistently *improves memory efficiency and privacy while maintaining or slightly improving accuracy on five tasks*, offering a practical and scalable solution for real-world deployment.

2 Background

Subword Model Subword models address key limitations of word-level tokenization by splitting words into frequent subword units, enabling better handling of rare or unseen words and reducing vocabulary size. Methods like Byte Pair Encoding (BPE)!(Sennrich et al., 2015) and SentencePiece (Kudo and Richardson, 2018) strike a balance between character and word granularity, improving generalization and efficiency. This makes subword models, such as BERT (Devlin et al., 2019) and GPT (Brown et al., 2020), the standard in modern NLP due to their strong performance and manageable memory footprint.

Byte Model Byte-level models process text directly as sequences of bytes, removing the need for tokenizers and enabling true language agnosticism. With a fixed 256-token vocabulary, they handle rare words, noisy input, and multilingual text uniformly. Models like CANINE (Clark et al., 2022), ByT5 (Xue et al., 2022), and byte-level mBART (Tang et al., 2022) show competitive performance while improving robustness and privacy. However, they incur longer input sequences and higher computational costs due to learning from lower-level representations.

Threat Models We consider a white-box adversary aiming to recover at least one sentence from a victim's private training batch $\mathcal{B}_i$ using model parameters θ^t and gradients $\nabla_{\theta^t}\mathcal{L}_{\theta^t}(\mathcal{B}_i)$. This access enables extraction of vocabulary $\mathcal{V}$ and embedding matrix $\mathbf{W}$, allowing identification of updated tokens. Repeated attacks can reveal a significant portion of the dataset, violating federated learning privacy. We investigate two gradient leakage attacks: (1) optimization-based attacks that reconstruct inputs by minimizing the gradient difference between dummy and real data (Zhu et al., 2019; Deng et al., 2021; Wei et al., 2020); and (2) the FILM attack (Gupta et al., 2022), which extracts likely tokens from embedding gradients, generates candidate sentences via beam search with a pretrained language model, and refines them by reordering subwords. While defenses exist for the first, FILM remains largely unmitigated.

3 EMBYTE

We introduce our method, *Embedding Bytes of Subwords* (**EMBYTE**), illustrated in Figure 1. The approach converts input text into byte sequences, retrieves corresponding byte embeddings, aggregates them, and applies a low-dimensional projection to produce final subword byte embeddings.

Our objective is to develop a subword encoding scheme that leverages a compact byte-level embedding matrix to defend against gradient inversion attacks, preserve subword boundaries for time efficiency, reduce memory usage, and improve prediction accuracy. This gives rise to three key challenges: (1) how to represent subwords as byte sequences, (2) how to compute effective embeddings from these byte-level representations, and (3) how to project high-dimensional byte embed-

dings into the same low-dimensional space used for standard subword embeddings.

3.1 Decomposition of A Subword to Bytes

We encode subwords as fixed-length sequences of bytes. Formally, let $\mathcal{V}_w$ be the subword vocabulary and $\mathcal{V}_b = \{0, 1, \ldots, V_b - 1\}$ the byte vocabulary. We define a mapping $\mathcal{M} : \mathcal{V}_w \to (\mathcal{V}_b)^n$ that assigns each subword a unique sequence of n bytes. Each byte sequence is sampled uniformly at random with replacement, and we ensure uniqueness to avoid collisions. This enables a compact, fixed-size representation for all subwords. For example, with $V_b = 64$ and $n = 4$, a subword like "Hello" may be mapped to $(14, 3, 10, 4)$. The probability of collision is negligible (e.g., $< 10^{-17}$ when $V_b = 128$ and $n = 8$), ensuring expressiveness.

A key advantage of the DeComp learning strategy is that it preserves the original sequence length as perceived by the downstream model. While a subword is internally decomposed into a sequence of n bytes, the compression projector (as shown in Figure 1 and detailed in Section 3.4) aggregates these byte embeddings and outputs a single fixed-dimensional vector for each original subword. For instance, after the subword 'dance' is decomposed and its byte embeddings are aggregated and projected, it is represented by one vector. Consequently, the input to the main model (e.g., BiLSTM or Transformer) has the exact same sequence length m as in a standard subword-based model. This design avoids the significant computational overhead associated with the much longer sequences found in pure byte-level models (See Appendix B.3).

And the random mapping is intentionally chosen over semantic clustering for two key reasons: (1) it preserves privacy through unpredictable assignments that resist gradient inversion attacks, and (2) it improves performance by encouraging the model to rely on contextual information. Semantically diverse groupings prevent representational ambiguity better than clustering similar words together, as demonstrated in our ablation studies (see Section 4.1.2).

Algorithm 1 outlines the construction process for this mapping, and the resulting subword embeddings are computed by aggregating the byte embeddings. Unlike byte-level models that significantly increase input length, we encode subwords instead of characters, preserving subword boundaries and structural information while keeping byte sequences compact and efficient.

Algorithm 1 Mapping Subword-to-Byte

Input : Byte vocab $\mathcal{V}_b$, Subword vocab $\mathcal{V}_w$, Bytes per subword n
Output : Mapping $\mathcal{M} : \mathcal{V}_w \to (\mathcal{V}_b)^n$
for $w_i \in \mathcal{V}_w$ **do**
  repeat
    for $j = 1$ *to* n **do**
      Sample $b_{ij} \sim \text{Uniform}[0, V_b)$
    end
  until $(b_{i1}, \ldots, b_{in}) \notin \mathcal{M}$
  $\mathcal{M}[w_i] \leftarrow (b_{i1}, \ldots, b_{in})$
end
return $\mathcal{M}$

Algorithm 2 Embedding Bytes

Input : Subword to byte sequence mapping $\mathcal{M} : \mathcal{V}_w \to (\mathcal{V}_b)^n$;
Subword sequence $S = (w_1, w_2, \ldots, w_m)$;
A feed-forward network FFN;
Embedding matrix is $\mathbf{B} \in \mathbb{R}^{V_b \times d}$;
Subword embedding dimension d'
Output : Subword embeddings $\mathbf{E}' \in \mathbb{R}^{m \times d'}$
for $i \leftarrow 1$ *to* m **do**
  $(b_{i1}, b_{i2}, \ldots, b_{in}) \leftarrow \mathcal{M}[w_i]$
end
Byte sequence for S : $(b_{11}, \ldots, b_{1n}, \ldots, b_{m1}, \ldots, b_{mn})$
$\mathbf{E} \in \mathbb{R}^{mn \times d} \leftarrow$ retrieve $(b_{11}, \ldots, b_{1n}, \ldots, b_{m1}, \ldots, b_{mn})$ from $\mathbf{B}$
$\tilde{\mathbf{E}} \in \mathbb{R}^{m \times nd} \leftarrow$ reshape $\mathbf{E}$ (in a row-major order)
$\mathbf{E}' = \text{FFN}(\tilde{\mathbf{E}}) \in \mathbb{R}^{m \times d'}$
return $\mathbf{E}'$

3.2 Byte Embedding

Let the byte embedding matrix be $\mathbf{B} \in \mathbb{R}^{V_b \times d}$, where d is the embedding size. Given a text S, we tokenize it into a subword sequence $(w_1, w_2, \ldots, w_m)$, map it to a byte sequence $(b_{11}, \ldots, b_{mn})$ using the mapping $\mathcal{M}$, and retrieve its byte embeddings $\mathbf{E} \in \mathbb{R}^{mn \times d}$ from $\mathbf{B}$.

$$\mathcal{M}(w_1, w_2, \ldots, w_m) = (b_{11}, \ldots, b_{mn})$$

Positional Embedding To capture subword embeddings, we add the byte representations for each byte in $\mathbf{E}$. To incorporate byte positions within subwords, we concatenate positional information for a better representation of the subword structure.

3.3 Aggregation of Byte Embeddings

We aggregate byte embeddings by summing subword embeddings. Given a subword $w_i = [b_{i1}, b_{i2}, \ldots, b_{in}]$, the aggregated embedding a_i is:

$$a_i = \sum_{j=1}^{n} \mathbf{B}_{b_{ij}:}$$

where $\mathbf{B}_{b_{ij}:}$ is the b_{ij}-th element of $\mathbf{B}$, and

$a_i \in \mathbb{R}^{1\times d}$. Summing one-hot vectors or real-valued embeddings is equivalent to applying the embedding matrix to each byte and summing the results. This process aggregates byte representations into subword embeddings, as shown in Algorithm 2.

3.4 Compression

The resulting aggregated vector is then fed into our projector Π. Given the retrieved n byte embeddings $\mathbf{E} \in \mathbb{R}^{m\times d}$, we reshape $\mathbf{E}$ to $\tilde{\mathbf{E}} \in \mathbb{R}^{m\times nd}$ in a row-major order, which is equivalent to concatenation. Then, an our projector is applied to project $\tilde{\mathbf{E}}$ into the dimension d' of the original subword embedding for language models:

$$\mathbf{E}' = \Pi(\tilde{\mathbf{E}}), \tag{1}$$

where $\mathbf{E}' \in \mathbb{R}^{m\times d'}$. Note that, the byte embedding matrix $\mathbf{B}$ can be either a real-valued or one-hot embedding matrix because the vocabulary size is small for bytes.

We design the following compression models to realize $\Pi(\cdot)$ in Equation 1:

1. **FFN**: A two-layer feedforward neural network (FFN) with ReLU activation.
2. **MLP**: A Multi-Layer Perceptron (Rumelhart et al., 1986) consisting of one linear layer with ReLU and dropout applied to one-hot byte embeddings, followed by a two-layer FFN after concatenation.
3. **Autoencoder**: The encoder part of an autoencoder (Hinton and Salakhutdinov, 2006), composed of two linear layers with a bottleneck, ReLU activations, and dropout, pre-processes the concatenated byte embeddings to learn a compressed representation. This representation is then passed through a standard two-layer FFN.
4. **TF Autoencoder**: A single Transformer Encoder (Vaswani et al., 2017) layer processes the sequence of individual one-hot byte embeddings. The output sequence is then concatenated and passed through a standard two-layer FFN.

We compare the performance of these models with baselines in Table 7 in Section 4.1.2.

4 Experiments

We conduct experiments to demonstrate the advantages of EMBYTE in preserving privacy for NLP models, slightly improving on model prediction accuracy, reducing space complexity, and maintaining time efficiency. In all experiments, we set $V_b = 256$ and $n = 8$, to minimize the risk of reassigning two subwords to the same byte sequence.

Implementation and result details are deferred to the Appendix B, with section numbers corresponding (e.g., Section 4.1.1 → Appendix B.1.1).

Task	model	# Params	Acc	F_1
MRPC	Subwords	4.0M	0.684	0.812
	EmByte	1.4M	0.691	0.813
RTE	Subwords	1.9M	0.501	
	EmByte	1.0M	0.536	

Table 2: EMBYTE vs. subwords on GLUE tasks.

4.1 Experiment on Performance

To provide a more comprehensive assessment of our proposed technique, we conduct experiments on two GLUE tasks, machine translation, sentiment analysis, and Language Modeling (LM) tasks.

4.1.1 GLUE Tasks

We conduct experiments on two representative tasks from the General Language Understanding Evaluation (GLUE) benchmark (Wang et al., 2018): Microsoft Research Paraphrase Corpus (MRPC) (Dolan and Brockett, 2005) for paraphrase detection and Recognizing Textual Entailment (RTE) (Dagan et al., 2006) for grammatical acceptability.

Main Results As shown in Table 2, EMBYTE consistently achieves competitive performance across both tasks while significantly reducing the number of parameters. Notably, EMBYTE outperforms the baseline with accuracy improvements of 0.7% and 3.5% on MRPC and RTE, respectively, while using 65% and 47% fewer parameters. These results highlight EMBYTE 's ability to deliver strong language understanding performance with substantially improved parameter efficiency.

4.1.2 Sentiment Analysis

Dataset and evaluation metrics We use the IMDb (Maas et al., 2011) and SST2 (Socher et al., 2013) datasets from Hugging Face, and evaluate performance using accuracy, following standard practice in prior work (Minaee et al., 2019; Yenter and Verma, 2017). There are 25000 training samples and 25000 testing samples for IMDb. We take 25% of the training data for validation and the rest for training. For SST2, The training, validation, and test examples in SST2 are 67349, 872, and

	good	great	funny	bad	worse	boring
good	1	0.63	0.49	-0.58	-0.61	-0.58
great	0.63	1	0.40	-0.53	-0.33	-0.38
funny	0.49	0.40	1	-0.72	-0.61	-0.60
bad	-0.58	-0.53	-0.72	1	0.72	0.85
worse	-0.61	-0.33	-0.61	0.72	1	0.88
boring	-0.58	-0.38	-0.60	0.85	0.88	1

Table 3: The cosine similarity of the subword embeddings calculated based on **EMBYTE** .

	good	great	funny	bad	worse	boring
good	1	0.06	0.04	-0.01	-0.01	0.09
great	0.06	1.00	0.00	-0.08	0.00	0.01
funny	0.04	0.00	1.00	0.07	-0.01	0.00
bad	-0.01	-0.08	0.07	1.00	0.03	0.00
worse	-0.01	0.00	-0.01	0.03	1.00	0.04
boring	0.09	0.01	0.00	0.00	0.04	1.00

Table 4: The cosine similarity of baseline subword embedding.

1821, respectively. The tokenizer is "basic_english" in the TorchText package. The minimum frequency needed to include a token in the vocabulary is 5. The maximum length of the sentence is 256.

Main results Table 3 and Table 4 demonstrate that **EMBYTE** retains subword interpretability through byte embeddings, grouping similar words together and assigning negative correlations to dissimilar ones, producing semantically intuitive embeddings that align more closely with human judgment than the subword baseline. In Table 3, **EMBYTE** assigns high positive cosine similarity scores to related words like 'good' and 'great' (0.74) and strong negative scores to antonyms like 'funny' and 'boring' (-0.70). In contrast, the baseline subword embeddings (Table 4) fail to capture these relationships, assigning near-zero similarity to both synonyms (e.g., 'good' and 'great', 0.06) and antonyms. This shows that the DeComp process effectively preserves and even enhances semantic structure.

Table 5 presents classification accuracy on the SST-2 task, comparing **EMBYTE** with subword embedding, gradient compression, and noisy gradient defenses (Wainakh et al., 2022). While existing defenses offer protection against label-leakage, they substantially degrade performance. In contrast, **EMBYTE** provides strong resistance to gradient-based privacy attacks while preserving high utility, i.e., achieving higher accuracy than all baselines and doubling the preservation of the privacy of subword embeddings.

Table 6 shows the effect of varying the number

Method	SST2 (%)	IMDB (%)
Subword Embedding	79.2	85.6
Gradient Compression	53.5	48.9
Noisy Gradients	53.4	49.0
EMBYTE	84.1	85.8

Table 5: Comparison of accuracy of subword embedding and baseline defense models on sentiment analysis.

M.# Bytes	Accuracy	Train	Infer.	# Params
Subword	0.814	120	19.4	5.93
EMBYTE $_1$	0.691	132	26.3	3.57
EMBYTE $_2$	0.816	129	15.9	3.60
EMBYTE $_3$	0.832	128	28.8	3.63
EMBYTE $_4$	0.817	129	18.6	3.67
EMBYTE $_5$	0.810	116	9.8	3.70
EMBYTE $_6$	0.823	126	18.0	3.73
EMBYTE $_7$	0.808	123	27.6	3.77
EMBYTE $_8$	0.832	116	19.7	3.80

Table 6: Experiment results for the baseline subword model and various EmByte variants. M.#Bytes: Subword baseline or **EMBYTE** model with different number of bytes per subword; Train: Training time in seconds; Infer.: Inference time in miliseconds per batch. #Pars: Parameter size in million.

of bytes (n) per subword in **EMBYTE** , for **EMBYTE** $_n$ ($n = 1$ to 8) on SST-2, compared to the subword baseline. All **EMBYTE** variants reduce parameter count significantly. While **EMBYTE** $_1$ performs poorly, accuracy improves with larger n, with **EMBYTE** $_3$ and **EMBYTE** $_8$ outperforming the baseline. Training and inference times vary without a clear trend across n.

To validate our random mapping design choice, we compared it with a semantically structured variant (Clustered EMBYTE) that groups similar subwords using k-means clustering. As shown in Table 8, our random mapping consistently outperforms structured clustering across multiple benchmarks, supporting our hypothesis that random mapping allows EMBYTE to generate more expressive and information-rich representations, using the same vocabulary size.

Compressor results Table 7 summarizes the performance of various neural compression projector architectures (Section 3.4) on SST-2, compared to the Baseline and **EMBYTE** . All MLP models consist of three layers, with the second layer having 128 dimensions and the third layer 256 dimensions. We vary the input size of the first layer based on the embedding size per byte. In MLP-256, each of the eight byte embeddings has 32 dimensions, resulting in a total input of $32 \times 8 = 256$. Similarly, MLP-512 uses 64-dimensional byte embeddings ($64 \times 8 = 512$), and MLP-768 uses 96-dimensional embeddings ($96 \times 8 = 768$). In the AE-Enc models,

EMBYTE	Accuracy	Train	Infer.	# Pars
Baseline	0.797	345	24.3	5.93
FFN	0.817	374	15.9	3.80
MLP-HD256	0.823	393	7.80	4.10
MLP-HD512	0.832	383	17.8	4.65
MLP-HD768	0.824	390	16.5	5.21
AE-Enc(512-256)	0.819	387	10.3	4.75
AE-Enc(1024-128)	0.813	417	9.00	5.78
TF-Enc-H4-FFN512	0.841	569	28.4	4.33
TF-Enc-H8-FFN1024	0.822	669	39.7	4.59

Table 7: Experiment results for **EMBYTE** with different neural network for compression. Train: Training time in seconds; Infer.: Inference time in milliseconds per batch; # Pars: Parameter size in million.

Dataset	Model	Accuracy	# Pars
SST-2	Clustered **EMBYTE**	80.88	3.80
SST-2	**EMBYTE** (Random)	84.10	3.80
IMDb	Clustered **EMBYTE**	78.39	3.80
IMDb	**EMBYTE** (Random)	85.80	3.80

Table 8: Random vs. Structured Mapping Performance

the autoencoder's encoder consists of three layers: an input layer of 2048 dimensions (8×256), a second layer with 512 dimensions, and a final layer with 256 dimensions. Variants also include different configurations of the second and third layers while keeping the input fixed. Finally, TF-Enc-H-FFN refers to Transformer encoder models with varying numbers of attention heads and a single feedforward network (FFN) of a specified output dimension.

The compressor with FFN achieves 0.817 accuracy with only 3.80M params, offering fast inference and efficient training. Among MLP+FFN variants, the best (MLP-512) reaches 0.832 accuracy with 4.65M params and 383s training time. AE-Enc+FFN (dim-512→256) matches FFN with 0.819 accuracy and 4.75M params. Transformer TF-Enc+FFN (H4, FFN-512) achieves the highest accuracy of 0.841 using 4.33M params, but with significantly higher training and inference time. Overall, **EMBYTE** offers a strong balance, while more complex projectors can improve accuracy at a computational cost.

4.1.3 Machine Translation

Dataset and Evaluation Metrics We evaluate on the medium-scale IWSLT14 (de→en) (Cettolo et al., 2014) and large-scale WMT14 (en→de) (Bojar et al., 2014) datasets, following the setups of (Shaham and Levy, 2020; Zhang and Xu, 2022) with Fairseq preprocessing (Ott et al., 2019). Evaluation uses case-sensitive SacreBLEU with the 13a tokenizer (Post, 2018).

Datasets	Embeddings	# Params	BLEU
IWSLT14	Subword	5.2M	34.54 ± 0.10
	EMBYTE $_{ar}$	4.3M	34.64 ± 0.15
	EMBYTE $_{cr}$	9.6M	35.32 ± 0.15
	EMBYTE $_{co}$	5.2M	$\mathbf{35.44 \pm 0.10}$
WMT14	Subword	22.3M	26.0
	EMBYTE $_{co}$	6.3M	26.0

Table 9: BLEU score of IWSLT14 and WMT14. **EMBYTE** $_{ar}$ and **EMBYTE** $_{cr}$: **EMBYTE** with added and concatenated real-valued embeddings, respectively. **EMBYTE** $_{co}$: **EMBYTE** with concatenated one-hot byte embeddings.

	# Parameters	Perplexity
Subword	12.8M	38.65
EMBYTE $_{co}$	10.5M	37.24

Table 10: Perplexity of language modeling for subword embedding and **EMBYTE** .

Embedding	Memory	Time
Subword	$\mathcal{O}(V_w d)$	$\mathcal{O}(m^2 d)$
Byte	$\mathcal{O}(V_b d)$	$\mathcal{O}(c^2 m^2 d)$
EMBYTE (Ours)	$\mathcal{O}((nd + V_b)d)$	$\mathcal{O}(m^2 d)$

Table 11: Complexity for conventional subword embeddings, byte embedding, and our proposed **EMBYTE** .

Main Results Table 9 shows that **EMBYTE** $_{cr}$ and **EMBYTE** $_{co}$ outperform subword baselines on IWSLT14 and match them on WMT14, with **EMBYTE** $_{co}$ using fewer parameters. Concatenation yields better performance than addition for byte aggregation (Section 3.2). **EMBYTE** reduces embedding size while preserving accuracy and enhancing privacy.

4.1.4 Language Modeling

Dataset and evaluation metrics We use the same data as Fairseq did for the language modeling tasks. The dataset we use is WikiText-103. We use the same preprocessing and training settings as the official Fairseq does. The number of samples for training, testing, and validation are 1801350, 3760, and 4358 respectively. We evaluate the language modeling performance with perplexity.

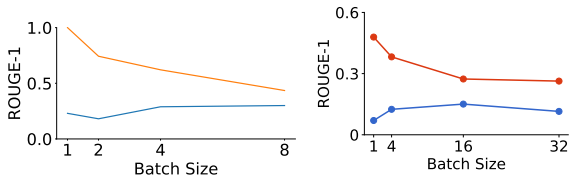
Main results For language modeling, **EMBYTE** achieves lower perplexity with fewer parameters (Table 10), demonstrating its efficiency and effectiveness over traditional subword embeddings.

4.2 Complexity Analysis and Evaluation

Analysis For the efficiency, we compare the space and time complexity of different embedding methods in Table 11. Subword embeddings require large vocabularies ($V_w > 10^4$), while byte-

# Params	Whole model	Embedding	BLEU
IWSLT14	37M	5.2M	34.54 ± 0.10
EMBYTE $_{co}$	33M ($\downarrow$ 12%)	0.7M ($\downarrow$ 94%)	34.62 ± 0.12
SST2	5.9M	2.4M	81.2 ± 0.7
EMBYTE $_{co}$	3.8M ($\downarrow$ 36%)	0.8M ($\downarrow$ 68%)	82.5 ± 0.7
IMDb	1.5M	1.5M	85.6 ± 0.5
EMBYTE $_{co}$	0.4M ($\downarrow$ 72%)	0.5M ($\downarrow$ 80%)	85.8 ± 0.2

Table 12: Subword and **EMBYTE** numbers in machine translation on IWSLT14 de-en, Sentiment analysis on SST2, and Sentiment analysis on IMDb.



(a) Pre-trained (b) Fine-tuned

Figure 2: Attack recovery performance per batch size on WikiText-103. Orange: Subword; Blue: **EMBYTE** .

based methods, including **EMBYTE** , use much smaller dictionaries ($V_b \leq 256$), reducing memory to $\mathcal{O}((nd + V_b)d)$. Unlike byte embeddings, which increase input length by a factor $c \approx 5$ (Shaham and Levy, 2020), **EMBYTE** preserves subword boundaries and thus retains the same time complexity as standard subword embeddings.

Empirical evaluation Table 12 shows model and embedding sizes on IWSLT14 and SST2. For IWSLT14 de→en, **EMBYTE** with 256 hidden units achieves comparable performance to subwords. Values in "()" indicating percentage reductions by **EMBYTE** compared to subword models. Across all tasks, **EMBYTE** $_{co}$ consistently reduces model size, enabling efficient on-device training in memory-constrained settings without sacrificing performance.

4.3 Experiments on Privacy Protection

Datasets, attack task, and evaluation metrics We followed the settings in the LLG attack (Wainakh et al., 2022) and FILM attack (Gupta et al., 2022). The dataset is WikiText-103 (Merity et al., 2016). For the attack task, we use GPT-2 base (Radford et al., 2019) with 117M parameters to recover the input batches. The ROUGE-1/2/L F-Scores (Lin, 2004) are used to evaluate the similarity between the recovered and original text.

Quantitative analysis of defense Figure 3 illustrates the defense performance in attack success rate of **EMBYTE** and subword-based models against the frequency-based Label-Leaking Gradient (LLG) attack from different level of neural network layers, the input, middle, and head layer.

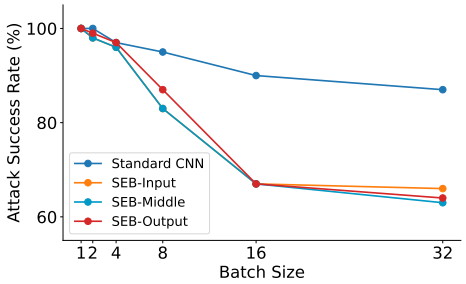


Figure 3: LLG attack defense: **EMBYTE** versus Subword from different layers: input layer, middle layer, and output head layer over various batch size.

Figure 2a shows FILM attack performance during pre-training on WikiText-103 with varying batch sizes. Due to the large subword vocabulary, we sample 7,000 random subwords plus those from the original text. ROUGE-1 scores, averaged over 5 batches, show near-perfect recovery for subword embeddings at batch size 1 (ROUGE-1 is approaching to 1), while **EMBYTE** yields much lower scores, indicating stronger defense.

Figure 2b shows that after fine-tuning with **EMBYTE** on pretrained subword model, **EMBYTE** further reduces recovery performance across batch sizes 1, 4, 16, and 32, outperforming the subword baseline.

On the other hand, gradient inversion attacks (e.g., FILM, LLG) are most effective at small batch sizes due to reduced gradient averaging. Although EMBYTE cannot eliminate all attacks at batch size 1 (the worst case), it provides substantial improvements across all batch sizes, with increasing protection as batch size grows, particularly relevant for real-world training scenarios where batch sizes typically range from 16-1024.

5 Related Work

Memory efficient embedding models We compare **EMBYTE** with several memory-efficient variants of BERT (Devlin et al., 2019), including DistilBERT (Sanh et al., 2019), TinyBERT (Jiao et al., 2020), ALBERT (Lan et al., 2020), MobileBERT (Sun et al., 2020), and Quantized BERT (Zafrir et al., 2019). While these models offer comparable efficiency and minor decrease in accuracy, they lack inherent privacy guarantees. In contrast, our model is not only compact but also designed with privacy in mind. A detailed comparison is provided in Appendix B.8.

Subword-level and byte-level models Subword tokenization like BPE (Sennrich et al., 2015) is widely used but limited by out-of-vocabulary issues, language-specific tokenizers, and large embedding matrices. Byte-level tokenization addresses these challenges (Shaham and Levy, 2020; Zhang and Xu, 2022; Xue et al., 2022) using UTF-8, which covers all languages and caps the vocabulary at 256 bytes, enabling smaller, language-agnostic embeddings. Building on this, we propose **EMBYTE** , which improves accuracy, memory and training efficiency with stronger privacy protection.

Subword-level model with character-/byte-level fusion Character and byte models increase sequence length and computational cost. To mitigate this, recent works fuse character/byte representations into subwords. CHARFORMER (Tay et al., 2021) introduces a soft tokenization mechanism that scores and aggregates subword blocks, followed by downsampling. While efficient, it lacks explicit subword boundaries, reducing interpretability. And while CHARFORMER achieves higher accuracy on some GLUE tasks, it lacks the privacy guarantees and compression benefits that are central to **EMBYTE** 's contribution. LOBEF (Sreedhar et al., 2022) preserves the word boundaries by locally fusing bytes, but does not shorten the sequences, limiting the efficiency of training and inference. Our **EMBYTE** both preserves the subword boundaries for interpretability and keep the input length for efficiency. It significantly outperforms LOBEF (26.0 vs 21.5 BLEU on WMT14) while maintaining subword-level inference speeds, whereas LOBEF is approximately twice as slow.

Gradient attacks and defenses Defenses like gradient encryption (Zhu et al., 2019; Deng et al., 2021; Balunovic et al., 2022) can be costly and ineffective against server-side leakage (Aono et al., 2017; Huang et al., 2021; Fang and Qian, 2021). Differential privacy (Zhu et al., 2019; Wei et al., 2020; Yin et al., 2021; Li et al., 2021) can degrade accuracy. While Zhang and Wang (2021) propose a secure FL framework against gradient-based attacks, it fails to prevent subword recovery from embedding gradients (Gupta et al., 2022). As shown in Table 5, **EMBYTE** outperforms defenses like gradient compression and noise gradient (Wainakh et al., 2022) on GIAs.

6 Conclusion

We present **EMBYTE** , a novel language representation method using **DECOMP** learning for efficient, privacy-preserving training. By projecting subword byte embeddings into lower dimensions, **EMBYTE** reduces model size, improves accuracy, and defends against gradient-based privacy attacks.

Acknowledgement

We gratefully acknowledge the partial support of the eBay eRUPT Program and ACC-New Jersey (Contract No. W15QKN-18-D-0040), whose invaluable support has greatly contributed to this work.

7 Limitations

While **EMBYTE** shows strong performance across diverse tasks, further evaluation on additional domains remains future work: (1) evaluation on additional domains and languages beyond those tested, (2) exploration of alternative privacy threats beyond gradient inversion attacks, and (3) investigation of learned or hybrid mapping strategies that might balance semantic structure with privacy preservation.

8 Ethical Considerations

We use AI to enhance grammar, conducting experiments exclusively on public, non-sensitive datasets. Our goal is to advance efficient and accurate NLP while promoting secure and responsible model.

References

Yoshinori Aono, Takuya Hayashi, Lihua Wang, Shiho Moriai, et al. 2017. Privacy-preserving deep learning via additively homomorphic encryption. *IEEE Transactions on Information Forensics and Security*, 13(5):1333–1345.

Mislav Balunovic, Dimitar Dimitrov, Nikola Jovanović, and Martin Vechev. 2022. Lamp: Extracting text from gradients with language model priors. *Advances in Neural Information Processing Systems*, 35:7641–7654.

Ondřej Bojar, Christian Buck, Christian Federmann, Barry Haddow, Philipp Koehn, Johannes Leveling, Christof Monz, Pavel Pecina, Matt Post, Herve Saint-Amand, Radu Soricut, Lucia Specia, and Aleš Tamchyna. 2014. Findings of the 2014 workshop on statistical machine translation. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*, pages 12–58, Baltimore, Maryland, USA. Association for Computational Linguistics.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Mauro Cettolo, Jan Niehues, Sebastian Stüker, Luisa Bentivogli, and Marcello Federico. 2014. Report on the 11th iwslt evaluation campaign. In *Proceedings of the 11th International Workshop on Spoken Language Translation: Evaluation Campaign*, pages 2–17.

Jonathan H. Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. 2022. Canine: Pre-training an efficient tokenization-free encoder for language representation. In *Transactions of the Association for Computational Linguistics (TACL)*.

Ido Dagan, Oren Glickman, and Bernardo Magnini. 2006. The pascal recognising textual entailment challenge. In *Machine Learning Challenges. Evaluating Predictive Uncertainty, Visual Object Classification, and Recognising Tectual Entailment*, pages 177–190. Springer.

Jieren Deng, Yijue Wang, Ji Li, Chenghong Wang, Chao Shang, Hang Liu, Sanguthevar Rajasekaran, and Caiwen Ding. 2021. Tag: Gradient attack on transformer-based language models. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 3600–3610.

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

William B Dolan and Chris Brockett. 2005. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing (IWP)*.

Haokun Fang and Quan Qian. 2021. Privacy preserving machine learning with homomorphic encryption and federated learning. *Future Internet*, 13(4):94.

Samyak Gupta, Yangsibo Huang, Zexuan Zhong, Tianyu Gao, Kai Li, and Danqi Chen. 2022. Recovering private text in federated learning of language models. *arXiv preprint arXiv:2205.08514*.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

Geoffrey E Hinton and Ruslan R Salakhutdinov. 2006. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507.

Yangsibo Huang, Samyak Gupta, Zhao Song, Kai Li, and Sanjeev Arora. 2021. Evaluating gradient inversion attacks and defenses in federated learning. *Advances in Neural Information Processing Systems*, 34:7232–7241.

Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2704–2713.

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2020. Tinybert: Distilling bert for natural language understanding. In *Findings of EMNLP*, pages 4163–4174.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

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

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. Albert: A lite bert for self-supervised learning of language representations. *ICLR*.

Xuechen Li, Florian Tramer, Percy Liang, and Tatsunori Hashimoto. 2021. Large language models can be strong differentially private learners. *arXiv preprint arXiv:2110.05679*.

Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain. Association for Computational Linguistics.

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. 2011. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA. Association for Computational Linguistics.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*.

Shervin Minaee, Elham Azimi, and AmirAli Abdolrashidi. 2019. Deep-sentiment: Sentiment analysis using ensemble of cnn and bi-lstm models. *arXiv preprint arXiv:1904.04206*.

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. 2019. fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of NAACL-HLT 2019: Demonstrations*.

Matt Post. 2018. A call for clarity in reporting BLEU scores. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 186–191, Belgium, Brussels. Association for Computational Linguistics.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. 1986. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. In *NeurIPS EMC2 Workshop*.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2015. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*.

Uri Shaham and Omer Levy. 2020. Neural machine translation without embeddings. *arXiv preprint arXiv:2008.09396*.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA. Association for Computational Linguistics.

Makesh Narsimhan Sreedhar, Xiangpeng Wan, Yu Cheng, and Junjie Hu. 2022. Local byte fusion for neural machine translation. *arXiv preprint arXiv:2205.11490*.

Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. 2020. Mobilebert: a compact task-agnostic bert for resource-limited devices. In *ACL*, pages 2158–2170.

Yinhan Tang, Xavier Garcia, Alessandro Raganato, Vishrav Chaudhary, and Jiatao Gu. 2022. An efficient multilingual byte-to-byte model for sequence-to-sequence tasks. In *Findings of the Association for Computational Linguistics: EMNLP 2022*.

Yi Tay, Vinh Q Tran, Sebastian Ruder, Jai Gupta, Hyung Won Chung, Dara Bahri, Zhen Qin, Simon Baumgartner, Cong Yu, and Donald Metzler. 2021. Charformer: Fast character transformers via gradient-based subword tokenization. *arXiv preprint arXiv:2106.12672*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Aidmar Wainakh, Fabrizio Ventola, Till Müßig, Jens Keim, Carlos Garcia Cordero, Ephraim Zimmer, Tim Grube, Kristian Kersting, and Max Mühlhäuser. 2022. User-level label leakage from gradients in federated learning. *arXiv preprint arXiv:2105.09369*.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2018. Glue: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP*.

Alex Warstadt, Amanpreet Singh, and Samuel R Bowman. 2019. Neural network acceptability judgments. *Transactions of the Association for Computational Linguistics*, 7:625–641.

Wenqi Wei, Ling Liu, Margaret Loper, Ka-Ho Chow, Mehmet Emre Gursoy, Stacey Truex, and Yanzhao Wu. 2020. A framework for evaluating gradient leakage attacks in federated learning. *arXiv preprint arXiv:2004.10397*.

Linting Xue, Noah Constant, Adam Roberts, Mihir Kale, Rami Al-Rfou, Aditya Siddhant, Paul Barham, and Colin Raffel. 2022. Byt5: Towards a token-free future with pre-trained byte-to-byte models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (ACL)*.

Alec Yenter and Abhishek Verma. 2017. Deep cnn-lstm with combined kernels from multiple branches for imdb review sentiment analysis. In *2017 IEEE 8th Annual Ubiquitous Computing, Electronics and Mobile Communication Conference (UEMCON)*, pages 540–546.

Xuefei Yin, Yanming Zhu, and Jiankun Hu. 2021. A comprehensive survey of privacy-preserving federated learning: A taxonomy, review, and future directions. *ACM Computing Surveys (CSUR)*, 54(6):1–36.

Ofir Zafrir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat. 2019. Q8bert: Quantized 8bit bert. In *5th Workshop on Energy Efficient Machine Learning and Cognitive Computing - NeurIPS*.

Mengjiao Zhang and Shusen Wang. 2021. Matrix sketching for secure collaborative machine learning. In *International Conference on Machine Learning*, pages 12589–12599. PMLR.

Mengjiao Zhang and Jia Xu. 2022. Byte-based multilingual NMT for endangered languages. In *Proceedings of the 29th International Conference on Computational Linguistics*, pages 4407–4417, Gyeongju, Republic of Korea. International Committee on Computational Linguistics.

Ligeng Zhu, Zhijian Liu, and Song Han. 2019. Deep leakage from gradients. *Advances in neural information processing systems*, 32.

A Environmental Settings

All the programs in our work are implemented using Python 3.12, PyTorch 1.13.0, and CUDA 12.2. For the hardware environment, we run all codes on a machine with Intel i7-11700K CPU, 64G memory, and NVIDIA GeForce RTX 3080 GPU.

B Experimental Details and More Results

B.1 Experiment on Performance

B.1.1 Glue Tasks

Dataset and evaluation metrics We used four standard benchmark datasets for GLUE tasks (Wang et al., 2018). The Microsoft Research Paraphrase Corpus (MRPC) (Dolan and Brockett, 2005) manually annotated sentence pairs extracted from news sources to indicate whether the sentences are semantically equivalent. The Quora Question Pairs (QQP) contains over 400,000 potential question duplicate pairs from the Quora platform, labeled as duplicate or non-duplicate. The Recognizing Textual Entailment (RTE) (Dagan et al., 2006) is to determine if a hypothesis sentence can be inferred from a premise sentence. The Corpus of Linguistic Acceptability (CoLA) (Warstadt et al., 2019) uses sentences from linguistics publications with binary acceptability judgments that indicate whether they are grammatically correct. We measured performance following standard evaluation protocols, using accuracy and F1 score for MRPC and QQP, accuracy for RTE, and Matthews Correlation Coefficient (MCC) for CoLA. The MCC metric is particularly appropriate for CoLA as it effectively handles the class imbalance present in the dataset. For all experiments, we used the officially provided train/validation splits.

Implementation Details For all GLUE tasks, we implemented comparable neural network models with either subword embeddings (Vaswani et al., 2017) or our **EMBYTE** approach. Each model used a 2-layer BiLSTMs, a hidden dimension of 256, and a dropout rate of 0.5 for regularization. For sentence pair tasks (MRPC, QQP, RTE), we encoded both sentences separately using the same LSTM network and then concatenated their representations before the final classification layer. For the single-sentence task (CoLA), we directly used the final hidden state of the LSTM for classification. The subword embedding dimension is 300, while for SEB the byte vocabulary size is 256 with 8 bytes per subword and the hidden dimension is 128. All models are trained using the Adam optimizer with a learning rate of 5e-4 and weight decay of 1e-4. For QQP, due to its larger size, we use a batch size of 128. For MRPC, RTE, CoLA, we use a batch size of 32 and trained on the full datasets.

B.1.2 Sentiment analysis

Dataset and evaluation metrics We use IMDb (Maas et al., 2011) and SST2 (Socher et al., 2013) datasets provided by Hugging Face. We use the accuracy for evaluation which is a routine in prior work (Minaee et al., 2019; Yenter and Verma, 2017).

We further investigated the impact of the number of bytes (n) used to represent each subword in our **EMBYTE** approach. Table 6 presents these findings, comparing various **EMBYTE** $_n$ (n ranges from 1 to 8) against the Subword baseline model on the SST-2 sentiment analysis task. Our **EMBYTE** variants consistently show a significant reduction in parameters. In terms of accuracy, although **EMBYTE** $_1$ (using a single byte per subword) performed poorly, increasing the number of bytes quickly improved model performance. The highestighest accuracy was observed with **EMBYTE** $_3$ and **EMBYTE** $_8$, both exceeding the baseline of the subword. Training times and the inference time per batch varied without a clear trend related to n.

Implementation Details We use 2-layer BiLSTMs for both IMDb and SST2 classification tasks. We keep all model architectures the same for the baseline models and models with our **EMBYTE** $_{co}$ except for the embedding parts. The subword embedding dimension is 64 and 256 for IMDb and SST2. The hidden units are 64 and 300 for IMDb and SST2. The hidden dimension of 2-layer FFN in **EMBYTE** $_{co}$ is 128 for both datasets. We optimize the model using Adam (Kingma and Ba, 2014) and the learning rate is 5×10^{-4} for the baseline and our method on both datasets. The best model parameters evaluated on validation data are applied for testing.

Compression Models The compression experiments comparing different projector architectures for our Subword Embedding from Bytes (**EMBYTE**) framework were carried out on the SST-2 dataset. A Bidirectional LSTM (BiLSTM) with 2 layers and a hidden dimension of 300 units per direction was used as the base sentiment classification model. The final subword embedding dimension fed into the BiLSTM was consistently 256. All models were trained for 15 epochs using the Adam optimizer with a learning rate of 5×10≏4 and a batch size of 64. A dropout rate of 0.4 was

applied within the LSTM and to the output of the embedding layer. For all SEB variants, n=8 bytes were used to represent each subword, and these bytes were initially represented as 256-dimensional one-hot vectors. The specific configurations for the tested projector architectures were as follows:

1. **FFN**: A two-layer Feed-Forward Network. ReLU activation and dropout (0.4) were applied after the first hidden layer.

2. **MLP-HD512**: An additional MLP layer (2048→512 with ReLU and dropout) was inserted before the original FFN. The main FFN then became 512→128→256.

3. **Attn-H4-D0.1**: A multi-head self-attention layer with 4 heads and an attention dropout of 0.1 operated on the sequence of 8 one-hot byte vectors (each 256-dim). The output sequence was then concatenated (to 8×256=2048 dimensions) and passed to the original FFN (2048→128→256).

4. **AE-Enc(512-256)**: An autoencoder-encoder structure (2048→512 with ReLU/dropout →256 with ReLU/dropout) processed the concatenated byte vectors. Its 256-dimensional output was then fed into the original FFN (256→128→256).

5. **TFEnc-H4-FFN512-D0.1**: A single Transformer Encoder layer processed the sequence of 8 one-hot byte vectors. This layer had 4 attention heads, an internal FFN dimension of 512, and a dropout of 0.1. The output sequence from the Transformer Encoder was concatenated (to 8×256=2048 dimensions) and then passed to the original FFN (2048→128→256).

Results of Compression Models Table 7 summarizes the performance of variant projector architectures (section 3.4) compared to a the Baseline traditional subword embedding model and **EMBYTE** , all evaluated on the SST-2. Our **EMBYTE** model using "FFN" improved accuracy to 0.817 while substantially reducing parameters to 3.80 million and maintaining a comparable training time and faster inference. Among the "MLP + FFN" variants, the MLP with hidden-dimension of 512 yielded the best accuracy of 0.832 with 4.65M parameters and a training time of 383 seconds. The "Self-Attention + FFN" models generally underperformed that even the best among them (Attention layer with 4 attention heads and dropout rate of 0.1) can only reach an accuracy of 0.786.The "AE-Encoder + FFN" model, specifically the Autoencoder-Encoder with intermediate dimension of 512 and bottleneck dimension of 256, performed similarly to the original FFN projector, achieving an accuracy of 0.819 with 4.75M parameters. Most notably, incorporating a "TFEncoder + FFN" layer yielded the highest accuracy. The Transformer Encoder with 4 heads and an internal FFN dimension of 512 achieved a top accuracy of 0.841 using 4.33M parameters. However, this came at the cost of significantly increased training time and a higher inference time compared to the simple **EMBYTE** . These results suggest that while the standard **EMBYTE** offers a strong balance of accuracy and efficiency, more complex projectors like a Transformer Encoder layer can further enhance accuracy, although with increased computational demands.

Results of Compression Models As our prior results show, the Transformer consistently outperforms other models. Standalone attention performs significantly worse than a Transformer because it lacks critical components—residual connections, layer normalization, and positional encoding—which are essential for stable training, deep representation learning, and capturing sequence order. A raw multi-head attention block, isolated from these stabilizing elements, struggles to learn effectively. To validate this, we conducted two additional experiments to incrementally reconstruct a full Transformer Encoder layer on top of our 'Attn' model (see Table 13):

1. **Adding Positional Encoding**: We first introduced positional encodings to the byte vectors before passing them to the attention layer. This alone improved the accuracy of the 'Attn-H4-D0.2' model substantially, raising it to 0.8080.

2. **Adding Residuals and LayerNorm**: Next, we added residual connections and layer normalization around both the attention and feed-forward components, forming a full Transformer Encoder layer. This further boosted accuracy to 0.8471, slightly surpassing our original 'TF-Enc-H4-FFN512' projector.

These results lead to a clear conclusion: the benefits of the more complex projector architecture are only realized when the full, stabilized Transformer design is used. The simple FFN remains a strong efficiency-performance baseline, while a complete Transformer Encoder layer delivers the highest accuracy. The failure of the standalone

attention block highlights the importance of architectural completeness.

B.1.3 Machine Translation

Dataset and evaluation metrics In the translation task, we consider two datasets, one is the medium-size IWSLT14 (Cettolo et al., 2014) dataset and a large-scale dataset WMT14 (Bojar et al., 2014). We follow the settings as prior work (Shaham and Levy, 2020; Zhang and Xu, 2022) and translate German (de) to English (en) in IWSLT14 (Cettolo et al., 2014). The translation of WMT is English (en) to German (de) and the preprocessing is the same as Fairseq (Ott et al., 2019). We use SacreBLEU, case-sensitive, with the 13a tokenizer (Post, 2018) as the evaluation metric.

Preprocessing Details For IWSLT14, there are 166K sentence pairs for training and validation and 5.6K for testing. The vocabulary shared by the source and target languages is built by BPE (Sennrich et al., 2015) with 10K tokens.

For WMT14, en-de contains 4.5M sentence pairs. Newstest2013 is used for validation and newstest2014 for testing respectively. The merge operation is 32K for BPE and the dictionary is shared by source and target.

Implementation Details The baseline we compare is the transformer with subword embedding (Vaswani et al., 2017). Our proposed method only replaces the subword embedding with **EMBYTE** .

For IWSLT14, the encoder and decoder layers are both 6 and have 4 attention heads. The hidden dimension of attention is 512 and the dimension of the feedforward layer is 1024. The optimizer is Adam (Kingma and Ba, 2014) with an inverse square root learning rate scheduler, and warm up 4000 steps. The learning rate is 5×10^{-4}. The total training epochs are 100, and we average the best 5 checkpoints for testing.

For WMT14, the encoder and decoder layers are both 6 and have 8 attention heads. The hidden dimension of attention is 512 and the dimension of the feedforward layer is 2048. The optimizer is Adam (Kingma and Ba, 2014) with an inverse square root learning rate scheduler, and warm up 4000 steps. The learning rate is 5×10^{-4}. The total training epochs are 100 and we use the early stop if the validation loss does not decrease in 5 epochs. We average the best 5 checkpoints for testing.

B.1.4 Language modeling

Dataset and evaluation metrics We use the same data as Fairseq did for the language modeling tasks. The dataset we use is WikiText-103. We use the same preprocessing and training settings as the official Fairseq does. The number of samples for training, testing, and validation are 1801350, 3760, and 4358 respectively. We evaluate the language modeling performance with perplexity.

Implementation Details In this experiment, we also use a two-layer FFN in **EMBYTE** , which has 4096 hidden units. The architecture is `transformer_lm` in the Fairseq framework. We share the input and output embedding in the encoder and the other hyperparameters and settings are the same as Fairseq.

B.2 Complexity Analysis

To demonstrate the efficiency of the proposed **EMBYTE** , we summarize the space and time complexity of each embedding method in Table 11. Here, the column "Memory" represents the memory usage for each embedding, and the column "Time" shows the time complexity in Transformer attention. For simplicity, we let $d' = d$ and use one linear layer as FFN in **EMBYTE** which contains nd^2 parameters.

In terms of space complexity, subword embeddings typically have an exponentially large vocabulary size V_w, exceeding 10^4, while the dictionary size of byte embeddings is no more than 256. For the proposed **EMBYTE** , the number of parameters in embedding is $\mathcal{O}(nd^2 + V_b d) = \mathcal{O}((nd + V_b)d)$, including the FFN and byte embedding matrix. In practice, $nd + V_b \ll V_w$. As a result, both byte embeddings and our proposed **EMBYTE** significantly reduce the memory cost required for embeddings. In 4.2, we show the analysis for space complexity in our experiments. Regarding time complexity, we analyze the attention in the widely used Transformer (Vaswani et al., 2017). Given the sequence length m, byte embedding is more time-consuming since the input length is c times longer than subword embedding. Here c is the average ratio between the lengths of byte and subword sequences. Based on the statistics (Shaham and Levy, 2020), c is usually around 5. However, our proposed **EMBYTE** maintains the same time efficiency as conventional subword embeddings because we preserve the subword sequence along with its boundaries.

One may consider the frequency analysis in cryptanalysis to get the original text if the attacker also has information about the tokens used for the text and the frequency of each byte. In Appendix C, we discuss the frequency analysis is not applicable to our proposed defense.

Table 13: Performance comparison of different compression architectures with attention-based improvements.

Model	Accuracy	Train (s)	Infer. (ms)	# Params (M)
Baseline	0.797	345	24.3	5.93
FFN	0.817	374	15.9	3.80
Attn-H4-D0.1	0.786	444	23.1	4.06
Attn-H4-D0.2	0.734	141	10.5	4.06
Attn-H8-D0.1	0.745	470	9.0	4.06
Attn-H8-D0.2	0.779	458	14.6	4.06
Attn-H4-D0.2 (+PosEnc)	**0.808**	450	23.5	4.06
Attn-H4-D0.2 (+PosEnc +Res + LN)	**0.847**	575	28.6	4.33
TF-Enc-H4-FFN512	**0.841**	569	28.4	4.33

B.3 Sequence Length Preservation

In our Decomposition-Compression (DeComp) learning framework, while the decomposition step may initially increase the sequence length by breaking a token into multiple semantic components, the subsequent compression phase reduces the length by projecting those components into a lower-dimensional representation.

To illustrate, suppose a token d has a direct embedding of [2, 5, 6]. After decomposition, it is represented by four semantic components (e.g., 14, 3, 10, 4), each of which is embedded into vectors such as [1, 2, 5], [3, 4, 2], [3, 2, 4], and [6, 2, 9]. These higher-granularity embeddings are then compressed into a single vector, e.g., [2, 3, 4], which is of the same dimensionality as the original token embedding. This means the final representation remains aligned with the original embedding size, effectively preserving compatibility and model training and running time efficiency.

Importantly, our method allows for even more aggressive compression. For instance, the final compressed vector could be reduced to a smaller size than the original token embedding, such as a scalar [2], if desired. In the paper, we chose to maintain the same dimensionality as the original embedding (e.g., [2, 3, 4]) for simplicity and fair comparison, not because of any technical limitation. In this way, DeComp preserves the same sequence length as the baseline subword embedding while requiring significantly less memory, using embeddings over just 256 bytes instead of the entire subword vocabulary.

B.4 Privacy Results

The reason for that is the parameters of the conventional subword embedding layer in BiLSTM take a large portion of the model parameters, making the model easily overfitting. In this experiment, **EMBYTE** $_{co}$ has smaller embedding parameters, which can address overfitting. We show that **EMBYTE** also learns the semantic meaning of subword

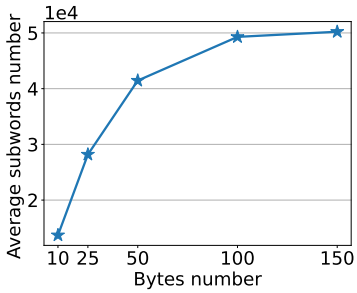


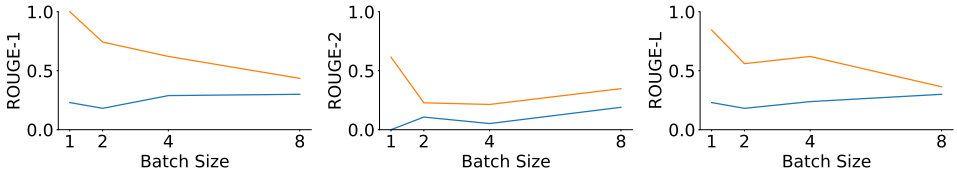
Figure 4: The average coverage of subwords given a random set of bytes with GPT-2 tokenizer.

B.5 Analysis for Memory Using

The Table 14 shows that, compared with the baseline subword model, our **EMBYTE** (equivalent to the FFN projector) demonstrated improved memory efficiency, requiring only 154.16 MB while achieving a higher accuracy of 0.817. Adding an MLP layer before the main FFN(MLP-HD512) resulted in a modest increase in inference memory to 168.51 MB, still below the baseline, and corresponded with a notable accuracy of 0.832. Similarly, the AE-Enc(512-256) model, which incorporates an autoencoder-encoder structure, showed comparable memory usage at 170.14 MB with an accuracy of 0.819. In contrast, projector architectures involving more complex attention mechanisms (Attn-H4-D0.1 and TFEnc-H4-FFN512-D0.1) exhibited higher memory demands. These results indicate that while the **EMBYTE** and its MLP or AE-Encoder augmented variants offer reduced inference memory usage compared to the baseline, the more powerful Transformer Encoder layer, despite its accuracy benefits, yields a higher memory cost during inference.

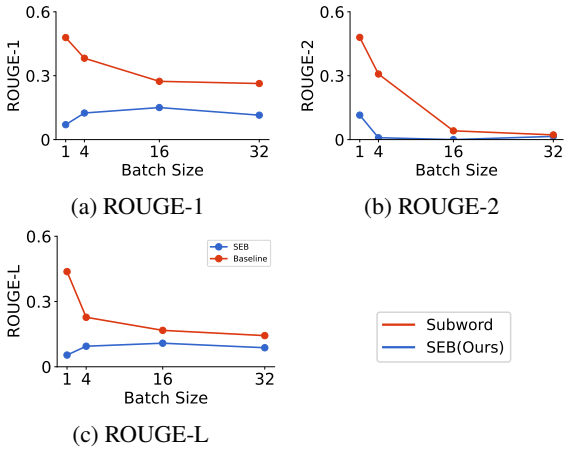
B.6 Parameter Reduction for Large Language Models

To estimate the impact of EMBYTE on prominent large language models (LLMs), we provide an analytical calculation of the potential parameter sav-



(a) ROUGE-1 (b) ROUGE-2 (c) ROUGE-L

Figure 5: Recovery performance for batch size 1, 2, 4, 8 on WikiText-103. Orange: Subword; Blue: **EMBYTE** .



(a) ROUGE-1 (b) ROUGE-2

(c) ROUGE-L

Figure 6: Recovery performance for batch size 1, 4, 16, 32 on pretrained subword model and pretrained-finetuned EmByte model.

Model Type	Accuracy	Infer GPU (MB)
Baseline Subword	0.7967	185.79
EMBYTE	0.8167	154.16
MLP-HD512	0.8315	168.51
Attn-H4-D0.1	0.7855	251.55
AE-Enc(512-256)	0.7967	170.14
TFEnc-H4-FFN512-D0.1	0.7967	232.68

Table 14: Memory usage of each compression model.

ings. The analysis uses the following formulas:

$$E_{\text{baseline}} = V \times D$$
$$E_{\text{EmByte}} = (256 \times n) \times H + H \times D$$
$$\Delta = E_{\text{baseline}} - E_{\text{EmByte}}$$

Where V is the subword vocabulary size, D is the model's hidden dimension, n is the number of bytes per subword, and H is the FFN hidden layer size in the EMBYTE projector.

The Table 15 shows the projected savings for GPT-2 and LLaMA-7B, assuming $n = 8$ and $H = 128$.

As the analysis shows, for a model like **GPT-2 small (117M)**, replacing its standard embedding layer (approx. 38.6M parameters) with our EmByte projector (approx. 0.36M parameters) would reduce the total model size by approximately **38.2M parameters**, a reduction of over **32%**.

For a larger model like **LLaMA-7B**, the embedding layer is proportionally smaller relative to the total model size, but the absolute reduction remains substantial. Replacing its embedding layer (approx. 131M parameters) with our EmByte projector (approx. 0.79M parameters) would reduce the model size by approximately **130M parameters**, representing a 1.9% decrease in total parameters. This demonstrates EMBYTE's significant compression benefits even at a very large scale.

B.7 Analysis for Semantic Meaning

In this section, we will analyze whether the derived subword embeddings from our method can truly encode the meaning of the words in the embedding space. To experiment on this aspect, we mainly calculate the cosine similarity between two word embeddings obtained based on our method **EMBYTE** . We list some examples in IMDb sentiment analysis in Table 3.

The cosine similarity demonstrates that the subword embedding of our proposed **EMBYTE** will learn the semantic meaning from the task. For example, positive words (good, great, and funny) have positive and high-value similarities with each other, which is also the same case for all negative words (bad, worse, and boring). However, all the negative-positive pairs have negative similarities, which means the subword embedding of our proposed **EMBYTE** can automatically learn the semantic meaning.

B.8 Comparison with BERT Models

Table 16 compares the memory, accuracy, and privacy properties of existing BERT models and **EMBYTE** . **EMBYTE** does not only reduce the memory footprint but also protect the privacy of the models against gradient attacks.

B.9 Comparison with Gradient Prune Defense

We compare the defense method of gradient pruning in the FILM attack for batch size = $8, 16, 32$.

Table 15: Projected parameter reduction for large language models using EMBYTE.

Model	V	D	Baseline Params	EmByte Params	Δ (Params Saved)	Δ% of Full
GPT-2 small (117M)	50,257	768	38.6M	0.36M	38.2M	32.6%
GPT-2 medium (345M)	50,257	1,024	51.4M	0.39M	51.0M	14.8%
LLaMA-7B (7B)	32,000	4,096	131M	0.79M	130M	1.9%

Model	Memory Reduction	Accuracy Loss	Good For	Privacy
DistilBERT	$\sim 40\%$	Minimal	General NLP tasks	✗
TinyBERT	$\sim 50\%$	Small	Mobile, fast inference	✗
ALBERT	$\sim 60\%$	Minimal	Pretraining & fine-tuning	✗
MobileBERT	$\sim 50\%$	None to small	On-device applications	✗
Quantized BERT	$\sim 50\%$	Negligible	Deployment, edge inference	✗
EMBYTE	$\sim 20\% - 90\%$	Negligible	Federated, memory-bound training	✓

Table 16: Comparison of training memory-efficient NLP models. Memory reduction refers to GPU/TPU memory usage during training. **EMBYTE** offers significant memory savings and privacy benefits.

Tables 17,18, and 19 show the precision and recall for gradient pruning and our method. Even without pruning, our method has a very low recall compared to FILM on subword embeddings when all batch sizes we experimented on, which shows the effectiveness of our defense.

Prune ratio	Precision		Recall	
	Subword	SEB	Subword	SEB
0	1	1	1	0.003
0.9	1	1	1	0.003
0.99	1	1	1	0.003
0.999	1	1	0.53	0.003
0.9999	1	0.46	0.08	0.003

Table 17: Defense results on precision and recall for batch size is 8.

Prune ratio	Precision		Recall	
	Subword	SEB	Subword	SEB
0	1	1	1	0.005
0.9	1	1	1	0.005
0.99	1	1	1	0.005
0.999	1	1	0.49	0.005
0.9999	1	0.50	0.06	0.005

Table 18: Defense results on precision and recall for batch size is 16.

C Discussion of Frequency Analysis

Frequency analysis is useful in cryptanalysis. For simple substitution ciphers, there is a characteristic distribution of letters that is roughly the same for almost all samples of that language. Frequency analysis uses the characteristic distributions of the plaintext and ciphertext to guess the mapping between them.

Prune ratio	Precision		Recall	
	Subword	SEB	Subword	SEB
0	1	1	1	0.009
0.9	1	1	1	0.009
0.99	1	1	1	0.009
0.999	1	0.99	0.51	0.009
0.9999	1	0.47	0.07	0.009

Table 19: Defense results on precision and recall for batch size is 32.

In the scenario of the threat model and our proposed method, the attacker only knows the gradients, model parameters and the mapping from subword to byte sequence. Based on these, the attacker can only get the information about distinct byte candidates which are updated in training. The attacker cannot determine the frequencies of the bytes based on the gradients.

To demonstrate the effectiveness of our method, we further assume the attacker have the information about the frequency of each byte. The goal of the attacker is to get the plaintext, given the plaintext to ciphertext mapping, and the characteristic distributions of the ciphertext.

To infer the plaintext, the attacker needs to know the order for combining all of the byte candidates and then use the ciphertext to plaintext mapping to obtain the original text. However, the attacker only knows a bag of bytes without ordering. It is difficult to infer the correct combination of the bytes as possible combinations of bytes is extremely large.

D Statics of distribution

When the vocabulary sizes of subwords and bytes are 50K and 256, the distribution of subword number, unique subword number, and unique byte num-

Algorithm 3 Fixed-Output-Length Coding

Input: A word sequence
Parameter: base b
Output: A code sequence

1: $L = \lceil \log_b^{|\mathbb{V}|} \rceil$ where $|\mathbb{V}|$ is the vocabulary size of the input word sequences, L is the code length, and b is the parameter of the alphabet size.
2: Generate all possible L-long code.
3: Shuffle the vocabulary words and assign one-to-one mapping between each word and the code.
4: **for** word in vocabulary **do**
5: Output its mapped code
6: **end for**
7: **return**

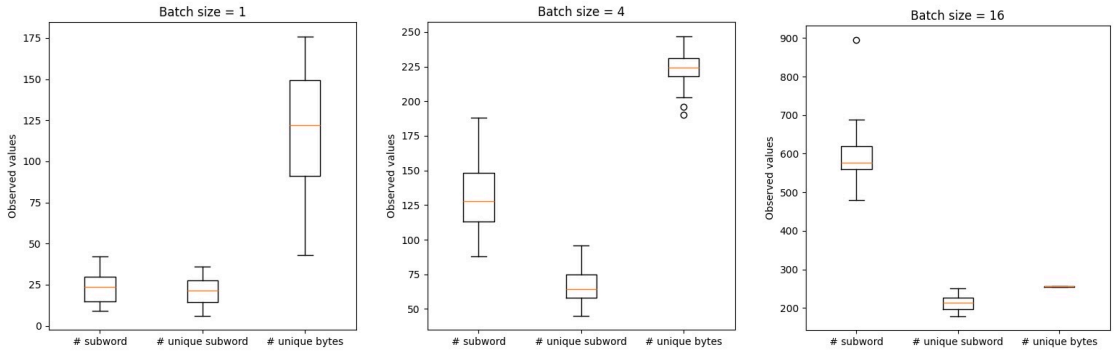
ber in batch sizes from 1 to 16 is shown in Figure 7.

E Subword-to-Byte Encoding Algorithm

Sensitivity analysis on FFN hidden units In this experiment, we test the sensitivity of **EMBYTE** $_{co}$ on FFN hidden units, because it is one of the major factors for embedding parameters. Here, we set different FFN hidden units as $\{128, 256, 512, 1024, 2048, 4096\}$, with the total embedding parameter numbers of 0.3M, 0.7M, 1.3M, 2.7M, 5.2M, and 10.5M, respectively. The number of embedding parameters and translation BLEU scores are shown in the left of Figure 8. When the numbers of hidden units are 256, 512, and 1024, **EMBYTE** $_{co}$ can obtain better performance with fewer parameters. Although the model can still achieve better performance when hidden units are larger than 2048, it does not have advantages over the original transformer on model size.

Sensitivity analysis on V_b and n To investigate the impact of the byte vocabulary size V_b and number of bytes per subword n, we set V_b as $64, 128, 256$ and n as $4, 8, 16$. Based on the previous experiments, we set the hidden units in the 2-layer FFN to 1024 in **EMBYTE** $_{co}$, which provides good performance with a small scale of parameters. We first discuss the model size in terms of embedding parameter numbers in Table 21. All settings have smaller embedding parameter numbers than the original Transformer. We further demonstrate the translation performance under these settings in Figure 8 (right). It indicates that increasing n leads to better model performance for a fixed V_b. The reason is increasing n results in more possible positions per byte token, providing more information in the aggregated vector. Similarly, when we fix n and increase V_b, the increased byte vocabulary diversity makes the aggregated vector more expressive. Therefore, increasing the byte vocabulary size and the number of byte tokens per subword can improve the model's expressiveness, leading to improved performance. Furthermore, Figure 8 (right) and Table 21 show that models with similar amounts of parameters have similar performance. As long as V_b and n ensure that **EMBYTE** $_{co}$ has sufficient expressive ability, the model performance is more related to the number of parameters than to specific V_b and n.

Machine Translation Results For IWSLT14, we run 5 trials and report the average performance with the standard deviation. We show the translation results of Transformer with subword embedding and **EMBYTE** in Table 9. The hidden dimension of the two-layer FFN is 2048 for IWSLT because we try to keep the total parameters of **EMBYTE** $_{co}$ the same as the original Transformer. For WMT, the hidden dimension of FFN is 4096. Here, we test three variants of **EMBYTE** when aggregating the byte embedding back to subword embedding: added real-valued embedding (**EMBYTE** $_{ar}$), concatenated real-valued embedding (**EMBYTE** $_{cr}$), and concatenated one-hot embedding (**EMBYTE** $_{co}$). In this experiment, the dimensions of real-valued and one-hot vectors are 512 and 256. Table 9 shows that **EMBYTE** $_{cr}$ and **EMBYTE** $_{co}$ can achieve better performances than subword embedding. Concatenating the one-hot vectors yields better results even with fewer model parameters than concatenating byte embedding. Therefore, we can conclude that **EMBYTE** is a better alternative to using large subword embeddings. Additionally, based on the comparison between **EMBYTE** $_{ar}$ and **EMBYTE** $_{cr}$, we find that concatenation is better than the simple adding of byte embeddings. This is expected as Section 3.2 because adding does not consider the positional information of bytes. The result of WMT14 shows the same performance as the subword-based model but with a smaller size of embedding parameters. It is important to emphasize that while privacy is improved, our model achieves the same or better accuracy than the baseline methods.



(a) Batch size = 1 (b) Batch size = 4 (c) Batch size = 16

Figure 7: The distribution of subword number, unique subword number, and unique byte number in a batch when batch size is 1, 4, 16. The vocabulary sizes of subwords and bytes are 50K and 256.

	Original Sentence	**Best Recovered Sentence**
Subword	The historic rainfall caused several dams to fill throughout northeast Mexico.	The rainfall caused several historic dams to fill throughout northeast Mexico.
EMBYTE	Pujols is a highly regarded hitter who has shown a "combination of contact hitting ability, patience, and raw power"	He is a professional who has a very high degree of ability, and always takes great advice, without ever assuming power" ("Pivotal Decision Making With Your Head". Retrieved 12 Dec 2007 16 Mar)

Table 20: The best recovered sentences by FILM using subword embedding and **EMBYTE** with batch size 1. Text in green are successfully recovered phrases and words.

Byte Tokens per Subword (n)	Byte Vocabulary Size (V_b)		
	64	128	256
4	0.79M	1.05M	1.57M
8	1.05M	1.57M	2.62M
16	1.57M	2.62M	4.72M

Table 21: Number of embedding parameters.

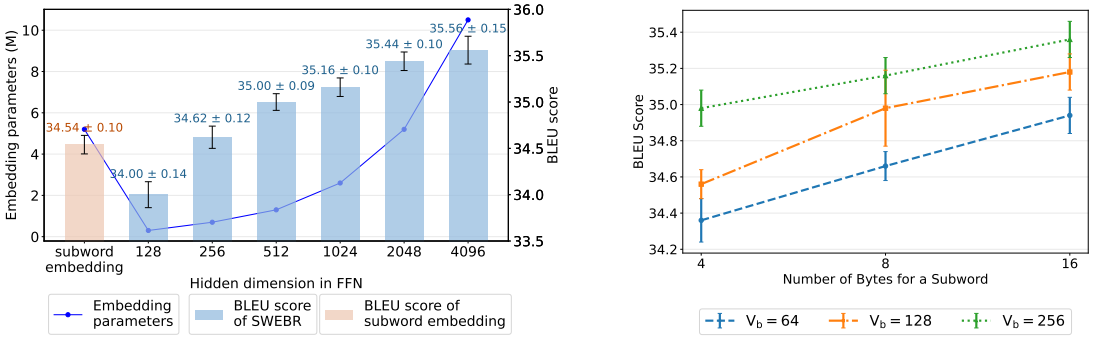


Figure 8: Results on embedding parameters, vocabulary size, and number of bytes per subword. Left: The BLEU scores versus hidden dimension in FFN and embedding parameters. Right: Comparison of mean BLEU scores for different byte vocabulary sizes and different numbers of bytes per subword.