SentinelLMs: Encrypted Input Adaptation and Fine-tuning of Language Models for Private and Secure Inference

Abhijit Mishra, Mingda Li, Soham Deo
School of Information
University of Texas at Austin
abhijitmishra, mingdali, soham.deo@utexas.edu

Abstract

This paper addresses the privacy and security concerns associated with deep neural language models, which serve as crucial components in various modern AI-based applications. These models are often used after being pre-trained and fine-tuned for specific tasks, with deployment on servers accessed through the internet. However, this introduces two fundamental risks: (a) the transmission of user inputs to the server via the network gives rise to interception vulnerabilities, and (b) privacy concerns emerge as organizations that deploy such models store user data with restricted context. To address this, we propose a novel method to adapt and fine-tune transformer-based language models on passkey-encrypted user-specific text. The original pre-trained language model first undergoes a quick adaptation (without any further pre-training) with a series of irreversible transformations applied to the tokenizer and token embeddings. This enables the model to perform inference on encrypted inputs while preventing reverse engineering of text from model parameters and intermediate outputs. After adaptation, models are fine-tuned on encrypted versions of existing training datasets. Experimental evaluation employing adapted versions of renowned models (e.g., BERT, RoBERTa) across established benchmark English and multilingual datasets for text classification and sequence labeling shows that encrypted models achieve performance parity with their original counterparts. This serves to safeguard performance, privacy, and security cohesively.

1 Introduction

In today's technology landscape, language models play a crucial role in numerous text-based applications. The rise of powerful pre-trained models like BERT (?), RoBERTa (?), and GPT (?) and their successors, followed by a massive outburst of generative large language models such as GPT-3 (?), PaLM (?), LLaMa (?) has led to a surge in innovative approaches and designs for large-scale language model pre-training. Characterized by an initial phase of broad pre-training, followed by targeted fine-tuning for specific contextual tasks, these models have found widespread utility across a diverse spectrum of applications (??) involving text classification, sequence labeling and text generation.

The widespread use of applications on server systems accessed through the internet has raised concerns about user privacy. Pre-trained language models are powerful tools, but their size and resource requirements make them challenging to use on personal devices. This leads to a situation where models are deployed on cloud-based servers and users send their text inputs over networks that can potentially be intercepted. For instance, sending text from users to servers using methods like HTTP GET/POST can expose the data to attacks known as *Man-in-the-Middle (MitM)* attacks (?), where an attacker sneaks between the user and the server, gaining access to and potentially altering the data being sent. This can result in data theft and misuse by malicious individuals. Another common worry related to using fine-tuned models on servers is the risk of unauthorized logging and misuse of users' personal data the organizations deploying these models (??).

To deal with these problems, strong protective measures like encrypting data and models can be undertaken, involving potent one-way string encryption techniques such as the Secure Hash Algorithm (SHA) and passkey-based secure hash algorithms like *Blake* (?). These methods convert input data into fixed-length hash values, which are incredibly difficult to reverse-engineer into the original input. However, encrypting input text introduces challenges for language models, typically trained and fine-tuned on plain text. This is due to two key reasons. Firstly, models pre-trained and fine-tuned on plain text may not adequately recognize encrypted inputs and may interpret them as *out-of-vocabulary* (OOV) terms. Secondly, encryption hash algorithms provide enhanced security mostly by transforming text in a way that sacrifices a significant amount of linguistic information. For instance, consider the words *cat* and *cats* in plain text. They are closely related, sharing morphology and lexical semantics. However, their respective encrypted forms using a 256-bit SHA encryption, `77af778b` and `d936608b` (truncated for brevity), bear no clear relationship. This poses difficulties in suitably modeling language tasks with encrypted inputs. Pre-trained language models thrive on linguistic patterns in the input text, a task made challenging by the transformation introduced during encryption.

Our work focuses on making transformer-based language models compatible with encrypted inputs. Recognizing the impracticality of extensively pre-training language models with encrypted raw internet data, we choose a more efficient route. We first adapt a pre-trained language model (such as BERT) to comprehend key-based encrypted text and only fine-tune it on task-specific encrypted training data. During adaptation, we encode vocabulary entries using a secure key-based string encryption method, such as *Blake* (**?**). We then shuffle the tokenizer's vocabulary and align the old token embeddings with the new vocabulary. This ensures that the text cannot be recovered either through network intrusion or server-side logging. Finally, to prevent retrieval of user inputs by reverse engineering of embeddings, we use distance-preserving geometric transformations of embeddings, which permanently alter embeddings to thwart attempts at reverting to original values.

After saving the altered model, we proceed with fine-tuning, where we encrypt the text examples in the training data using the same passkey used for the tokenizer transformation. The fine-tuning process follows usual established practices outlined in existing literature. Once fine-tuned and deployed, during inference, we propose that user text inputs first undergo pre-tokenization and encryption of text on users' devices, which is computationally lightweight. The encrypted text then reaches the server-based model pipeline, where a second-stage tokenizer further tokenizes the inputs for the adapted language models' inference.

We implement these adaptation and fine-tuning techniques on widely used pre-trained transformer based language models like BERT and RoBERTa, and evaluate these models on benchmark datasets for text classification and sequence labeling tasks. Our experiments demonstrate that the adapted and fine-tuned models achieve performance parity with the original models on relevant task metrics while ensuring enhanced privacy and security for the users.

Overall, this work focuses on a threat model identifying security and privacy risks, specifically interception vulnerabilities (unauthorized interception of user inputs) and privacy concerns (potential mishandling of user data). We prioritize irreversible data protection in fortification and fine-tuning, avoiding costly pre-training of the base model. The contributions can be summarized as follows:

- **Innovative Adaptation of Language Models:** We introduce a novel lightweight approach to easily adapt and fine-tune transformer based language encoders, enabling them to process passkey-encrypted inputs without requiring additional pre-training. This approach diverges from conventional practices in cybersecure language model development.
- **Enhanced Text Security:** We enhance text security in the adapted models by implementing systematic techniques such as tokenizer encryption, token shuffling, and token embedding transformation. These measures collectively make it challenging to reverse engineer the original text from the tokens and embeddings.
- **Empirical Validation:** Through extensive evaluation on various NLP benchmark datasets, we demonstrate that the overall transformation process maintains a lossless performance while being cost-effective. Additionally, the process is designed for easy replication to address new encryption requirements.

The code for this paper, is available at https://github.com/abhijitmishra/sentinellm-aaai2024.

2 Related Work

Deep neural network language models like BERT, RoBERTa, and GPT have gained significant prominence in modern natural language processing. Yet, the layers within these architectures, particularly those involved in input processing such as embedding layers, introduce a potential risk of information leakage. In certain instances, attackers can exploit mathematical analysis based reverse engineering to recover 50-70% of the original input (**?**). Some recent studies have addressed such privacy vulnerabilities, but few efficiently tackle this issue without compromising model performance. **?** (**?**) propose a homomorphic encryption based method for privacy preserved text classification. **?** (**?**) explore data encryption based on polynomial spaces and matrices for privacy preserving language model access, where as **?** (**?**) propose differential privacy based solutions for the same. However, these approaches are not entirely immune to the risk of data leakage. Also for differential privacy based solutions, susceptibility arises from the fact that the definition of private data varies by company, and even data labeled as "non-private" by the company remains vulnerable to leakage. Furthermore, these methods are prone to potential performance losses. Our approach focuses on fully encrypting the data at the user's end, with little to no performance loss in most cases.

Training smaller, private models based on large pre-trained models has become another prominent approach in privacy-preserving model development. Knowledge distillation has proven to be highly effective in this context. In their work, **?** (**?**) and **?** (**?**) adopt the knowledge distillation technique to safeguard member privacy in high-volume machine learning. Their focus, however, is on preserving the model's classification ability, thus making the solutions non-generic, unlike ours. Our work shares similarities with **?** (**?**), who introduce an embedding transformation technique involving noise induction and token character perturbation-based encryption. However, they apply it to a limited task set, and also need to continue pre-training language models like BERT with a masked language modeling objective. In contrast, we perform post-hoc transformation of pre-trained language models, thus bypassing the resource-intensive process of extensive model pre-training. This makes our approach a first of its kind to the best of our knowledge.

3 Central Idea

Making sure that network-based transactions are secure and private involves a critical requirement: *at no point during or after the transaction should the text provided by the user be understandable by people or machines to reveal any information about the text.* In usual situations, there are two ways security could be compromised: (a) a person sneaking into the network could intercept and collect personal data from the network packets, or (b) data reaching the server and being used by models for predictions might be stored

Raw Text	List down the medicare benefits that are associated with Social Security Numbe 000-00-0000 and Dates of Birth 1st January, 1950.
Tokenized Text	list down the medicare benefits that are associated with social security numb ##e 000 - 00 - 000 ##0 and dates of birth 1st january , 1950
Encrypted Version1	53bb 9b57 a6b7 5325 9919 c438 0187 eec5 98e2 f3c6 cd0b 4ae4 6c6d 9a31 fb0e ae90 fb0e 9a31 a771 3ecd e353 5787 af69 5293 af69 5293 713f 0ee1
Encrypted Version2	670c ba35 e816 eef6 382c 0bf7 baab 7563 0595 0c5a 1b5d e924 2c06 f9d8 8a5f 3f9b 8a5f f9d8 5faa 9881 7fc2 92c6 fda9 d228 fda9 d228 0cd1 25b2

Table 1: Examples showing input, tokenized and encrypted text. We show word-pieces from BertTokenizer. Version1 and Version2 are different encrypted hashes obtained using *Blake-32* bit encryption with example passkeys `llm123` and `nlp2023` respectively. Hashes are truncated for brevity.

and accessed for longer than necessary. Although these actions cannot be completely prevented, one can avoid security breaches by employing special encryption methods with *one-way encryption* algorithms. These algorithms are intentionally designed to be slow, rendering it impossible to revert the encrypted data to its original form; in other words, they cannot be decrypted. The encrypted data, often called ciphertext, is a scrambled or hashed uninterpretable version of the input text. Well-known algorithms that are considered strong for encrypting sensitive data (like passwords) include the IBM's Data Encryption Standard (DES), the Secure Hash Algorithm family (SHA) (**?**), MD5, and more recent methods like Blake (**?**). Algorithms like these typically help encrypt any length text into 128 or 256 bit hashes.

While these encryption methods guarantee robust conversion of text into ciphertext, they often obscure the underlying linguistic signals when the entire input is encrypted. To find a middle ground, we propose encrypting the text at token levels, encrypting one token at a time. This allows language models to still learn from repeated tokens and patterns in training examples. This strategy, however, harbors a potential drawback. Encryption applied solely at the token level may fall short in providing substantial security, as attackers could still recover the original text by exploiting repeated usage, data logging, and pattern extraction via rudimentary means like regular expressions.

To tackle this, we propose utilizing keyed encryption methods, such as *Blake*. This technique generates distinct ciphertext outputs based on user-specified passkeys or passphrases. Illustrated in Table 1, you can see an example sentence and its tokenized version. The encryption process transforms the same input into two completely distinct encrypted results using two different passkeys. In practical scenarios, integrating keyed encrypted tokens in widely used language model tokenizers like WordPiece, SentencePiece (**?**), and BytePair Encoding (**?**) is easy once a token-to-cipher map is established. This allows seamless use of pre-trained models on encrypted text, making them readily applicable. Yet, this doesn't guarantee safeguarding against server-side logging and reconstitution of text from cipher versions just based on token order. For instance, if the token indices in the original and encrypted vocabularies remain unchanged, recovering the original tokens becomes straightforward. This vulnerability can be mitigated by introducing random shuffling of tokens, disrupting their original positions in the vocabulary and other components of the language model reliant on token indices, such as token embedding layers of transformer-based models.

Following token shuffling, the last layer of defense that we propose is towards ensuring that the retrieval of original tokens can't occur through accessing and reverse-engineering *token embeddings*. Token embeddings serve as one of the earliest and potentially final layers of information accessible on the server side, directly conveying token-related details. In an ideal setting, token embeddings, renowned for capturing the distributional relationships between words (**??**), can inadvertently expose word information when subjected to certain mathematical operations like vector similarity based ranking and retrieval techniques. To address this concern, we propose a straightforward spatial and geometric transformation of token embeddings. This transformation preserves crucial semantic fidelity and token semantics in the altered space, while strategically ensuring that the original embedding space remains indecipherable from the transformed one. We believe, this approach will help prevent the reversal of the original tokens from the transformed embeddings, bolstering the overall security measures.

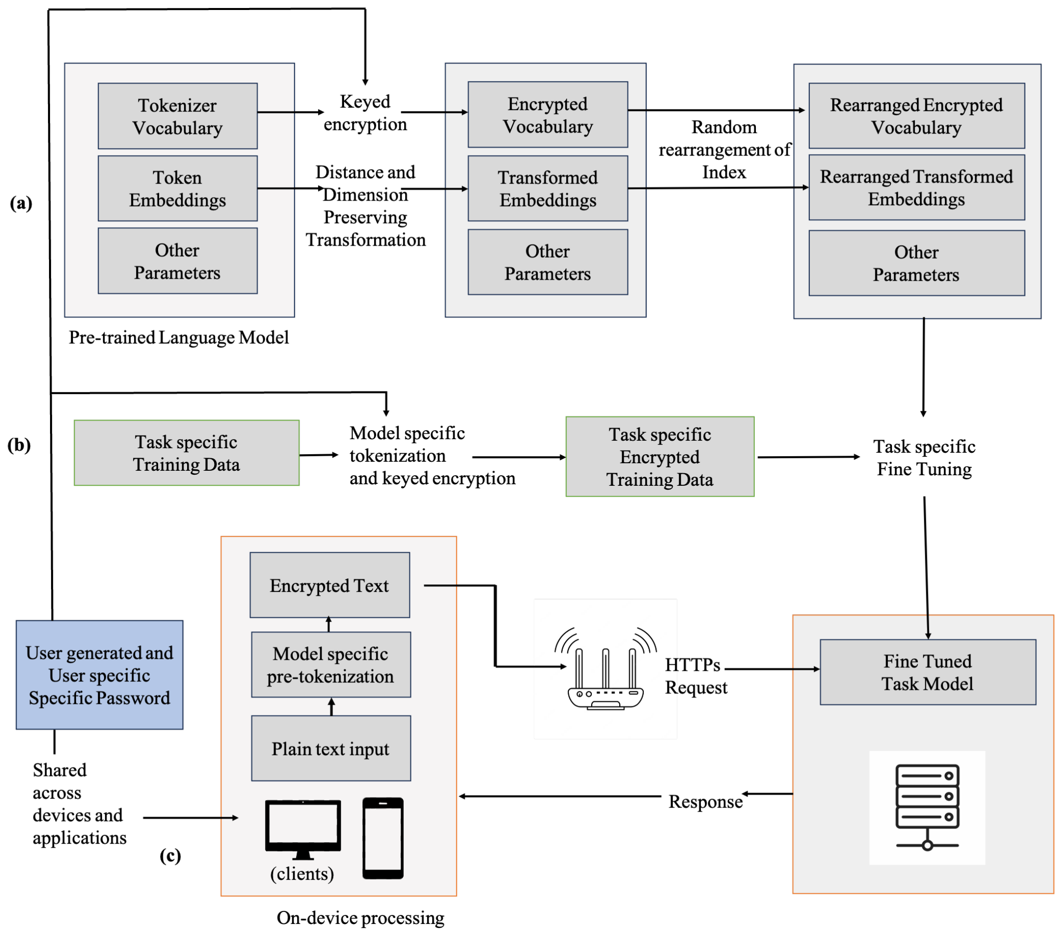
We bring these ideas together in the upcoming section, where we explain how we adapt and refine pre-trained models using encrypted text.

4 Model Adaptation and Fine Tuning

Our approach is designed with personalized user security in focus, ensuring that data preparation and encryption are tailored for each individual user (or user groups). The entire process, encompassing pre-trained model adaptation, model fine-tuning, and runtime usage, is depicted in Figure 1. We will now delve into the three key components of our methodology. It is important to note that in our discussion, we focus on a typical task setup: fine-tuning a pre-trained language model like BERT for a language-related task, such as text classification. This fine-tuning process involves utilizing an existing dataset for both training and evaluation purposes.

4.1 Adaptation of Pre-trained Models

This is a one time process and can be prompted by either a password update within an application or initiated by the server through periodic password changes. The process of adaptation entails the following three subprocesses.



Keyed encryption
Tokenizer Vocabulary
Token Embeddings
Other Parameters
Pre-trained Language Model
Distance and Dimension Preserving Transformation
Encrypted Vocabulary
Transformed Embeddings
Other Parameters
Random rearrangement of Index
Rearranged Encrypted Vocabulary
Rearranged Transformed Embeddings
Other Parameters
(a)
(b)
Task specific Training Data
Model specific tokenization and keyed encryption
Task specific Encrypted Training Data
Task specific Fine Tuning
Encrypted Text
Model specific pre-tokenization
Plain text input
User generated and User specific Specific Password
Shared across devices and applications
(c)
(clients)
On-device processing
HTTPs Request
Fine Tuned Task Model
Response

Figure 1: Illustration of the Workflow: User-Initiated Password-Driven Language Adaptation and Fine-Tuning Process. (a) Initial phase where a user-generated passkey initiates a one-time language adaptation. (b) Subsequent stage involving a one-time fine-tuning process. (c) Run-time scenario showcasing server-side inference on encrypted user input.

Tokenizer Encryption Initially, the tokenizer of the pre-trained model is modified by substituting its vocabulary items with their corresponding keyed encrypted forms using *Blake* that generates a 32 bit encrypted output. This approach enables the creation of multiple tokenizers, each tailored for various users and groups, and these tokenizers can be stored on servers. Tokenizers are resource-efficient to store and manage, making it cost-effective to maintain multiple versions of the same foundational tokenizer, each with distinct encryption characteristics. Since model-specific tokenizers operate on-device and at server side, we implement a generic tokenizer that processes the encrypted version of the pre-tokenized text, and this approach can work on a large range of vocabulary based tokenizers like word-piece and sentence-piece.

Token Embedding Transformation After tokenizer encryption, the token embeddings of the pre-trained transformer-based language models are extracted and transformed into an entirely different but distance preserving space. Considering a token embedding matrix of $M \times E$ where M is the number of tokens in the vocabulary and E is dimension of the embeddings, we apply a repeated number of *glide-reflections* on the matrix. A glide reflection is a type of transformation that combines a reflection and a translation in Euclidean geometry. Given any line vector l and a translation vector t that have equal number of dimensions as an embedding vector e (*i.e.* $e, l, t \in \mathbb{R}^d$), the reflection of the embedding vector e can be given as:

$$\mathbf{e}_{\text{reflected}} = \mathbf{e} - 2 \cdot \frac{\mathbf{e} \cdot \mathbf{l}}{\mathbf{l} \cdot \mathbf{l}} \cdot \mathbf{l} \tag{1}$$

and following a translation, the glided transformation would result in

$$\mathbf{e}_{\text{glide}} = \mathbf{e}_{\text{reflected}} + t \tag{2}$$

Glide reflection holds significance in embedding transformations due to its ability to maintain the spatial distances between any pair of embedding vectors, labeled as e_1 and e_2, in Minkowski space. This preservation remains consistent regardless of the specific choices regarding the line of reflection or translation vector used. This is because both re-

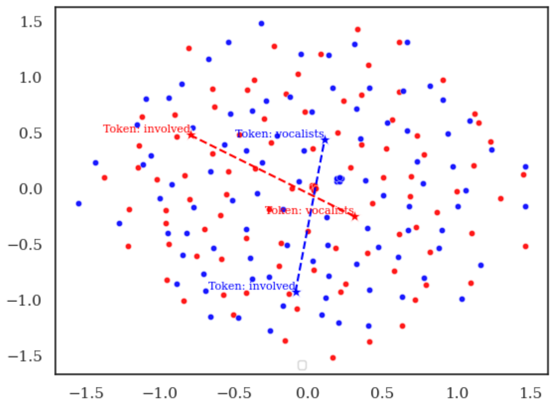


Figure 2: 2D plot of embeddings for 100 random tokens from the original bert-base-uncased model (in red dots) and same tokens from the transformed model after 10 iterations of glide reflection (in blue dots). The plot further illustrates that sample tokens "vocalists" and "involved" have altered positions with preserved spatial distance (dashed lines)

.

flections and translations are operations that inherently keep distances unchanged.

While we acknowledge that a straightforward process like glide reflection does not provide an absolute assurance against the potential recovery of original vectors from the transformed space, it is reasonable to assume that employing a matrix of vectors undergoing a sequence of glide reflections around various random lines and translation vectors would greatly amplify the complexity of any attempt to recover the initial vectors. This intricate sequence of transformations makes the task of returning to the original vectors highly improbable. To illustrate glide reflection, take a look at Figure 2. This figure displays embeddings from the original BERT (`bert-base-uncased`) model for 100 random tokens in red, while the transformed token embeddings resulting from 10 glide reflection iterations (*i.e.,* $nglide = 10$) are shown in blue. To fit the 768-sized embeddings into a 2D space for visualization, we used Multi Dimensional Scaling (**?**). As observed, even though two randomly chosen tokens, "involved" and "vocalists" have shifted from their original positions, the distances between them, measured in a straight line, have been maintained. This insight highlights the intriguing property of glide reflection. If this approach is to be adopted by multiple clients, the computational concern of allocating different passkeys and gliding values to each client is minimal as they involve deterministic transformations.

Vocabulary and Embedding Rearrangement After tokenization, encryption, and embedding transformations, the next step involves reorganizing the vocabulary and corresponding embedding indices while preserving alignment. This process is achieved by randomly selecting a subset of indices for shuffling, excluding specific tokens reserved for

Algorithm 1: AdaptLM

```
 1: function AdaptLM($M_E$, $nglides$, $V$, $key$)
 2:     $d \leftarrow$ dimension of E
 3:     for $i \leftarrow 1$ to $nglide$ do
 4:         l $\leftarrow$ GenRandomVector($d$)
 5:         t $\leftarrow$ GenRandomVector($d$)
 6:         $M_E \leftarrow$ GlideReflection($M_E$, $l$, $t$)
 7:     end for
 8:     $V \leftarrow$ Blake2bEncryption($V$, $key$)
 9:     $r \leftarrow$ shuffled indices from 0 to $d-1$ except special tokens
10:     $V, M_E \leftarrow$ Shuffle($V$, $M_E$,$r$)
11:     return $V$, $M_E$
12: end function
13: function GlideReflection($M_E$, l, t)
14:     Initialize $M_{\text{reflected}} \leftarrow$ to empty.
15:     for vector $e \leftarrow M_E$ do
16:         $e_{\text{reflected}} = e - 2 \cdot \frac{e \cdot e}{l \cdot e} \cdot l$
17:         Add $e_{\text{reflected}}$ to $M_{\text{reflected}}$
18:     end for
19:     $M_E \leftarrow M_{\text{reflected}} + t$
20:     return $M_E$
21: end function
22: function Shuffle($V$, $M_E$, $r$)
23:     $V, M_E \leftarrow$ Shuffled row indices of $M_E$ and vocabulary $V$
        and based on $r$
24:     return $V$, $M_E$
25: end function
```

padding, classification, beginning and end of inputs. The shuffling and reassignment of indices occur within both the vocabulary list and the embedding matrix.

After the reordering of indices, the adapted pre-trained model is stored and becomes available for future fine-tuning processes. The complete process is outlined in Algorithm 1. The inputs to the algorithm are the embedding matrix M_E, indexed vocabulary list V, number of iterations for glide reflection $nglide$ and the algorithm returns a transformed and shuffled version of M_E and encrypted and shuffled vocabulary list V.

4.2 Fine-tuning

With the model now prepared to accept and process encrypted inputs, we proceed to fine-tune it using encrypted task-specific training data. In the case of a pre-trained model and a tokenizer customized with a provided passkey, the fine-tuning process begins by tokenizing the training data with the original tokenizer (prior to encryption). These tokens are then isolated and encrypted using the *Blake* keyed encryption algorithm. Using the encrypted tokens and existing labels from the dataset, models are fine-tuned for various classification and sequence labeling tasks. In scenarios involving sequence labeling tasks, the integrity of the input and token mapping is preserved during the preprocessing of training data. It is worth noting that, given the similar configurations of the adapted and original models, fine-tuning demands equivalent time and resources as original models.

4.3 Real-time Usage

To accommodate the need for token-level encryption during inference, we present a straightforward approach as follows.

Model	CoLA	SST-2	MRPC	STS-B	QQP	MNLI	QNLI	RTE	WNLI
$\text{BERT}_{\text{original}}$	56.53	92.32	88.9	88.6	90.7	83.9	90.7	65.7	56.3
$\text{RoBERTa}_{\text{original}}$	67.8	94.8	90.2	91.2	91.9	87.6	92.8	78.7	56.34
$\text{EBERT}_{\text{passkey=llm123,nglide=3}}$	55.3	92.55	89.76	88.61	91.14	84.43	91.45	66.06	56.34
$\text{ERoBERTa}_{\text{passkey=llm123,nglide=10}}$	59.05	94.61	92.04	89.96	91.19	87.75	92.48	70.04	56.34
$\text{EBERT}_{\text{passkey=nlp2023,nglide3}}$	55.24	92.66	89.64	88.69	91.19	84.56	91.43	66.79	56.34
$\text{ERoBERTa}_{\text{passkey=nlp2023,nglide10}}$	60.57	94.38	92.14	89.74	91.40	87.62	92.71	68.95	56.34

Table 2: GLUE benchmark task metrics for the encryption adapted models starting with the letter E with different passkeys and number of glides (nglides), specified in the subscripts. Metrics: CoLA - Matthews Correlation, SST-2 - Accuracy, MRPC - F1, STS-b - Pearson, QQP - Accuracy, MNLI - Matched Accuracy, QNLI - Accuracy, RTE - Accuracy, WNLI - Accuracy.

First, the user's input is tokenized using a model-specific tokenizer. For instance, if the adapted model is derived from a bert-base model, the applicable tokenizer is BertTokenizer, which should execute on the user's device. Next, the tokenized input is encrypted using the user specific passkey. It is reasonable to anticipate that on-device tokenization and encryption are feasible, as these operations generally involve lightweight processes that do not require substantial computational resources. With on-device tokenization and encryption done, the encrypted data is transmitted through the network, adhering to security protocols, such as HTTPS.

Upon arriving at the server, the tokenizer corresponding to the adapted model takes over. This tokenizer further tokenizes the encrypted input and correlates the tokens with their respective indices. Given that indices and embeddings have been shuffled at this stage, the likelihood of order-dependent retrieval diminishes significantly. Following tokenization and embedding lookup, the input undergoes the model's deeper layers in the fine-tuned architecture and the resulting decisions are relayed back to the user in an appropriate response structure. This streamlined approach ensures data security while maintaining efficiency, offering a robust solution for token-level encryption during inference.

5 Experimental Setup

5.1 Datasets and Language models

With the objective of demonstrating equivalence between fine-tuned original models and fine-tuned adapted models (as detailed in Section 4), we opt for established and extensively documented datasets for text classification. These datasets include the well-regarded General Language Understanding Evaluation (GLUE) benchmark dataset (**?**), the CoNLL2003 Named Entity Recognition Dataset (**?**) for sequence labeling tasks, and the XNLI dataset (**?**) specifically curated for multilingual natural language inference evaluation. For XNLI, we evaluate all models in two settings: (a) *In-language*, where the data for training and testing come from the same language source and (b) *Zero-shot*, where the training data contains only English examples and test data is in target language.

5.2 Implementation Details

For the pre-trained models, we select BERT (`bert-base-uncased`), RoBERTa (`roberta-base`), and mBERT (`bert-base-multilingual-uncased`). We source these pretrained models and their corresponding fine-tuning configurations from Huggingface[1], a well-known platform in the field. For adaptation, we consider two sample passkeys: "llm123" and "nlp2023". To ensure reproducibility, all randomized operations are initiated using numbers obtained from the passkeys through character-to-byte conversion. The line and translation vectors are obtained by randomly initializing vectors with non-zero positive values between 0 and 1. We explore the effects of different numbers of gliding iterations: 3 and 10. For keyed encryption, **Blake2** implementation of the `hashlib` Python library is used.

6 Results and Analysis

Table 2 showcases the results obtained across various tasks within the GLUE benchmark. Our diverse set of adapted and fine-tuned model variants consistently demonstrates performance within a 2% margin of the original models[2] across tasks, with the exception of CoLA and RTE (with RoBERTa). A specific observation is the performance parity achieved for the WNLI task by specifying a slightly different learning rate (from the default value of 2e-5 to 1e-4). Similar parity is evident in the NER task for both BERT and RoBERTa models. Additionally, for the multilingual XNLI task, our adapted fine-tuned models demonstrate accuracy parity when compared to the original models across both in-language and zero-shot settings. Notably, models subjected to a higher number of transformation iterations ($nglide = 10$) exhibit similar performance as the original model while ensuring better privacy preserving abilities due to repeated embedding transformations, as discussed in Section 4.1.

In instances where models display lower performance, we speculate that refining the remaining layers of the language model may require extending the training iterations– a hypothesis currently under validation. This highlights an ongoing effort to optimize model performance and underscores the importance of iterative refinement in achieving robust language understanding across diverse tasks within the GLUE benchmark. It is also worth noting that difference in results with identical models with different pass keys arise from using distinct random seeds during adaptation. Consis-

[1]https://huggingface.co

[2]Result references: https://gluebenchmark.com/leaderboard/, https://github.com/facebookresearch/fairseq/tree/main/examples/roberta, https://huggingface.co/textattack/roberta-base-WNLI

Model	Precision	Recall	F1	Accuracy
$\mathrm{BERT}_{\mathrm{original}}$	94.69	95.15	94.92	98.93
$\mathrm{RoBERTa}_{\mathrm{original}}$	95.34	95.98	95.66	99.25
$\mathrm{EBERT}_{\mathrm{passkey=llm123,nglide=3}}$	93.86	94.87	94.36	98.90
$\mathrm{ERoBERTa}_{\mathrm{passkey=llm123,nglide=10}}$	94.95	95.79	95.37	99.23
$\mathrm{EBERT}_{\mathrm{passkey=nlp2023,nglide3}}$	93.86	94.68	94.27	98.85
$\mathrm{ERoBERTa}_{\mathrm{passkey=nlp2023,nglide10}}$	95.16	95.96	95.56	99.24

Table 3: Classification Metrics of Models on Sequence Labeling Task (NER) on CoNLL2003 Dataset

In-language	es-es	de-de	zh-zh
$\mathrm{BERTMultilingual}_{\mathrm{original}}$	77.3	75.2	74.2
$\mathrm{EBERTMultilingual}_{\mathrm{passkey=llm123,nglide=10}}$	78.7	76.8	73.9
Zero-shot	en-es	en-de	en-zh
$\mathrm{BERTMultilingual}_{\mathrm{original}}$	74.3	70.5	63.8
$\mathrm{EBERTMultilingual}_{\mathrm{passkey=llm123,nglide=10}}$	75.5	71.1	65.3

Table 4: Accuracy of Models Multilingual Natural Language Inference (XNLI) Task

tent outcomes can be achieved by employing the same seed values.

6.1 Investigations on Recoverability of Tokens

We investigate the feasibility of retrieving tokens by leveraging embeddings from both the original and transformed versions. To explore this, we calculate the pairwise Euclidean distances between original and transformed token embeddings, subsequently compiling a list of closest neighbors for each vocabulary token. We then determine the accuracy of predicting the nearest neighbor (with neighbors obtained from the original embeddings considered as the benchmark). The accuracies turn out to be negligible (**0.009%** in most cases), illustrating a systematic shift of embeddings from their original positions.

Given that distance preservation occurs in the transformations applied to token embeddings during adaptation, there also arises a valid concern that pairwise distances might serve as clues for recovering embeddings. While this holds true for non-fine-tuned models, our primary focus is to ensure the infeasibility of such recovery once the models undergo fine-tuning, as these fine-tuned models are the ones ultimately deployed for practical use. To validate this, we execute a comprehensive experiment. Both the original and adapted models for BERT and RoBERTa are fine-tuned using only 10 examples and only for one epoch across all GLUE tasks. After fine-tuning, we extracted the embedding matrices and generated ranked lists of tokens based on pairwise distances for each token in the vocabulary. Remarkably, the rankings obtained for the original fine-tuned models and their adapted counterparts exhibit only about a **0.01%** match across all tasks. This substantial disparity underscores the inefficacy of potential attacks aimed at recovering tokens. Overall, these experiments serve as a crucial assurance of the security and integrity of our fine-tuned models in real-world deployment scenarios.

7 Conclusion and Future Work

In this paper, we addressed pressing privacy and security concerns associated with pre-trained transformer-based language models, emphasizing the substantial risks posed by server-deployed models for user privacy. To mitigate these challenges, we proposed an innovative approach involving the adaptation and fine-tuning of existing language models for passkey-encrypted user-specific text. Adaptation of pre-trained language models through irreversible transformations on the tokenizer and token embeddings facilitates secure inference on encrypted inputs, ensuring the confidentiality of text. Furthermore, we observed that fine-tuning on encrypted training data maintains performance parity with original models across various benchmark datasets. In the future, we aim to expand our approach to include generative models and investigate improved encryption strategies for non-English and multilingual tasks. As encryption requirements evolve, our method can seamlessly incorporate advanced cryptographic algorithms or explore post-quantum encryption methods. While we consider our approach robust against common attacks, additional safeguards like multi-party computation or homomorphic encryption could strengthen its security, marking a substantial advancement in ensuring user privacy and security in practical applications.

Ethical Statement

language model, it is essential to note that our approach has been evaluated exclusively on fine-tuned language encoders and not on generative language models. Generative models heavily capitalize on shared tokenizers and vocabulary for both encoders and decoders, which is incompatible with our approach. Also, while we are advancing privacy-preserving methodologies, it is crucial to emphasize that our endeavors have not involved the utilization of private data at any point. Our approach has exclusively relied on publicly accessible benchmark datasets, notably including GLUE. However, it's essential to acknowledge the potential biases present in these public datasets, particularly within the realm of language model benchmarking.