On the (In-)Security of the Shuffling Defense in the Transformer Secure Inference

Zhengyi Li[1], Yakai Wang[1], Kang Yang[3],
Yu Yu[1,2], Jiaping Gui[1], Yu Feng[1], Ning Liu[1], Minyi Guo[1,2], Jingwen Leng[1,2]
[1]Shanghai Jiao Tong University, [2]Shanghai Qizhi Institute,
[3]State Key Laboratory of Cryptology
Correspondence: jgui@sjtu.edu.cn, yangk@sklc.org, ningliu@sjtu.edu.cn

Abstract

For Transformer models, cryptographically secure inference ensures that the client learns only the final output, while the server learns nothing about the client's input. However, securely computing nonlinear layers remains a major efficiency bottleneck due to the substantial communication rounds and data transmission required. To address this issue, prior works reveal intermediate activations to the client, allowing nonlinear operations to be computed in plaintext. Although this approach significantly improves efficiency, exposing activations enables adversaries to extract model weights. To mitigate this risk, existing works employ a shuffling defense that reveals only randomly permuted activations to the client. In this work, we show that the shuffling defense is not as robust as previously claimed. We propose an attack that aligns differently shuffled activations to a common permutation and subsequently exploits them to extract model weights. Experiments on Pythia-70m and GPT-2 demonstrate that the proposed attack can align shuffled activations with mean squared errors ranging from 10^{-9} to 10^{-6}. With a query cost of approximately \$1, the adversary can recover model weights with L1-norm differences ranging from 10^{-4} to 10^{-2} compared to the oracle weights.

1 Introduction

The rapid advancement of the Generative Pre-trained Transformer (GPT) (Cohen and Gokaslan, 2020) has led to various applications in the real world. Machine Learning as a Service (MLaaS) is a primary way for clients to access intelligent applications. In MLaaS, a server hosting a private Transformer model provides a public API, allowing the client to submit data and receive inference results. This raises privacy concerns for both clients and servers, as each party is unwilling to share their data with the other.

Secure inference using multi-party computation (MPC) or homomorphic encryption (HE) offers a privacy-preserving solution for both parties. Full-model encryption (FME) inference operates entirely on encrypted data, ensuring that the client only learns the inference result while the server learns nothing. However, employing cryptographic protocols incurs substantial costs (Huang et al., 2022; Knott et al., 2021). One major bottleneck arises from nonlinear layers. Secure computation of nonlinear layers typically utilizes piecewise polynomial approximations or iterative methods, which require multiple calls for secure multiplication, truncation, and comparison. These operations involve tens of communication rounds and large data transmission. Compared to linear layers, which require only a single multiplication and truncation, nonlinear layers contribute most to the overall latency. For example, prior studies report that the secure computation of nonlinear layers accounts for $75\% \sim 90\%$ of the total latency (Cho et al., 2022; Li et al., 2023; Pang et al., 2024).

To mitigate the bottleneck on nonlinear layers, some works reveal intermediate activations to the client to evaluate nonlinear layers in plaintext. This approach is referred to as linear-only encryption (LOE) inference. Even for complex nonlinear layers such as softmax, GELU, and layernorm, their evaluation can be completed in two rounds of communication. Existing works report tens of times speedup for Transformer models, making LOE inference practically feasible. However, revealing activations to the client introduces potential leakage of model weights as a trade-off. Early attempts to mitigate these risks, such as limiting the number of queries (Zhang et al., 2018) or using Bayesian networks (Xie et al., 2019), have been proven insufficient(Wong et al., 2020). More recent research (Zheng et al., 2022; Yuan et al., 2023; Zheng et al., 2024; Liu et al., 2024; Luo et al., 2025) introduces the insight that computing nonlinear layers is independent of the positions of values in the activation tensor. This insight enables a more ro-

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 29084–29098

bust defense by revealing shuffled intermediate activations to the client. When permuting a tensor with d elements, the probability that a client correctly guesses the permutation is $1/(d!)$, which is negligibly small for Transformer models with hundreds of dimensions. Consequently, shuffled activations are considered difficult to exploit by adversaries to exploit in existing black-box attacks (Carlini et al., 2020, 2024).

This work challenges the security claims of the shuffling defense applied to Transformer models. We show that model weights can be extracted as long as the adversarial client is able to align differently shuffled activation vectors to a common permutation, even when this permutation remains unknown. To achieve this, the adversary queries the Transformer to generate activation vectors that are differently shuffled but numerically close. Since shuffling preserves the values, this proximity enables the recovery of element-wise correspondences across vectors. The aligned activations can then be used to extract model weights by solving a linear system. As the true permutation is unknown to the adversary, the recovered weights differ from the original weights by row-wise and column-wise permutations. Despite this discrepancy, we show that the extracted weights preserve computational correctness during forward propagation.

The extracted weights are useful for several reasons. First, they closely approximate the original weights and can be used to analyze the private model. Second, they strengthen existing black-box attacks, such as by serving as effective initialization for surrogate models. Overall, the proposed attack reveals information leakage beyond what is assumed under the shuffling defense, calling for a reevaluation of its security in secure inference and related privacy-preserving settings (Shen et al., 2022; Meehan et al., 2021).

Our contributions are summarized as follows:

- We demonstrate that an adversary can successfully align activation vectors that are shuffled differently but have close values. Moreover, to address the issue of discrete inputs in Transformer models that cannot generate intermediate activations with close values, we leverage the probabilistic errors inherent in the secure truncation protocol to inject the desired perturbation.
- With the aligned activations, we theoretically show that the adversary can extract model weights that differ from the original weights up to row-wise and column-wise shuffling. Despite the row and column shuffling, the extracted weights perform the same forward propagation as the original weights, even if the adversary does not know the specific permutation.
- We evaluate the effectiveness of the proposed attack on two Transformer architectures widely used in secure inference, Pythia-70m and GPT-2. Experimental results show that we can align the shuffled activations with mean squared errors ranging from 10^{-9} to 10^{-6}. With a total query cost of approximately one dollar, the adversary can extract model weights whose L1-norm differences from the original weights range from 10^{-4} to 10^{-2}.

2 Preliminaries

This work examines the secure inference of the Generative Pre-trained Transformer (GPT) (Cohen and Gokaslan, 2020). There is a client with a private input and a service provider (server) with a private model. The model architecture is required to be known by both parties to interactively execute the protocols. The server aims to hide the model weights, while the client wants to protect the input.

2.1 Full-model-Encryption Secure Inference

Privacy goal of the FME. Secure inference based on homomorphic encryption (HE) and multi-party computation (MPC) provides a solution to protect the privacy of both servers and clients. Secure inference with full-model encryption (FME) achieves the following privacy goals: the client P_1 only learns the inference results, while the server P_0 knows nothing about the client's input.

Explain FME Inference. Figure 1 illustrates the FME secure inference (Li et al., 2024; Kei and Chow, 2025; Park et al., 2025; Moon et al., 2025) that operates entirely on encrypted inputs and model weights, and only the final output is revealed to the client. Each layer's inputs and outputs are additive secret shares $[\cdot]_i$, allowing for flexible linkage of different layers. The additive secret share (Ito et al., 1989) over the ring $\mathbb{Z}_{2^\ell}$ is defined as follows: for a given value $x \in \mathbb{Z}_{2^\ell}$, the client and server each hold two random shares $[x]_0 \in \mathbb{Z}_{2^\ell}$ and $[x]_1 \in \mathbb{Z}_{2^\ell}$ such that $x = [x]_0 + [x]_1 \mod 2^\ell$ holds. To convert the floating-point value x_f of the model to the ring value x, it is scaled by a large factor and rounded to the nearest integer:

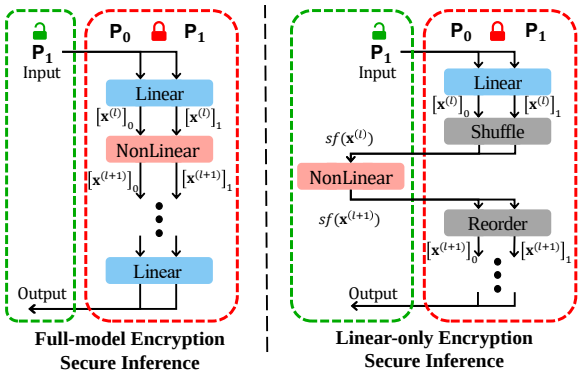

Figure 1: The red locked part is private to the owner or invisible for both parties, and the green unlocked part is revealed to the client. FME inference only reveals the final output to the client. LOE inference also reveals shuffled activations to the client.

$x = \lfloor x_f \cdot 2^p \rceil$ for p-bit precision. The decoding of x is computed as $x_f = x/2^p$.

Nonlinear Layers as the Bottleneck. The primary efficiency bottleneck for cryptographic methods lies in nonlinear layers. In contrast to linear layers, which typically require only two rounds of communication, nonlinear layers usually demand dozens of rounds of communication along with a significant amount of data transmission. In secure computations, the costs associated with communication rounds and data transmission are the primary bottleneck. Consequently, the considerable communication overhead makes nonlinear layers become the major bottleneck in secure inference, accounting for approximately $75\% \sim 90\%$ of the communication (Cho et al., 2022; Li et al., 2023; Pang et al., 2024), especially under suboptimal network conditions.

2.2 Linear-only-Encryption Secure Inference

Explain LOE Inference. Despite numerous efforts (Rathee et al., 2020, 2021; Pang et al., 2024; Li et al., 2024) to enhance protocol efficiency or replace nonlinear layers with cryptography-friendly alternatives (Mishra et al., 2020; Cho et al., 2022; Chen et al., 2022a; Li et al., 2023), nonlinear layers remain the major bottleneck. Another category of works aim to find a balance between performance and privacy. They propose linear-only encryption (LOE) inference, which reveals intermediate activations to the client, enabling the computation of nonlinear layers in plaintext (Zhang et al., 2018; Xie et al., 2019; Zheng et al., 2022; Liu et al., 2024; Luo et al., 2025). This approach allows nonlinear layers to require only two rounds of communication, regardless of their complexity. LOE inference shows tens of times speedup compared to FME inference, effectively advancing secure inference to a practical solution.

Shuffling defense. As a trade-off for performance, revealing activations to the client may expose model weights. In a typical deep neural network model, which consists of interleaved linear and nonlinear layers, revealing activations makes both the input and output of the linear layers accessible to the client. Consequently, the weights can be extracted by solving a linear system. Early attempts to mitigate this risk, such as limiting the number of queries (Zhang et al., 2018) or using Bayesian networks (Xie et al., 2019), have proven insufficient(Wong et al., 2020).

Later works propose the shuffling defense (Zheng et al., 2022; Liu et al., 2024; Zheng et al., 2024; Luo et al., 2025) to enhance the performance-privacy trade-off, as illustrated in Figure 1. They provide insight that correctly computing nonlinear layers is independent of the positions of values in the activation tensor. Some nonlinear layers, such as ReLU and GELU, support shuffling across the entire activation tensor. Other nonlinear layers, like softmax and LayerNorm, support only row-wise shuffling due to information reduction along the last dimension. Based on this insight, the activations are randomly shuffled before being revealed to the client. The results of nonlinear layers are then re-shared and reordered among the parties. The shuffling defense does not introduce extra communication overhead since shuffling and reordering can be performed locally.

Prior security analysis. Prior works claim (Zheng et al., 2022; Liu et al., 2024; Zheng et al., 2024) recovering the shuffling is practically impossible when the number of elements is not too small. For a vector with hidden dimension h, there are $h!$ possible permutations. In Transformer models, where $h \geq 100$, there are at least $100! \approx 10^{157}$ possible permutations. Thus, the probability of correctly guessing the permutation is negligible. These works also show original activations and shuffled activations have minimal correlation, and thus existing black-box attacks fail to exploit the destroyed information for meaningful attack. Note prior works do not regard permutation among different vectors as a defense. For example, the intermediate activation in the GPT decoding only contains one vector. In other Transformer applications, the adversary can also bypass the permutation in

this dimension by feeding a single token as input.

2.3 Transformer Model

This work focuses on the Transformer (Vaswani et al., 2017) models, especially the widely used GPT model (Cohen and Gokaslan, 2020). The main body of Transformer models comprises stacked Transformer encoders/decoders, each containing an attention module and a feed-forward network (FFN) module. Existing LOE works (Zheng et al., 2024; Yuan et al., 2023; Luo et al., 2025) allow the softmax, GELU, and layernorm layers to be evaluated in plaintext.

Attention Module. The attention module starts with a linear layer Linear_{qkv}, which projects the input into three independent activation tensors: $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$. The multi-head attention mechanism splits them into H heads and parallelly compute all heads. The h_{th} head is computed as:

$$\mathbf{Attn}(\mathbf{Q}^h, \mathbf{K}^h, \mathbf{V}^h) = \mathsf{softmax}\left(\frac{\mathbf{Q}^h {\mathbf{K}^h}^T}{\sqrt{d_k}}\right)\mathbf{V}^h, \quad (1)$$

where d_k is the hidden dimension of the $\mathbf{K}^h$. The outputs of all heads are concatenated and fed into another linear layer Linear_o. Finally, a residual connection and a normalization layer are applied to generate the module's output. The $\mathbf{K}$ and $\mathbf{V}$ of prefix tokens are cached as the KV cache to save the redundant computation (Shi et al., 2024).

FFN Module. The FFN module consists of two linear layers and one activation layer, structured as:

$$\mathbf{FFN}(\mathbf{X}) = \mathsf{Linear}_{h_2}(\mathsf{ACT}(\mathsf{Linear}_{h_1}(\mathbf{X}))), \quad (2)$$

where ACT is the activation function, such as GELU and ReLU. Similar to the attention module, the output incorporates a residual connection and a normalization layer.

3 Problem Formulation

Notations. We use bold lowercase letters to represent vectors, such as the input of the l_{th} linear layer $\mathbf{x}^{(l)}$, and the output probability distribution $\mathbf{y}$. The bold uppercase letters represent the matrix, such as the weights of the l_{th} linear layer $\mathbf{W}^{(l)}$. The permutation matrix is a square matrix obtained by permuting the rows (or columns) of the identity matrix (Strang, 2022). Shuffling columns of a vector or matrix is represented by multiplying a permutation matrix π on the right, denoted as $\mathsf{sf}(\mathbf{x}, \pi) = \mathbf{x}\pi$. For simplicity, we denote it as $\mathsf{sf}(\mathbf{x})$ when the specific permutation is not important in the context.

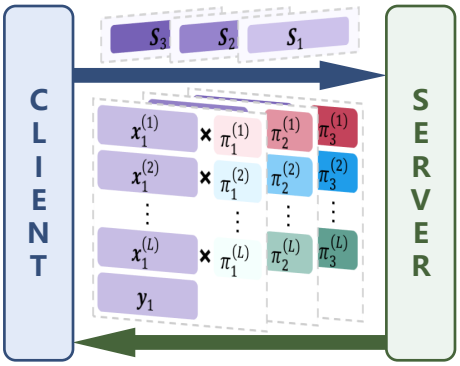

Figure 2: The interface of LOE inference.

MPC Setup. We follow the semi-honest threat model, which is widely used in both FME and LOE secure inference (Mohassel and Rindal, 2018; Zheng et al., 2022; Liu et al., 2024; Pang et al., 2024; Hao et al., 2022). In this threat model, the client and server are curious about each other's information but strictly adhere to cryptographic protocols. The shuffling defense in LOE inference works in either a 2-party or 3-party setting and can be generalized to different cryptographic protocols. The proposed attack is also applicable to different numbers of parties and cryptographic protocols. Later discussion focuses on the 2-party setting and protocols commonly used in prior LOE works as an example (Liu et al., 2024; Zheng et al., 2024).

Threat Model. We use the GPT to illustrate our attack. We define the LOE inference's interface to the model by $\mathcal{O}$, as illustrated in Figure 2. It functions as $\left[\mathsf{sf}(\mathbf{x}_i^{(1)}, \pi_i^{(1)}), \mathsf{sf}(\mathbf{x}_i^{(2)}, \pi_i^{(2)}), \cdots, \mathbf{y}_i\right] = \mathcal{O}(\mathbf{S}_i)$, where i is the query index. The input $\mathbf{S}$ is a prefix that includes the client prompt and previously generated tokens. The client obtains the shuffled activation vector $\mathsf{sf}(\mathbf{x}_i^{(l)}, \pi_i^{(l)})$ of all linear layers, along with the final probability $\mathbf{y}_i$ of the generated token. *Note that for activation vectors at the same layer depth, the applied permutations are random for different model inputs.* Other requirements follow those commonly used in existing model attacks, such as allowing free choice of input and permitting a mild number of queries.

4 Extraction Attack of the LOE Inference

Section 4.1 introduces the proposed attack that aligns distinct shuffled activation vectors without knowing the actual permutation. Furthermore, Section 4.2 addresses the unique challenges associated

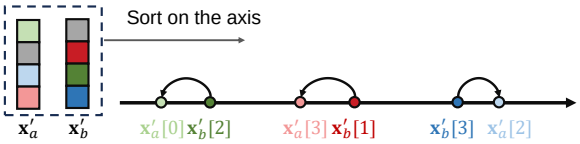

Figure 3: Aligning two shuffled vectors based on the closest distance. Three out of four pairs of elements are drawn to illustrate their correspondence for two sufficiently close vectors.

with alignment in Transformer models. Finally, Section 4.3 demonstrates that the aligned vectors enable the extraction of model weights that perform the same functionality as the original model. The complete attacking algorithm is in Appendix A.

4.1 Breaking shuffling by Vector Alignment

Alignment Definition. Suppose the adversarial client holds two distinct shuffled activation vectors $\mathbf{x}'_a = \mathsf{sf}(\mathbf{x}_a, \pi_a)$ and $\mathbf{x}'_b = \mathsf{sf}(\mathbf{x}_b, \pi_b)$. The objective of the alignment is to rearrange the elements in vectors such that the corresponding elements are matched up, even though the adversary remains unaware of specific permutations π_a and π_b. The alignment can be mathematically formalized as transforming the permutation π_b of $\mathbf{x}'_b$ to π_a of $\mathbf{x}'_a$ through a transformation matrix $\mathbf{M}$, which is also a permutation matrix (Strang, 2022), such that $\pi_a = \pi_b * \mathbf{M}$ or $\mathsf{sf}(\mathbf{x}_b, \pi_a) = \mathsf{sf}(\mathbf{x}_b, \pi_b) * \mathbf{M}$.

Aligning Close Activation Vectors. Directly guessing the transformation matrix $\mathbf{M}$ of two unrelated activation vectors requires enumerating all possible permutation matrices. This is impractical and the adversary cannot verify whether a transformation matrix is correct. In contrast, the proposed attack considers the alignment of vectors $\mathbf{x}_a$ and $\mathbf{x}_b$ that are ϵ-close, such that $\|\mathbf{x}_a - \mathbf{x}_b\| < \epsilon$. With the closely aligned activation vectors, the proposed attack solves $\mathbf{M}$ by leveraging the proximity between corresponding elements, as illustrated in Figure 3. The elements of the two shuffled vectors are positioned along the axis, with similar colors indicating correspondences between elements. Since shuffling preserves the original values, when the two original vectors are sufficiently close, the distance between corresponding elements in the shuffled vectors remains much smaller than the distance between non-corresponding elements. With this property, solving the transformation matrix $\mathbf{M}$ is equivalent to minimizing the following objective

$$\min_{\mathbf{M}} \|\mathbf{x}'_a - \mathbf{x}'_b * \mathbf{M}\|_2. \tag{3}$$

An extreme case is when the two ϵ-close model inputs have $\epsilon = 0$, enabling trivial alignment by matching elements with identical values. In general cases when $\epsilon > 0$, solving $\mathbf{M}$ using Equation (3) is difficult as the elements in $\mathbf{M}$ are discrete. Therefore, we reformulate it as follows and defer the proof of equivalence to Appendix B.

$$\begin{aligned} \min_{\mathbf{M}} & \sum_{i,j \in \mathcal{H}} \mathbf{M}[i,j] \odot \mathbf{D}[i,j] \\ \text{s.t.} & \sum_{i \in \mathcal{H}} \mathbf{M}[i,j] = 1 \text{ for } j \in \mathcal{H}, \\ & \sum_{j \in \mathcal{H}} \mathbf{M}[i,j] = 1 \text{ for } i \in \mathcal{H} \end{aligned} \tag{4}$$

The operator $\odot$ denotes the Hadamard product. Suppose both vectors contain h elements, and $\mathcal{H} = \{1, 2, \cdots h\}$ is the set of possible values of i and j. $\mathbf{D} \in \mathbb{R}^{h \times h}$ is a matrix whose element $\mathbf{D}[i,j] = (\mathbf{x}'_a[i] - \mathbf{x}'_b[j])^2$. The constraint is because $\mathbf{M}$ is a permutation matrix. Each element $\mathbf{M}[i,j] \in \{0,1\}$ can be interpreted as whether the i_{th} element of $\mathbf{x}'_b$ corresponds to the j_{th} element of $\mathbf{x}'_a$. This is a standard assignment problem that can be optimally solved using the Hungarian algorithm (Crouse, 2016).

4.2 Vector Alignment in GPT

This section details how the proposed alignment works in a real Transformer model. We describe the alignment of differently shuffled activation vectors at a certain layer depth, and the procedure is equally applicable to activations at other layer depths.

Constructing Close Activation vectors. To construct the close intermediate activation vectors required by the proposed alignment, we exploit the Lipschitz continuity of neural network models (Virmaux and Scaman, 2018). Given two ϵ-close model inputs $\mathbf{S}_1$ and $\mathbf{S}_2$, such that $\|\mathbf{S}_1 - \mathbf{S}_2\| < \epsilon$, the following inequality holds:

$$\|\mathbf{x}_a - \mathbf{x}_b\| \leq \mathcal{L}\|S_a - S_b\| < \mathcal{L}\epsilon, \tag{5}$$

where $\mathcal{L}$ is the Lipschitz constant. It bounds the difference between intermediate activations $\mathbf{x}_1$ and $\mathbf{x}_2$ based on the difference between the model inputs $\mathbf{S}_1$ and $\mathbf{S}_2$.

In this way, the adversarial client initiates the attack by querying the Transformer model with n model inputs that are ϵ-close to each other. When generating the next token, at a certain layer depth, the client acquires n activation vectors $\mathsf{sf}(\mathbf{x}_1, \pi_1), \mathsf{sf}(\mathbf{x}_2, \pi_2), \cdots, \mathsf{sf}(\mathbf{x}_n, \pi_n)$ that exhibit similar magnitudes ($\|x_i - x_j\| < \mathcal{L}\epsilon$ for

any $i, j \in \{1, 2, \cdots, n\}$) but distinct permutations. The client then selects an arbitrary $k \in \{1, \cdots, n\}$ and uses permutation π_k of $\mathsf{sf}(\mathbf{x}_k, \pi_k)$ as the reference. Aligning the remaining vectors to the chosen vector can be represented as $\mathsf{sf}(\mathbf{x}_1, \pi_1), \mathsf{sf}(\mathbf{x}_2, \pi_2), \cdots, \mathsf{sf}(\mathbf{x}_n, \pi_n) \xrightarrow{Alignment} \mathsf{sf}(\mathbf{x}_1, \pi_k), \mathsf{sf}(\mathbf{x}_2, \pi_k), \cdots, \mathsf{sf}(\mathbf{x}_n, \pi_k)$. The vectors aligned with the common permutation π_k are organized into an matrix $\mathbf{X}\pi_k$ with n rows, which is later used for model weight extraction. It is crucial to emphasize that the specific permutation π_k remains unknown to the client. The extraction of model weights does not require explicit knowledge of π_k, which we explain in Section 4.3.

Challenge of Constructing Close Inputs in GPT. The alignment process in Transformer models presents unique challenges in constructing ϵ-close model inputs. Transformer models accept discrete token indices as inputs, which are transformed into floating-point hidden vectors through the embedding layer. This constraint fails the adversary's ability to manipulate model inputs freely to achieve the desired ϵ-close conditions. Moreover, directly manipulating intermediate activations and redistributing them among parties is infeasible, as this approach violates the semi-honest threat model assumed in existing LOE frameworks, thus make the attack trivial.

To overcome this limitation, we exploit a seemingly innocuous property of secure inference to construct close intermediate activations. To execute secure inference protocols, floating-point values in Transformer models are encoded as fixed-point values on the ring, and a truncation step is necessary to reset the fixed-point precision after multiplication. To maintain model accuracy, secure inference frameworks typically employ high precision; for example, the SecretFlow-SPU framework (Ma et al., 2023) defaults to 18-bit precision. We notice existing secure truncation protocols introduce a stochastic 1-bit error (Mohassel and Rindal, 2018; Rathee et al., 2020; Ma et al., 2023). Then such 1-bit error manifests as a perturbation of magnitude 2^{-18}, which we strategically exploit to generate close intermediate activations. Consequently, as the adversary repeatedly feeds the same prompt, the truncation process inherently introduces the desired minor perturbations into the intermediate activations.

4.3 Exposing Model Weights

With the aligned activation vectors, this section explains how the adversary derives weights of linear layers. For the l_{th} linear layer, its weights satisfy the following equation

$$\mathbf{X}^{(l+1)} = \mathbf{X}^{(l)}\mathbf{W}^{(l)}, \tag{6}$$

where $\mathbf{W}^{(l)} \in \mathcal{R}^{h_{in} \times h_{out}}$ is the weight of l_{th} linear layer. The standard way to solve weight is through $\mathbf{W}^{(l)} = \mathbf{X}^{(l)^-}\mathbf{X}^{(l+1)}$, where $\mathbf{X}^{(l)^-}$ is the inverse of $\mathbf{X}^{(l)}$. To guarantee a meaningful inverse matrix, the adversary needs at least h_{in} input-output vector pairs (assuming full-rank weight matrix).

In the proposed attack, the adversary obtains inputs and outputs in the form of $\mathbf{X}^{(l)}\pi^{(l)}$ and $\mathbf{X}^{(l+1)}\pi^{(l+1)}$. Using these aligned matrices, the adversary extracts weights by

$$\mathbf{W}^{(l)'} = \pi^{(l)^-}\mathbf{X}^{(l)^-}\mathbf{X}^{(l+1)}\pi^{(l+1)} = \pi^{(l)^-}\mathbf{W}^{(l)}\pi^{(l+1)}. \tag{7}$$

The appendix D also addresses the case when the matrix multiplication of activations in the attention module is privately computed.

Functionality Invariance to Shuffling. In equation (7), the adversary's solved weights $\mathbf{W}^{(l)'}$ differ from the original weights $\mathbf{W}^{(l)}$ by row-wise and column-wise shuffling. Importantly, the stolen permuted weights perform the same functionality as the original weights. This is because the weight matrices of the neural network are inherently invariant to such shuffling (Carlini et al., 2020; Jagielski et al., 2020). For solved weight matrices of consecutive layers, as long as the output dimension of the former layer and the input dimension of the latter layer follow the same permutation, the functionality of forward propagation remains correct. We provide a detailed explanation in the Appendix C.

Handling the Ill-conditioned Problem. Although Equation (7) is theoretically solvable, practical computation introduces additional considerations. The proximity of the aligned activation vectors makes the computation of the inverse of the input matrix $\mathbf{X}^{(l)}\pi^{(l)}$ ill-conditioned (Neumaier, 1998). The ill-conditioning is measured by the condition number $\sigma_{max}/\sigma_{min}$, where σ_{max} and σ_{min} represent the maximum and minimum singular values of the input matrix. A large condition number indicates that numerical errors from the machine during computation are amplified, leading to significant inaccuracies when computing the matrix inverse. To mitigate such inaccuracies, the adversary needs

to manually set an upper bound $\mathcal{C}$ on the condition number. Consequently, when computing the pseudo-inverse of the input matrix $\mathbf{X}^{(l)}\pi^{(l)}$, singular values smaller than $\sigma_{max}/\mathcal{C}$ are discarded to ensure a stable approximation. In future work, the accuracy of the solution can be further enhanced by utilizing higher machine precision or adopting iterative methods (Richardson, 1911).

Query Complexity. The attack complexity of the query number is highly efficient at $O(N)$, where N is the largest rank of the weight matrix in the model, typically corresponding to the dimension of the FFN module. The efficient linear complexity is because 1) our attack finally reduces to solving a linear system, where the minimum number of required vectors exceeds the weight matrix rank, and 2) each query simultaneously gathers one activation vector for all layer depths.

5 Evaluation

5.1 Experimental Setup

Victim Transformer Models. We validate the proposed attack on two Transformer models: Pythia-70m (Biderman et al., 2023) and GPT-2 (Radford et al., 2019). These model scales have been widely employed in prior works on secure Transformer inference (Chen et al., 2022a; Luo et al., 2025; Pang et al., 2024; Li et al., 2024). We demonstrate the weight stealing results of all types of linear layers in the Transformer model, including $\mathbf{W}_{qkv}$, $\mathbf{W}_o$, $\mathbf{W}_{h1}$, and $\mathbf{W}_{h2}$.

Query Number. Theoretically, the number of required queries to compute the input matrix inverse is the weight's dimension. In our experiments, due to the ill-conditioned problem, we set the query number to 16 times the model's maximum dimension to improve the accuracy of the matrix inverse in numerical computation. The query count for Pythia-70m with the maximum dimension 2048 is 32768. The query count for GPT-2 with the maximum dimension 3072 is 49512. Most GPT APIs charge users on a per-token basis. Since our attack is independent of response length or content, the adversary can feed a query that receives very short responses, such as having the model respond only "yes" or "no". Using the commercial API pricing (OpenAI, 2025), the estimated costs for the attack on both Pythia-70m and GPT-2 remain under one dollar, demonstrating practical feasibility.

Fixed-point (FXP) Precision. In secure inference, fixed-point precision of the ring number is originally intended to ensure the fixed-point errors within the tolerance of the model. For such an innocuous property, our attack algorithm strategically exploits the fixed-point error to construct close activation vectors. Thus, the choice of fixed-point precision impacts the effectiveness of the attack algorithm. In the well-established secure inference framework, such as SecretFlow-SPU (Ma et al., 2023), the default precision of 18 bits is sufficient to maintain computational accuracy. To further evaluate the robustness of our attack, we examine smaller precisions, specifically 14 and 16 bits.

Table 1: For input and output of the layer, the table lists counts of correctly corresponded elements and mean squared errors of aligned vectors compared to the oracle-aligned vectors.

(Input,Output)	Fxp	Input		Output	
Dimension	Pre.	Count	Error	Count	Error
(512,2048)	14	508	9.4E-07	1996	1.2E-06
	16	511	1.2E-08	2040	9.1E-08
	18	512	0.0E+00	2046	4.0E-08
(2048,512)	14	2004	3.0E-06	504	5.9E-06
	16	2038	3.6E-07	510	1.2E-07
	18	2046	5.5E-08	511	1.5E-08
(768,3072)	14	756	6.0E-07	3052	1.2E-07
	16	766	2.4E-08	3066	6.4E-09
	18	766	2.5E-08	3070	3.7E-09
(3072,768)	14	3056	2.5E-07	762	5.8E-08
	16	3068	1.0E-08	764	1.9E-08
	18	3070	1.2E-08	764	5.0E-09

5.2 Alignment Error

Table 1 presents the counts of correctly corresponded elements and mean squared errors (MSE) compared to the oracle-aligned activation vectors. The evaluation focuses on the Linear_{h1} and Linear_{h2} of Pythia-70m and GPT-2 at a certain depth. They are selected because their input dimensions represent the maximum and minimum cases among all linear layer types, corresponding to the highest and lowest difficulty. The results are averaged across the results of all obtained activation vectors. For all precisions, less than 2% of the elements are mismatched, with higher fixed-point precision resulting in a higher correct alignment ratio. This is due to increased precision reducing perturbations in the intermediate activations, enabling more accurate element matching. The magnitudes of MSE range from 10^{-9} to 10^{-6}, primarily caused by rare mismatches and occur only when the mismatched element is close to the correct corresponding element. This accurate alignment builds a solid

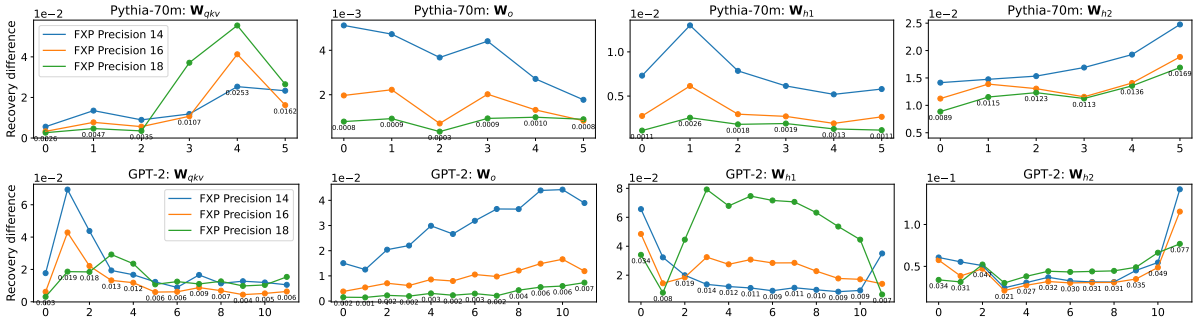

Figure 4: The attacking results on Pythia-70m and GPT-2. The x-axis indicates the depth of the decoders, and the y-axis indicates the L1-norm difference between the original and extracted weights (up to a row and column shuffling). Different colors correspond to different fixed-point precisions.

foundation for later extracting weights.

5.3 Results on Extracted Model Weights

Figure 4 shows the L1-norm difference between the original and extracted weights. The extracted weights differ from the original ones by a random permutation of a row and column, hence we report the difference between each value in the extracted weights and its counterpart in the original weights. The results indicate our attack successfully extracts weights with minor errors. This provides an accurate approximation of the original weights for malicious usage. For example, the extracted weights can be directly used to analyze model information or as an initialization for training the surrogate model. Below, we analyze the attack results in detail. In most cases, our attack successfully extracts weights with L1-norm difference ranging from 10^{-4} to 10^{-2}. We find the smaller model Pythia-70m consistently exhibits lower L1-norm differences compared to GPT-2. A common intuition regarding neural network attacks is that the difficulty of the attack increases with model size (Carlini et al., 2020, 2024). The same principle applies to our attack, as the difficulty to extract a weight depends on its input dimension. Similarly, the L1-norm differences for $\mathbf{W}_{h1}$ with input dimension of d_{model} are generally smaller than that for $\mathbf{W}_{h2}$ with input dimension of $4 * d_{model}$.

Higher precision generally results in smaller L1-norm differences. This is because higher precision leads to more accurate alignment, which benefits solving the linear system. However, there are cases where a lower precision of 14 results in smaller errors. This occurs because lower precision means more diverse vectors in the input matrix and relieves the ill condition of the problem, leading to improved approximations of the matrix inverse when numerically solving the linear system.

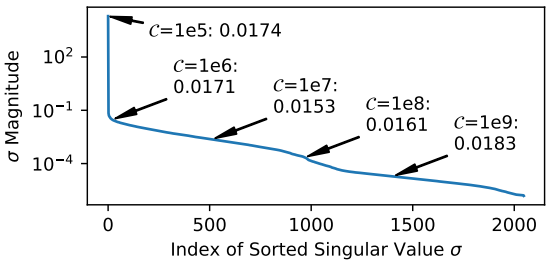

Figure 5: The curve indicates the magnitude of sorted singular values. We annotate five thresholds of the condition number, with the corresponding L1-norm differences of the solved model weights.

5.4 Singular Value Analysis

As Section 4.3 shows, the proximity in activation matrices results in a high condition number $\sigma_{max}/\sigma_{min}$, which means inaccuracies of matrix inverse and solved weights. To deal with it, an upper threshold of the condition number is set as $\mathcal{C}$, meaning that singular values σ lower than $\sigma_{max}/\mathcal{C}$ are truncated. Figure 5 shows the magnitude of the sorted singular values of the certain layer of Pythia-70m. Five thresholds (1e5, 1e6, 1e7, 1e8, and 1e9) and the corresponding L1-norm difference of the solved model weights compared to the original weight are annotated on the curve. As the threshold is gradually increased to retain more singular values, the L1-norm difference decreases. However, further increasing of the threshold causes the difference to grow instead, as numerical errors begin to counteract the benefits. We choose $\mathcal{C}$ as 1e7 throughout the experiments, which we find yields the most accurate solution in most cases.

Table 2: Perplexity on the Wikitext dataset. Lower perplexity indicates better performance.

	Original Model	Stolen Model	Finetuned Stolen Model
Pythia-70m	31.81	44.46	32.43
GPT-2	21.11	47.92	21.15

5.5 Perplexity of Extracted Model Weights

To evaluate how the recovered weights perform on real tasks, we compare the perplexity of the extracted model weights with that of the victim models on the wikitext dataset (Merity et al., 2017).

Our results show that the extracted weights can already serve as an effective proxy model for downstream black-box attacks. After a short fine-tuning stage (at most 6 minutes), the recovered model closely matches the victim model's perplexity and can effectively substitute it. Such near-lossless recovery substantially increases the attack severity, as it enables the attacker to obtain a model that behaves almost identically to the original and transform the following attacks into a white-box manner.

6 Related Work

Cryptography-based Secure Inference of Transformer Models. Due to the rapidly growing concerns about data privacy in Transformer-based applications, numerous works have investigated two-party secure inference for the Transformer model. Secure computation of nonlinear layers is significantly slower than that of linear layers (Li et al., 2023; Chen et al., 2022a; Zheng et al., 2022; Hao et al., 2022; Zhang et al., 2024), and thus becomes the major performance bottleneck. This is because evaluating a nonlinear layer generally involves polynomial approximations or complex protocols, necessitating multiple communication rounds and extensive data transmission. The optimization of full-model encryption inference can be categorized into two main aspects as follows.

One type of work optimizes cryptographic protocols to improve the efficiency of secure inference. CryptFlow2 (Rathee et al., 2020) proposes a faster millionaire protocol to reduce the communication size of the comparison operation. Some methods (Rathee et al., 2021; Gupta et al., 2023; Pang et al., 2024) optimize secure lookup tables. Another type of work modifies the Transformer model to tailor the cryptographic protocols. Since accurate piecewise polynomials to approximate the nonlinear layers (Dong et al., 2023; Lu et al., 2023) is costly, some works (Chen et al., 2022b; Zeng et al., 2023; Li et al., 2023) use aggressive approximations of Softmax and GELU. However, such aggressive approximations lead to noticeable accuracy loss, even when employing knowledge distillation to mitigate the decline in accuracy. Later works adopt fully homomorphic encryption and use GPU acceleration (Zhang et al., 2025; Park et al., 2025; Moon et al., 2025). Despite these optimizations, nonlinear layers remain the main bottleneck.

Plaintext Nonlinear Layer Computation. Due to the nonlinear layers being inherently unfriendly to cryptographic protocols, a workaround is to let the client evaluate the nonlinear layer in plaintext (Zhang et al., 2018; Xie et al., 2019; Wong et al., 2020; Zheng et al., 2022; Liu et al., 2024; Zheng et al., 2024). These works allow the client to access intermediate activations for evaluating nonlinear layers. To prevent the exploitation of such activations, early approaches (Zhang et al., 2018; Xie et al., 2019) restrict query access or employ Bayesian neural networks; however, they remain vulnerable to extraction attacks given sufficient queries (Wong et al., 2020). Subsequent works (Zheng et al., 2022; Liu et al., 2024; Zheng et al., 2024; Luo et al., 2025) observe that nonlinear computations are invariant to the positions of values in the activation tensor, and thus propose revealing shuffled activations to the client. Although shuffling disrupts activation structure and is widely considered secure in practice, this work demonstrates an attack that can still recover model weights (up to symmetries).

7 Conclusion

Shuffling defense in the LOE inference demonstrates great potential for balancing security and performance within the Transformer model. This work revisits the shuffling defense by introducing an attack that aligns shuffled activations and subsequently extracts model weights. Through both theoretical analysis and experimental results, we demonstrate that our method can successfully extract model weights, with the L1-norm difference from the original weights ranging from 10^{-4} to 10^{-2}. By addressing these issues early, we aim to understand the limitations of the shuffling defense and contribute to its improvement, ultimately enhancing its security and reliability.

Limitations

While our approach theoretically guarantees the feasibility of parameter extraction and has been successfully validated on small-scale models, it remains subject to the inherent limitations of neural network-based attacks with respect to model scale. Specifically, as the target model size increases, the fidelity of the extracted parameters tends to degrade—resulting in reduced recovery accuracy. However, it is important to emphasize that the goal of this work is not to facilitate malicious model extraction, but rather to expose previously overlooked vulnerabilities in existing defense mechanisms. From this perspective, successful validation on small-scale models is sufficient to fulfill our objective.

Acknowledgements

This work was supported by the Shanghai Committee of Science and Technology, China (Grant No. 24BC3200900), the Shanghai Academy of Future Internet Technology, the National Natural Science Foundation of China (NSFC) (Grant No. 62532006), and the Shanghai Qi Zhi Institute Innovation Program (SQZ202316).

References

Stella Biderman, Hailey Schoelkopf, Quentin Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, Aviya Skowron, Lintang Sutawika, and Oskar Van Der Wal. 2023. Pythia: a suite for analyzing large language models across training and scaling. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org.

Nicholas Carlini, Matthew Jagielski, and Ilya Mironov. 2020. Cryptanalytic extraction of neural network models. In *Annual International Cryptology Conference*, pages 189–218. Springer.

Nicholas Carlini, Daniel Paleka, Krishnamurthy Dj Dvijotham, Thomas Steinke, Jonathan Hayase, A Feder Cooper, Katherine Lee, Matthew Jagielski, Milad Nasr, Arthur Conmy, and 1 others. 2024. Stealing part of a production language model. *arXiv preprint arXiv:2403.06634*.

Tianyu Chen, Hangbo Bao, Shaohan Huang, Li Dong, Binxing Jiao, Daxin Jiang, Haoyi Zhou, Jianxin Li, and Furu Wei. 2022a. The-x: Privacy-preserving transformer inference with homomorphic encryption. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 3510–3520.

Tianyu Chen, Hangbo Bao, Shaohan Huang, Li Dong, Binxing Jiao, Daxin Jiang, Haoyi Zhou, Jianxin Li, and Furu Wei. 2022b. The-x: Privacy-preserving transformer inference with homomorphic encryption. *arXiv preprint arXiv:2206.00216*.

Minsu Cho, Ameya Joshi, Brandon Reagen, Siddharth Garg, and Chinmay Hegde. 2022. Selective network linearization for efficient private inference. In *International Conference on Machine Learning*, pages 3947–3961. PMLR.

Vanya Cohen and Aaron Gokaslan. 2020. Opengpt-2: Open language models and implications of generated text. *XRDS: Crossroads, The ACM Magazine for Students*, 27(1):26–30.

David F Crouse. 2016. On implementing 2d rectangular assignment algorithms. *IEEE Transactions on Aerospace and Electronic Systems*, 52(4):1679–1696.

Ye Dong, Wen-jie Lu, Yancheng Zheng, Haoqi Wu, Derun Zhao, Jin Tan, Zhicong Huang, Cheng Hong, Tao Wei, and Wenguang Cheng. 2023. Puma: Secure inference of llama-7b in five minutes. *arXiv preprint arXiv:2307.12533*.

Kanav Gupta, Neha Jawalkar, Ananta Mukherjee, Nishanth Chandran, Divya Gupta, Ashish Panwar, and Rahul Sharma. 2023. Sigma: Secure gpt inference with function secret sharing. *Cryptology ePrint Archive*.

Meng Hao, Hongwei Li, Hanxiao Chen, Pengzhi Xing, Guowen Xu, and Tianwei Zhang. 2022. Iron: Private inference on transformers. *Advances in Neural Information Processing Systems*, 35:15718–15731.

Zhicong Huang, Wen-jie Lu, Cheng Hong, and Jiansheng Ding. 2022. Cheetah: Lean and fast secure two-party deep neural network inference. *IACR Cryptol. ePrint Arch.*, 2022:207.

Mitsuru Ito, Akira Saito, and Takao Nishizeki. 1989. Secret sharing scheme realizing general access structure. *Electronics and Communications in Japan (Part III: Fundamental Electronic Science)*, 72(9):56–64.

Matthew Jagielski, Nicholas Carlini, David Berthelot, Alex Kurakin, and Nicolas Papernot. 2020. High accuracy and high fidelity extraction of neural networks. In *Proceedings of the 29th USENIX Conference on Security Symposium*, pages 1345–1362.

Andes YL Kei and Sherman SM Chow. 2025. Shaft: Secure, handy, accurate, and fast transformer inference. In *Network and Distributed System Security Symposium, NDSS*, volume 2025.

Brian Knott, Shobha Venkataraman, Awni Hannun, Shubho Sengupta, Mark Ibrahim, and Laurens van der Maaten. 2021. Crypten: Secure multi-party computation meets machine learning. *Advances in Neural Information Processing Systems*, 34:4961–4973.

Dacheng Li, Hongyi Wang, Rulin Shao, Han Guo, Eric Xing, and Hao Zhang. 2023. MPCFORMER: FAST, PERFORMANT AND PRIVATE TRANSFORMER INFERENCE WITH MPC. In *The Eleventh International Conference on Learning Representations*.

Zhengyi Li, Kang Yang, Jin Tan, Wen-jie Lu, Haoqi Wu, Xiao Wang, Yu Yu, Derun Zhao, Yancheng Zheng, Minyi Guo, and 1 others. 2024. Nimbus: Secure and efficient two-party inference for transformers. *arXiv preprint arXiv:2411.15707*.

Qingxiu Liu, Qun Huang, Xiang Chen, Sa Wang, Wenhao Wang, Shujie Han, and Patrick PC Lee. 2024. Pp-stream: Toward high-performance privacy-preserving neural network inference via distributed stream processing. In *Proceedings of the 40th IEEE International Conference on Data Engineering (ICDE 2024)*.

Wen-jie Lu, Zhicong Huang, Zhen Gu, Jingyu Li, Jian Liu, Kui Ren, Cheng Hong, Tao Wei, and WenGuang Chen. 2023. Bumblebee: Secure two-party inference framework for large transformers. *Cryptology ePrint Archive*.

Jinglong Luo, Guanzhong Chen, Yehong Zhang, Shiyu Liu, Hui Wang, Yue Yu, Xun Zhou, Yuan Qi, and Zenglin Xu. 2025. Centaur: bridging the impossible trinity of privacy, efficiency, and performance in privacy-preserving transformer inference. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 22751–22770.

Junming Ma, Yancheng Zheng, Jun Feng, Derun Zhao, Haoqi Wu, Wenjing Fang, Jin Tan, Chaofan Yu, Benyu Zhang, and Lei Wang. 2023. {SecretFlow-SPU}: A performant and {User-Friendly} framework for {Privacy-Preserving} machine learning. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, pages 17–33.

Casey Meehan, Amrita Roy Chowdhury, Kamalika Chaudhuri, and Somesh Jha. 2021. Privacy implications of shuffling. In *International Conference on Learning Representations*.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2017. Pointer sentinel mixture models. In *International Conference on Learning Representations*.

Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. 2020. Delphi: A cryptographic inference service for neural networks. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 2505–2522.

Payman Mohassel and Peter Rindal. 2018. Aby3: A mixed protocol framework for machine learning. In *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, pages 35–52.

Jungho Moon, Dongwoo Yoo, Xiaoqian Jiang, and Miran Kim. 2025. Thor: Secure transformer inference with homomorphic encryption. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security*, pages 3765–3779.

Arnold Neumaier. 1998. Solving ill-conditioned and singular linear systems: A tutorial on regularization. *SIAM review*, 40(3):636–666.

OpenAI. 2025. Api pricing.

Qi Pang, Jinhao Zhu, Helen Möllering, Wenting Zheng, and Thomas Schneider. 2024. Bolt: Privacy-preserving, accurate and efficient inference for transformers. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 130–130. IEEE Computer Society.

Dongjin Park, Eunsang Lee, and Joon-Woo Lee. 2025. Powerformer: Efficient and high-accuracy privacy-preserving language model with homomorphic encryption. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11090–11111.

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

Deevashwer Rathee, Mayank Rathee, Rahul Kranti Kiran Goli, Divya Gupta, Rahul Sharma, Nishanth Chandran, and Aseem Rastogi. 2021. Sirnn: A math library for secure rnn inference. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1003–1020. IEEE.

Deevashwer Rathee, Mayank Rathee, Nishant Kumar, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. 2020. Cryptflow2: Practical 2-party secure inference. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 325–342.

Lewis Fry Richardson. 1911. Ix. the approximate arithmetical solution by finite differences of physical problems involving differential equations, with an application to the stresses in a masonry dam. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, 210(459-470):307–357.

Tianxiang Shen, Ji Qi, Jianyu Jiang, Xian Wang, Siyuan Wen, Xusheng Chen, Shixiong Zhao, Sen Wang, Li Chen, Xiapu Luo, and 1 others. 2022. {SOTER}: Guarding black-box inference for general neural networks at the edge. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*, pages 723–738.

Luohe Shi, Hongyi Zhang, Yao Yao, Zuchao Li, and Hai Zhao. 2024. Keep the cost down: A review on methods to optimize llm's kv-cache consumption. *arXiv preprint arXiv:2407.18003*.

Gilbert Strang. 2022. *Introduction to linear algebra*. SIAM.

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.

Aladin Virmaux and Kevin Scaman. 2018. Lipschitz regularity of deep neural networks: analysis and efficient estimation. *Advances in Neural Information Processing Systems*, 31.

Harry W. H. Wong, Jack P. K. Ma, Donald P. H. Wong, Lucien K. L. Ng, and Sherman S. M. Chow. 2020. Learning model with error – exposing the hidden model of bayhenn. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 3529–3535. International Joint Conferences on Artificial Intelligence Organization. Main track.

Peichen Xie, Bingzhe Wu, and Guangyu Sun. 2019. Bayhenn: Combining bayesian deep learning and homomorphic encryption for secure dnn inference. *arXiv preprint arXiv:1906.00639*.

Mu Yuan, Lan Zhang, and Xiang-Yang Li. 2023. Secure transformer inference. *arXiv preprint arXiv:2312.00025*.

Wenxuan Zeng, Meng Li, Wenjie Xiong, Tong Tong, Wen-jie Lu, Jin Tan, Runsheng Wang, and Ru Huang. 2023. Mpcvit: Searching for accurate and efficient mpc-friendly vision transformer with heterogeneous attention. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5052–5063.

Jiawen Zhang, Xinpeng Yang, Lipeng He, Kejia Chen, Wen-jie Lu, Yinghao Wang, Xiaoyang Hou, Jian Liu, Kui Ren, and Xiaohu Yang. 2025. Secure transformer inference made non-interactive. In *NDSS*.

Qiao Zhang, Cong Wang, Hongyi Wu, Chunsheng Xin, and Tran V Phuong. 2018. Gelu-net: A globally encrypted, locally unencrypted deep neural network for privacy-preserved learning. In *IJCAI*, pages 3933–3939.

Yancheng Zhang, Mengxin Zheng, Yuzhang Shang, Xun Chen, and Qian Lou. 2024. Heprune: Fast private training of deep neural networks with encrypted data pruning. *Advances in Neural Information Processing Systems*, 37:51063–51084.

Fei Zheng, Chaochao Chen, Zhongxuan Han, and Xiaolin Zheng. 2024. Permllm: Private inference of large language models within 3 seconds under wan. *arXiv preprint arXiv:2405.18744*.

Fei Zheng, Chaochao Chen, Xiaolin Zheng, and Mingjie Zhu. 2022. Towards secure and practical machine learning via secret sharing and random permutation. *Knowledge-Based Systems*, 245:108609.

Algorithm 1 Weight Extraction Attack

Input: Sequence S;

{Prepare aligned dataset}

1: Initialize n as an appropriate value greater than hidden dimension h;

2: **for** $i = 1$ to n **do**

3: $\mathsf{sf}(\mathbf{x}_i^{(1)}, \pi_1^{(i)}), \mathsf{sf}(\mathbf{x}_i^{(2)}, \pi_2^{(i)}), \cdots, \mathbf{y}_i^{(L)} = \mathcal{O}_\pi(S)$;

4: **end for**

5: **for** $l = 1$ to $L - 1$ **do**

6: Align the dataset $\{\mathsf{sf}(\mathbf{x}_i^{(l)}, \pi_i^{(l)})\}$ using $\pi_k^{(l)}$ as the reference permutation and obtain $\{\mathsf{sf}(\mathbf{x}_i^{(l)}, \pi_k^{(l)})\}$;

7: $\mathbf{X}^{(l)}\pi_k^{(l)} = concate(\{\{\mathsf{sf}(\mathbf{x}_i^{(l)}, \pi_k^{(l)})\})$;

8: Compute the pseudo-inverse $\pi_k^{(l)^-}\mathbf{X}^{(l)^-}$;

9: $\mathbf{W}^{(l)'} = \pi_k^{(l)^-}\mathbf{X}^{(l)^-}\mathbf{X}^{(l+1)}\pi_k^{(l+1)} = \pi_k^{(l)^-}\mathbf{W}^{(l)}\pi_k^{(l+1)}$, where $\mathbf{W}^{(l)}$ is the original weights;

10: **end for**

Output: Weight set $\{\mathbf{W}^{(l)'}\}$ of all linear layers;

A Appendix

A Complete Attack Algorithm

With the insight of breaking shuffling defense and constructing proximate inputs through random truncation error, we formally describe the attack algorithm that can extract the model weights of the Transformer model.

The adversary first decides the number of queries n. Note that the query number should be at least the greatest size of the model, i.e., $4*d_{model}$, which is the dimension of the FFN module. This is because solving the linear system requires computing the inverse of the aligned input vectors. Thus, n should be greater than $4*d_{model}$ for accurate matrix inverse (line 1).

Then, the adversary keeps feeding the model with the same prompt S. Due to the secure truncation, in operations such as the LayerNorm in the embedding layer, a small perturbation is introduced to the activation vectors, which the adversary expects to construct close activations. In this way, the client collects a dataset that includes the shuffled intermediate activations of all linear layers $\{\mathsf{sf}(\mathbf{x}_i^{(l)}, \pi_i^{(l)})\}$, for $i \in [n]$ and $l \in [L]$ (lines 2-4). Note that the model may generate many tokens in a response. We only collect tokens at the same generation step.

With the dataset, the adversary can align them using the method in Section 4. For each layer, the adversary arbitrarily selects the permutation scheme of a vector as the reference, which we refer to as $\pi_k^{(l)}$, and then aligns the other vectors according to $\pi_k^{(l)}$. The aligned vectors are then concatenated as a matrix. The model weights can be computed using the inverse of the input matrix (lines 5-10). Although the obtained weights $\mathbf{W}^{(l)'}$ differ from the original weights $\mathbf{W}^{(l)}$ up to a row and column shuffling and this shuffling is unknown to the adversary, the solved weights still perform the original functionality as the consecutive layer adopts the same shuffling scheme.

B Proof of the Equivalence between Equation (3) and Equation (4)

This section demonstrates the equivalence between Equation (3) and Equation (4). We first recall some notations. $\mathbf{x}'_a, \mathbf{x}'_b \in \mathbb{R}^h$ are two differently shuffled but value-close activation vectors. $\mathbf{M} \in \{0,1\}^{h\times h}$ is a permutation matrix that has only single one in each row or column.

We begin from the Equation (3)

$$\|\mathbf{x}'_a - \mathbf{x}'_b\mathbf{M}\|_2^2 = \sum_{i=1}^{h}\left(\mathbf{x}'_a[i] - \sum_{j=1}^{h}\mathbf{x}'_b[j]\mathbf{M}[j,i]\right)^2. \tag{8}$$

Since $\mathbf{M}$ only has single one in each column, we have $\sum_{j=1}^{h}\mathbf{x}'_b[j]\mathbf{M}[j,i] = \mathbf{x}'_b[\sigma(i)]$, where $\sigma(i)$ is an index function indicates the row index of 1 in i_{th} column of $\mathbf{M}$. Then Equation (8) becomes

$$\begin{aligned}
&\sum_{i=1}^{h}\left(\mathbf{x}'_a[i] - \mathbf{x}'_b[\sigma(i)]\right)^2 \\
=&\sum_{i=1}^{h}\left[\left(\mathbf{x}'_a[i] - \mathbf{x}'_b[\sigma(i)]\right)^2 + 0 * \sum_{j=1,j\neq i}^{h}\left(\mathbf{x}'_a[j] - \mathbf{x}'_b[\sigma(j)]\right)^2\right] \\
=&\sum_{i,j\in\mathcal{H}}\mathbf{M}[i,j]\cdot(\mathbf{x}'_a[i] - \mathbf{x}'_b[j])^2.
\end{aligned} \tag{9}$$

This is exact the objective $\sum_{i,j\in\mathcal{H}}\mathbf{M}[i,j]\odot\mathbf{D}[i,j]$ in Equation (4). □

C Equivalent Functionality of Shuffled Weights

As stated in Section 4.3, for shuffled weights of consecutive layers, functional equivalence with the original weights is preserved if the output dimension of the preceding layer and the input dimension of the subsequent layer adhere to the same permutation. We use two consecutive layers to illustrate this and the same idea can be extended to the entire model.

Let $\mathbf{W}^{(l)}$ and $\mathbf{W}^{(l+1)}$ denote the weights of two consecutive layers. For a given input vector $\mathbf{x}^{(l)}$ to layer l, the forward propagation is expressed as

$$\mathbf{x}^{(l+2)} = g(\mathbf{x}^{(l)}\mathbf{W}^{(l)})\mathbf{W}^{(l+1)} \tag{10}$$

The $g(\cdot)$ are nonlinear layers, such as the GELU, Softmax, or Layernorm.

In the proposed attack, the solved weights differ from the original weights up to a row-wise and column-wise shuffling, which are $\mathbf{W}^{(l)'} = \pi^{(l)^-}\mathbf{W}^{(l)}\pi^{(l+1)}$ and $\mathbf{W}^{(l+1)'} = \pi^{(l+1)^-}\mathbf{W}^{(l+1)}\pi^{(l+2)}$. The input to $\mathbf{W}^{(l)}$ is $\mathbf{x}^{(l)}\pi^{(l)}$ since the output channel of prior weights $\mathbf{W}^{(l-1)'} = \pi^{(l-1)^-}\mathbf{W}^{(l-1)}\pi^{(l)}$ is also shuffled. Then the forward propagation using the equivalent weights is

$$\begin{aligned}
&g(\mathbf{x}^{(l)}\pi^{(l)}\mathbf{W}^{(l)'})\mathbf{W}^{(l+1)'} \\
=&g(\mathbf{x}^{(l)}\pi^{(l)}\pi^{(l)^-}\mathbf{W}^{(l)}\pi^{(l+1)})\pi^{(l+1)^-}\mathbf{W}^{(l+1)}\pi^{(l+2)} \\
=&g(\mathbf{x}^{(l)}\mathbf{W}^{(l)})\pi^{(l+1)}\pi^{(l+1)^-}\mathbf{W}^{(l+1)}\pi^{(l+2)} \\
=&g(\mathbf{x}^{(l)}\mathbf{W}^{(l)})\mathbf{W}^{(l+1)}\pi^{(l+2)} \\
=&\mathbf{x}^{(l+2)}\pi^{(l+2)}
\end{aligned} \tag{11}$$

The equality in the third line is because the permutation invariance property of the activation function $g(\cdot)$, allowing the permutation matrix $\pi^{(l+1)}$ to be factored out. Subsequently, the permuted output $\mathbf{x}^{(l+2)}\pi^{(l+2)}$ serves as input to the next shuffled weight matrix $\mathbf{W}^{(l+2)'} = \pi^{(l+2)^{-1}}\mathbf{W}^{(l+2)}\pi^{(l+3)}$. This formulation ensures that the same permutation between the output dimension of $\mathbf{W}^{(l)}$ and the input dimension of $\mathbf{W}^{(l+1)}$ preserves the functional equivalence of the shuffled weight matrices.

D Dealing with the Multi-head Attention

Recovering weights becomes more complicated when the matrix multiplication of activations in the attention module is privately computed in some studies (Zheng et al., 2024). In such a case, although the adversary cannot directly obtain the input and output of the linear layer, the activation matrix multiplication still preserves the linear system, allowing for the derivation of equivalent weights that function identically to the original linear layers. However, the multi-head attention mechanism increases the complexities of the linear system, as detailed below. For ease of discussion, we first define additional notations. The weight of layer

$Linear_{qkv}$ is decomposed into weights $\mathbf{W}_q$, $\mathbf{W}_k$, and $\mathbf{W}_v$. For the token being generated, the input and output of $\mathbf{W}_q$ are $\mathbf{x}$ and $\mathbf{q}$. For prompt and already generated tokens, the input matrix of $\mathbf{W}_k$ and $\mathbf{W}_v$ are denoted by $\mathbf{X}_{pre}$ with output $\mathbf{K}$ and $\mathbf{V}$, which are known as the KV cache.

Equivalent Weights of $\mathbf{W}_q$ and $\mathbf{W}_k$. For the h_{th} attention head, we define the equivalent weights $\mathbf{W}_{qk}^h = \mathbf{W}_q^h \mathbf{W}_k^{h^T}$. Different heads represent the independent sub-problems that can be solved in the same way. The adversary holds the input $\mathbf{x}$ to $\mathbf{W}_q$, the prefix input $\mathbf{X}_{pre}$ to $\mathbf{W}_k$, and the input $\mathbf{s}$ of the softmax (override the notation meaning model input in prior sections). According to Equation (1), their relationship is formulated as $\mathbf{s}^h = \mathbf{x}\mathbf{W}_q^h \mathbf{W}_k^{h^T} \mathbf{X}_{pre}^T$. The problem is still a linear problem through the transformation:

$$\mathbf{s}^h = \mathbf{x}\mathbf{W}_{qk}^h \mathbf{X}_{pre}^T = (\mathbf{X}_{pre} \otimes \mathbf{x})^T \, \mathrm{vec}(\mathbf{W}_{qk}^h), \tag{12}$$

where the $\otimes$ is the Kronecker product and $\mathrm{vec}(\cdot)$ means vectorizing the matrix. The adversary can first compute the Kronecker product, resulting in a matrix with a hidden dimension equal to the square of the model's hidden dimension. Then, it becomes the same linear system as Equation (6) to solve the equivalent weight $\mathbf{W}_{qk}^h$.

Equivalent Weights of $\mathbf{W}_v$ and $\mathbf{W}_o$. The equivalent weight for $\mathbf{W}_v$ and $\mathbf{W}_o$ is given by $\mathbf{W}_{vo} = \begin{bmatrix} \mathbf{W}_v^1 \mathbf{W}_o^1 \\ \mathbf{W}_v^2 \mathbf{W}_o^2 \\ \cdots \\ \mathbf{W}_v^H \mathbf{W}_o^H \end{bmatrix}$, where the superscripts indicate the head index. In this case, the adversary holds the prefix input $\mathbf{X}_{pre}$ to $\mathbf{W}_v$, the softmax output $\mathbf{p}$, and the $\mathbf{W}_o$ output $\mathbf{o}$. Due to the multi-head mechanism, the original relationship $\mathbf{o} = \mathbf{p} \cdot \mathbf{X}_{pre}\mathbf{W}_v \cdot \mathbf{W}_o$ in Equation (1) transforms into:

$$\begin{aligned} \mathbf{o} &= \text{ Concat } (\mathbf{head}_1, \ldots, \mathbf{head}_H) * \mathbf{W}_o \\ &= \left[\mathbf{p}^1 \mathbf{X}_{pre} \mathbf{W}_v^1 \mid \mathbf{p}^2 \mathbf{X}_{pre} \mathbf{W}_v^2 \mid \ldots \mid \mathbf{p}^H \mathbf{X}_{pre} \mathbf{W}_v^H\right] * \mathbf{W}_o \\ &= \left[\mathbf{p}^1 \mathbf{X}_{pre} \mid \mathbf{p}^2 \mathbf{X}_{pre} \mid \ldots \mid \mathbf{p}^H \mathbf{X}_{pre}\right] * \begin{bmatrix} \mathbf{W}_v^1 \mathbf{W}_o^1 \\ \mathbf{W}_v^2 \mathbf{W}_o^2 \\ \cdots \\ \mathbf{W}_v^H \mathbf{W}_o^H \end{bmatrix} \end{aligned} \tag{13}$$

Note that the $\left[\mathbf{p}^1 \mathbf{X}_{pre} \mid \ldots \mid \mathbf{p}^H \mathbf{X}_{pre}\right]$ is a vector whose dimension is H times the model dimension. Finally, the equivalent weight $\mathbf{W}_{vo}$ is solved in the same way as Section 4.3.

Evaluation. Figure 6 shows the results when the activation matrix multiplications are privately computed. Results are presented only for the equivalent weight $\mathbf{W}_{vo}$. The L1-norm differences for Pythia-70m are smaller than 0.004 and the differences for GPT-2 are smaller than 0.14. The larger differences in GPT-2 are due to its relatively higher number of attention heads (12), making the linear system more challenging to solve. Recovering equivalent weights $\mathbf{W}_{qk}$ remains a challenge. Computing the Kronecker product makes the input dimension in Equation (12) become the square of the model dimension, such as 768^2. We leave this for future work.

E Possible Defenses

Our attack identifies the weaknesses of the shuffling defense in the existing LOE inference. We also briefly list corresponding defense that can mitigate the proposed attack.

Noise Addition. The first defense strategy is to increase the alignment error by adding noise. Adding noise is a common technique used in neural network defense to obscure leaked information. However, noise addition is generally constrained within a threshold to prevent degradation of model accuracy, meaning this approach can only partially mitigate the attack.

Partial Plaintext Nonlinear Layer Computation. The second strategy is to make only part of the nonlinear layers publicly computed. We observe that the primary computational cost mainly arises from the Softmax and GELU layers, while the Layernorm is much faster. This is because the nonlinear operation, the reciprocal square root, of the Layernorm only computes once per vector (Lu et al., 2023; Pang et al., 2024). The defender can tolerate a slight increase in latency to make the Layernorm is computed privately. This approach ensures that for all types of linear layers in the Transformer decoder, either the input or output remains unknown to the adversary, making weight recovery impossible.

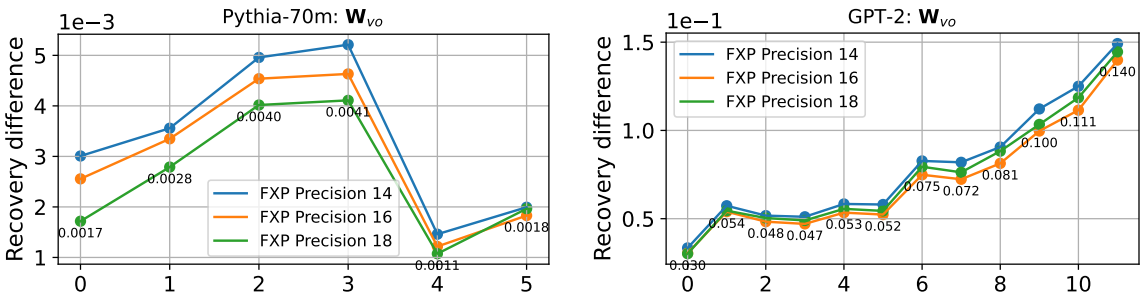

Figure 6: The attacking results on the equivalent weights $\mathbf{W}_{vo}$.