STIP: Secure Three-Party Transformer Inference Protocol with Lossless Accuracy and Millisecond-Level Latency

Mu Yuan[1,2], Lan Zhang[1], Guoliang Xing[2], Xiang-Yang Li[1]
[1]*University of Science and Technology of China*
[2]*The Chinese University of Hong Kong*

Abstract

Security of model parameters and user data is critical for Transformer-based services, such as ChatGPT. While recent strides in secure two-party protocols have successfully addressed security concerns in serving Transformer models, their adoption is practically infeasible due to the prohibitive cryptographic overheads involved. Drawing insights from our hands-on experience in developing two real-world Transformer-based services, we identify the inherent efficiency bottleneck in the two-party assumption. To overcome this limitation, we propose a novel three-party threat model that consists of model developer, model server, and data owner. Based on this framework, we design a semi-symmetric permutation-based protection scheme and present STIP, the first secure Transformer inference protocol without any inference accuracy loss. We analyze STIP's resistance to brute force, known-plaintext, and social engineering attacks and prove the privacy leakage upper bound using distance correlation. And we propose a method to integrate the trusted execution environment with STIP to make model parameters resistant to model extraction and fine-tuning attacks. Experiments on six representative series of Transformer models, with up to 70 billion parameters, in real systems show that STIP has strong security and no loss of accuracy. For auto-regressive token generation, STIP achieves 31.7 ms latency for LLaMA2-7b model, significantly reducing the 5-minute overhead of the state-of-the-art two-party protocols.

1 Introduction

Transformer inference-based services are the most cutting-edge artificial intelligence applications [13, 24, 53]. Cloud computing capabilities, such as auto-scaling [54], meet the requirements of serving Transformers, especially large models with billions of parameters. Therefore, major organizations like OpenAI opt for full-cloud deployment for their Transformer-based services [5]. Nevertheless, sending raw data to the cloud is infeasible in privacy-sensitive domains, as illustrated by incidents such as Samsung's prohibition of ChatGPT use after a sensitive code leak [22].

Model split is not secure enough. Model split inference [33,59,70] strategically distributes neural network layers between the device and the cloud. The device sends intermediate activations to the cloud to continue inference. Model split inference avoids raw data transmission while maintaining efficiency [47, 48]. However, concerns arise as research reveals the potential for reverse-engineering sensitive information from intermediate activations [2, 46].

Secure two-party protocols incur prohibitive overheads. Recent studies explore secure Transformer inference through homomorphic encryption (HE) and secure two-party computation (2PC) [11, 17, 26, 28, 40, 67]. However, these protocols incur prohibitive computational and device-cloud communication overheads, especially with non-linear complex layers like LayerNorm and ReLU. For example, CipherGPT costs a 25-minute processing time and 90 GiB traffic for generating a single token with GPT2 [28] while Puma takes around 5 minutes and 2 GiB communication for LLaMA2-7b [17].

First principles thinking: three-party threat model. To overcome the efficiency barrier, we use first principles to rethink the basic two-party assumption: model owner and data owner. Drawing insights from managing two real-world Transformer-based services, a consistent experience emerged: *model developer* $\neq$ *model server*. For both services, we developed Transformer models by fine-tuning [29] open-sourced parameters [39, 60] with collected data. We have adequate computing power for offline model development but lack the capability for large-scale, long-term services to hundreds of thousands of users. Consequently, as model developers, we rely on third-party cloud platforms to serve our models. In line with our real-world development experiences, we propose a new three-party threat model. Within this model, we decompose the model owner into two entities: model developer and model server. As the developed models are proprietary, model developers must safeguard their model parameters against potential attacks from model servers [72], therefore we assume they do not collude.

STIP insights and design. Based on our introduced three-party threat model, we developed STIP, Secure Transformer Inference Protocol, with two insights. First, we employ efficient feature-space permutation for secure and equivalent Transformer inference. Since the inference is executed on the untrusted server, model parameters and on-device data must be transformed before uploading to the cloud. Based on efficient permutation in the feature space, we design a data and parameter transformation approach for Transformer layers. We prove the mathematical equivalence of computation using our proposed transformation, thus ensuring no loss of accuracy. Second, we design a semi-symmetrical protection scheme between the model developer and data owners. This insight stems from the sequential structure of neural networks. We reveal that the model developer only needs to share identical permutations in the first and last layers with the data owner, and can exclusively retain the information of intermediate layer transformation. Similar semi-symmetric protection schemes have found application in diverse domains, such as image encryption [16] and online shopping [21]. We demonstrate the privacy-preserving capability of STIP based on distance correlation [58] and prove its resistance to brute-force and known-plaintext attacks.

Contributions. We summarize three key contributions of this work as follows:

• We identify the efficiency bottleneck inherent in the two-party setting and its misalignment with real-world applications. We propose a new three-party threat model, decomposing the model owner into two distinct entities: model developer and model server.

• We present STIP, the first secure protocol for three-party Transformer inference, with the theoretical bound of privacy leakage and guarantee of no loss of accuracy.

• We implement STIP and conduct evaluations on various Transformer models with up to 70 billion parameters on real systems. Experimental results verify the security of STIP against brute force, known plaintext, model extraction, and fine-tuning attacks. Our evaluation also showcases the efficiency of STIP, comparable to unprotected full-cloud inference, surpassing state-of-the-art secure two-party protocols [11, 17, 26, 28, 40] by millions of times.

This work does not raise any ethical issues.

2 Background

2.1 Device-Cloud Collaboration

Transformer inference-based services have become the most compelling artificial intelligence applications, e.g., ChatGPT sets a record for the fastest-growing user base [53].

Full-cloud inference: efficient but unsafe. Serving Transformers, especially large models with billions of parameters, aligns with cloud computing capabilities, like auto-scaling [54]. Several cloud-native frameworks for Transformer inference have been released, like NVIDIA NeMo [35] and Microsoft DeepSpeed [4]. Therefore, most organizations, including OpenAI, adopt full-cloud deployment for their Transformer-based services [5]. However, sending raw data to the cloud is infeasible for various privacy-sensitive areas, e.g., Samsung officially banned employees from using ChatGPT after sensitive code was leaked [22].

Full-device inference: secure but not scalable. To address data privacy concerns, another option is deploying Transformer models entirely on the device. Through model compression techniques like weight quantization [14], Transformer models with billions of parameters can run inference on devices [10]. However, the scalability of full-device deployment is limited. First, FLOPs (floating point operations) and memory footprints scale linearly with the number of parameters, while on-device computing resources grow much slower than the explosive size of Transformer models [20, 74]. Second, model compression inevitably brings accuracy loss [65], but in the highly competitive large Transformer market, even a slight QoS degradation may result in lagging behind competing products. [30].

Device-cloud collaboration: best of both worlds. Since neither the cloud nor the device can satisfactorily serve Transformer models independently, device-cloud collaboration has naturally emerged in recent research efforts [33, 59, 70]. By reasonably assigning inference workloads between the device and the cloud, collaborative inference avoids transmitting raw data while maintaining efficiency [47, 48].

2.2 Secure Two-Party Inference

Not transmitting raw data is just a bottom line for security. Research on attacks against deep neural networks shows that intermediate activations (e.g., text embeddings [46]) can be reverse-engineered to disclose the sensitive information in raw data [2]. Tighter security is urgently needed.

HE and 2PC for Transformer inference. Homomorphic encryption (HE) is a cryptographic technique that allows computations to be performed on encrypted data without decrypting it, while multi-party computation (MPC) allows multiple parties to jointly compute a function over their inputs while keeping those inputs private. In the context of model inference, two-party computation (2PC) is usually considered, a special case of MPC where the model owner and the data owner represent two parties respectively. Recent research has demonstrated the ability to serve Transformer inference using a combination of HE and 2PC [11, 26, 28]. However, these protocols incur significant computational overhead when processing non-linear complex layers, such as LayerNorm and ReLU. Additionally, there are high costs associated with device-cloud communications, e.g., CipherGPT takes over 25 minutes to generate a single token with GPT2 model [28].

2.3 Align with Real-World Applications: Three-Party Threat Model

The simplicity of the two-party setting, where one party represents the device and the other the cloud, seamlessly fits HE and 2PC theories. However, the efficiency challenges also stem from the computational hardness inherent in HE and MPC theories [49], which motivates us to think about a question: *Does the two-party setting truly align with the demands of real-world applications?*

Surprisingly but fortunately, the answer is no. This conclusion comes from our experience of managing two real-world Transformer-based services.

Service-1: campus security chatbot. At ANONYMOUS University, we host a large language model-based chabot for campus security. Our chatbot uses the database of surveillance video analytics as the information source. Users, including students and campus security officers, can ask questions like "Did any abnormal behavior occur during a certain period?" to the chatbot and get responses in natural language.

Service-2: vehicle cabin assistant. At ANONYMOUS automotive company, we deploy a multi-modal Transformer model to enhance the functionality of the smart assistant in vehicle cabins. The multi-modal Transformer takes the in-cabin video frames as the input and generates scene descriptions in natural language. Scene descriptions can help the in-car assistant become more user-friendly, such as recommending music based on facial expressions.

Common experience: the model developer is not the model server. For both services, we developed the Transformer model by fine-tuning [29] open-sourced parameters [19, 60] with our collected data. The computing power of our on-campus lab and company can afford offline model development, but cannot support large-scale long-term services. There are approximately a few hundred users of the campus chatbot, and the vehicle assistants need to serve hundreds of thousands of users. We, as model developers, need to resort to third-party cloud computing platforms to serve our Transformer models. In fact, this experience is not just for us, but also for other model development companies such as OpenAI. OpenAI uses the Microsoft Azure cloud platform to provide ChatGPT services to hundreds of millions of users [1, 5].

Three-party threat model. To align with experience from developing real-world services, we introduce a three-party threat model, as shown in Fig. 1. Departing from the classic two-party setting, we divide the model owner into two distinct entities: the model developer and the model server. Given that the developed models are proprietary, model developers must safeguard their model parameters against attacks from model servers [72]. We make the assumption that model developers do not engage in collusion with cloud platforms. Any collaboration between these entities would lead to a regression to the classic two-party setting. This decomposition of the model owner role not only enhances practicality but also relaxes adversary assumptions. This adjustment proves instrumental in overcoming the computational hardness inherent in two-party protocols.

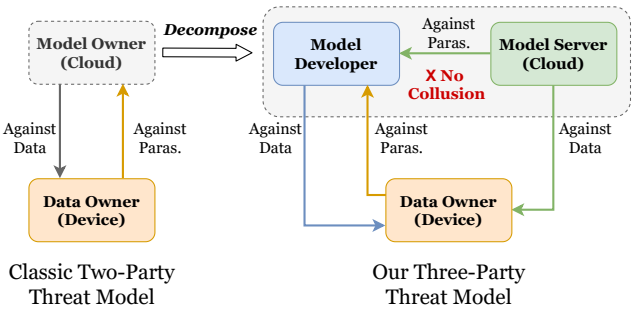


Figure 1: Three-Party Threat Model

2.4 Design Space

Scope. This work focuses on achieving efficient and secure execution of Transformer model inference, i.e., the end-to-end forward pass, within the three-party setting. Transformer inference results can serve as a foundation for downstream applications, e.g., AI agents that automatically call external APIs [55]. Potential security risks and efficiency issues in downstream applications are out of the scope of this work.

Design goals. Our protocol has four main design goals:
(1) Data and parameter security. The foremost objective is to ensure the security of on-device data and model parameters.
(2) No accuracy loss. The protocol is required to perform accuracy-lossless inference, meaning there should be no approximation of any computations in Transformer models.
(3) Support production environments. It is crucial that the protocol supports inference frameworks used in production environments, incorporating techniques such as kv-cache for efficiency optimization [4, 35].
(4) Flexible extension to Transformer variants. Given the continuous evolution of Transformer models with various emerging variants [6, 32, 39, 52, 60], the protocol must possess the ability for flexible extension to accommodate Transformer variants. This ensures long-term availability without necessitating case-by-case adaptation.

Tab. 1 shows the comparison of STIP and existing Transformer inference methods on our four design goals.

3 Challenges

In designing STIP that achieves both efficiency and security, we encountered two non-trivial challenges.

3.1 Prohibitive Cryptographic Overhead

Transformation of data and parameters is key to protection. Existing protocols that combine HE and 2PC techniques for

Table 1: Comparison of our proposed STIP and existing Transformer inference methods.

Method	Secure	No Loss	Prd. Env.	Tested Models
Full-Cloud	✗	✓	✓	All
Iron [26]	✓	✗	✗	BERT series
THE-X [11]	✓	✗	✗	BERT-Tiny
CipherGPT [28]	✓	✗	✗	GPT-2
Bumblebee [40]	✓	✗	✗	GPT-2/LLaMA
Puma [17]	✓	✗	✗	GPT-2/LLaMA
STIP	✓	✓	✓	GPT/LLaMA/ ViT/LLaVA/ BERT/Mixtral

security have prohibitive computing and communication overheads. As shown in Fig. 2, CipherGPT [28] takes over 25 minutes and 90 GiB traffic to perform a single forward pass of GPT2 [52] with 123 million parameters.

The success of the Transformer model hinges on the utilization of global matrix multiplication in its self-attention and feedforward modules, making it highly parallelizable compared to recursive architectures [56]. The inductive bias of the Transformer architecture is not only efficient at the implementation level but also has some properties such as permutation symmetries of hidden units [3] and the token-wise permutation invariance [36]. Drawing attention to the prevalent use of random permutation in addressing privacy and security concerns, including secure communication [7], property inference [23], machine learning [27, 41, 75], etc. In light of these observations, we present our first insight:

▷ ***Insight 1: Efficient feature-space permutation for transformed and equivalent Transformer inference.***

Since the inference is executed on the untrusted server, the parameters from the model developer and the on-device data must be transformed before uploading to the cloud. We design a data and parameter transformation specifically for Transformer layers based on random permutation in the feature space. The transformation can be efficiently implemented with $O(d)$-complexity movement of memory pointers, where d is the feature dimension. As Fig. 2 shows, our protocol, STIP , achieves orders of magnitude higher efficiency than CipherGPT, and the latency is close to full-cloud deployment. We prove the mathematical equivalence of computation using our proposed transformation, thus ensuring no loss of accuracy (§ 5.1).

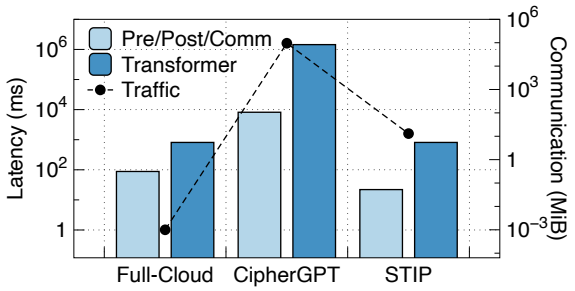


Figure 2: Latency and communication overheads of GPT2-124m model.

3.2 Attack Surface Vulnerability

While random permutation-based schemes are efficient and accuracy-lossless, endowing them with robust security poses non-trivial challenges. A direct adoption of sequence-level permutation scheme [36] leads to $n!$ possible permutations, rendering it inadequate for safeguarding data against brute-force attacks (BFA) when the number of input tokens n is small. Opting for permutation in the feature space can enhance protection, yet using a single permutation matrix π remains vulnerable to known-plaintext attacks (KPA). The reason is, once the cloud gains a pair of known plaintext of on-device data and the transformed data, it can easily recover the permutation matrix, subsequently inverse-transforming all parameters and exposing sensitive information. Tab. 2 summarizes the number of possible permutations for data and parameters and resistance to BFA and KPA attacks.

▷ ***Insight 2: Semi-symmetrical set of permutation matrices between the model developer and data owners.***

This insight stems from the sequential structure of neural networks. The data owner only needs to share identical permutations in the first and last layers with the model developer. Intermediate layer transformation information can be exclusively retained by the model developer. Consequently, we propose a feature-space permutation scheme utilizing a set of matrices $\pi_1, \ldots, \pi_{3L}$, where L represents the number of layers. Similar semi-symmetric protection schemes have been explored in domains such as image encryption [16] and online shopping [21]. We analyze the privacy-preserving capability of our proposed scheme based on distance correlation [58] and prove its resistance to BFA and KPA (§ 5.3).

4 Definition

With these insights in mind, in order to formally present mathematically equivalent transformations and analyze theoretical security, this section defines Transformer inference (§ 4.1) and the three-party threat model (§ 4.2).

Table 2: Comparison of different permutation-based schemes in terms of number of possibilities and resistance to attacks.

Protection Scheme	Data	Paras.	BFA	KPA
Seq. Perm.	$n!$	1	✗	✗
Feat. Perm. with Single π	$d!$	$d!$	✓	✗
Feat. Perm. with $\{\pi_1, ..., \pi_{3L}\}$	$d!$	$(d!)^{3L}$	✓	✓

4.1 Transformer Inference

We use the original Transformer architecture [61] to introduce the inference workflow, without loss of generality, advanced Transformer variants (GPT [51], LLaMA [60], ViT [18] and Mixtral [32]) will be discussed in Sec. 6.

Device-cloud model split In Transformer models, the embedding operation is the initial step that maps discrete inputs (such as words or images) into continuous vectors [61]. In order to decouple our protocol from the original data modality, we set the starting point of device-cloud collaborative inference to the embeddings instead of the input. By default, only the embedding operation executes on the device while expensive Transformer layers and the classifier are deployed on the cloud, as shown in Fig. 3.

Transformer layer forward pass. As shown in Fig. 3, the on-cloud Transformer model consists of L sequential Transformer layers and a classifier. Let F_θ denote the Transformer model with trainable parameters θ. We define $\{f_i : \mathbb{R}^{n\times d} \mapsto \mathbb{R}^{n\times d} | i \in [L]\}$ as Transformer layers, and $f_c : \mathbb{R}^{n\times d} \mapsto \mathbb{R}^{n\times s}$ as the classifier, where n is the sequence length (e.g., the number of tokens), d is the model feature dimension, and s is the output dimension (e.g., vocabulary size). We use x_i and y_i to denote the input and output of the i-th Transformer layer, and all these intermediate activations share the same shape $\mathbb{R}^{n\times d}$. A forward pass of a Transformer layer, i.e., $f(x) = y$, is computed as follows[1]:
Self-attention sub-block:

$$Q = xW_q, \quad K = xW_k, \quad V = xW_v, \qquad W_q, W_k, W_v \in \mathbb{R}^{d\times d},$$
$$u = \text{SoftMax}\left(\frac{QK^T}{\sqrt{k}} + M\right)VW_o, \quad M \in \mathbb{R}^{n\times n}, W_o \in \mathbb{R}^{d\times d},$$
$$v = \text{LayerNorm}(u + x; \gamma_1, \beta_1), \qquad \gamma_1, \beta_1 \in \mathbb{R}^d,$$

Feedforward sub-block:

$$z = \text{ReLU}(vW_1)W_2, \qquad W_1 \in \mathbb{R}^{d\times m}, W_2 \in \mathbb{R}^{m\times d},$$
$$y = \text{LayerNorm}(z + v; \gamma_2, \beta_2), \qquad \gamma_2, \beta_2 \in \mathbb{R}^d,$$

where k and m are constants that depend on the model architecture hyperparameters, and M denotes the mask matrix. SoftMax, LayerNorm, and ReLU are commonly used neural network functions and their definitions are not necessary in this work. Following the L-layer Transformers, a classifier computes as follows:

$$o = y_L W_c, \qquad W_c \in \mathbb{R}^{d\times s}.$$

[1]For simplicity of expression, we use xW instead of xW^T which is used for real-world implementation.

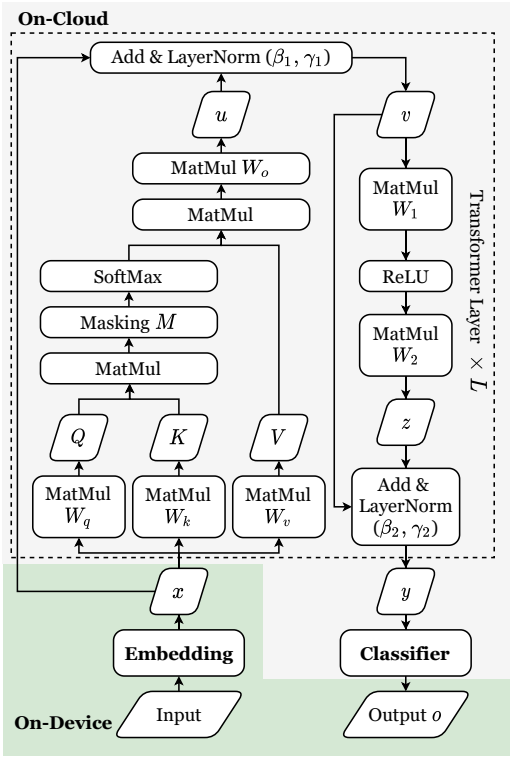


Figure 3: Inference workflow of original Transformer.

Use of masking. The mask matrix is a lower triangular matrix, where the elements above the main diagonal are set to negative infinity, and the elements on and below the main diagonal are set to zero. In the original Transformer work [61], two types of Transformer layers are proposed, encoder and decoder. The mask is only applied to the self-attention sub-block in the decoder to prevent positions after the current position from being attended to. It ensures that during the generation of each token in the output sequence, the model only attends to the tokens preceding it. Masking is not trivial, as it results in the infeasibility of constructing equivalent computations using sequence-level permutation (§ 5.1).

4.2 Three-Party Setting and Threat Model

For serving Transformer models, we consider three parties:
• Model developer (P_1) that trains and owns private Transformer model parameters, e.g., OpenAI developed GPT4.
• Model server (P_2) that has the computing hardware, e.g., high-end GPUs on Azure cloud platform.
• Data owner (P_3) that own private input and inference output, e.g., text prompts and responses of ChatGPT users.

The inference protocol should ensure that P_1 and P_2 are unaware of P_3's input x_1 and inference output o. And P_1's model parameters θ should be hidden to both P_2 and P_3. The parameter θ consists of attention weights (W_q, W_k, W_v, W_o), feedforward weights (W_1, W_2), LayerNorm weights (γ, β), and classifier weights (W_c). In our context of Transformer inference, P_3's input can be text prompt [60], images [19], and a combination of multiple modalities [39], and the inference output is a probability vector of the last classification head [74].

We adopt the widely used semi-honest setting [11, 26, 28, 75] where each party will follow the protocol specifications but attempt to infer additional sensitive information from the observed protocol messages.

5 Design

This section first introduces how to perform equivalent inference of Transformer models with feature space permutation (§ 5.1). Then we present, Secure Transformer Inference Protocol (STIP), the core algorithm (§ 5.2), and analyze the protocol security (§ 5.3).

5.1 Feature Space Permutation

The permutation operation is defined by a permutation matrix π, which is a square binary matrix that has exactly one entry of 1 in each row and each column with all other entries of 0. For $x \in \mathbb{R}^{n\times d}$, $\pi_{n\times n} x$ and $x\pi_{d\times d}$ performs sequence-level and feature-level permutation respectively.

Mask makes sequence-level permutation not equivalent. For the Transformer encoder (self-attention without masking), the sequence-level permutation equivariance property, i.e., $f(\pi x) = \pi f(x)$, has been studied [36, 66]. However, due to the mask inside the decoder, attention computation on sequence-level permuted data cannot return equivalent output. A quick fix is to send a permuted $M' = \pi M \pi^T$ to the cloud computing platform. However, since the value structure of M is known, the cloud platform can easily infer π, which will result in a loss of protection.

Parameter transformation. Instead, we propose to transform parameters in the feature space with a set of random permutation matrices. First, we generate $\pi \in \{0,1\}^{d\times d}$ for the input x. For the i-th Transformer layer, we transform parameters with another three matrices $\pi_{i,1}, \pi_{i,2}, \pi_{i,3}$:

$$
\begin{aligned}
&W_q' = \pi^T W_q \pi_{i,1}, \quad W_k' = \pi^T W_k \pi_{i,1}, \quad W_v' = \pi^T W_v \pi_{i,2}, \\
&W_o' = \pi_{i,2}^T W_o \pi, \quad W_1' = \pi^T W_1 \pi_{i,3}, \quad W_2' = \pi_{i,3}^T W_2 \pi, \\
&\gamma_1' = \gamma_1 \pi, \quad \beta_1' = \beta_1 \pi, \quad \gamma_2' = \gamma_2 \pi, \quad \beta_2' = \beta_2 \pi.
\end{aligned}
$$

For the classifier, we need to generate a permutation matrix $\pi_c \in \{0,1\}^{s\times s}$. We transform the classifier parameters by

$$
W_c' = \pi^T W_c \pi_c.
$$

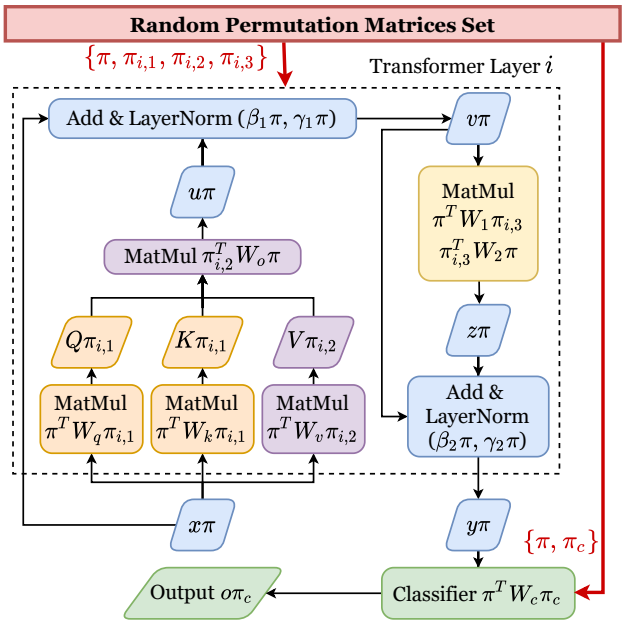


Figure 4: Feature-space parameter transformation. Colors represent the use of different permutation matrices.

Fig. 4 illustrates the parameter transformation procedure.

Computational equivalence. Let $F_{\theta'}$ denote the Transformer model with transformed parameters, we prove that original inference results can be recovered equivalently:

Theorem 1. $F_{\theta'}(x\pi)\pi_c^T = F_\theta(x)$.

Due to page limitations, the proof is placed in Appendix A.

5.2 Protocol

Based on our proposed permutation-based transformation for Transformer models, we develop STIP protocol. Fig. 5 shows an overview of STIP. STIP has two phases: initialization and inference. In the initialization phase, the model developer P_1 randomly generates the permutation matrices set $\Pi = \{\pi, \pi_c\} \cup \{\pi_{i,1}, \pi_{i,2}, \pi_{i,3} | i \in [L]\}$. P_1 transforms its owned trained model F_θ with Π and obtain the transformed version $F_{\theta'}$. Then P_1 sends the transformed model $F_{\theta'}$ to the cloud platform and distributes the permutation matrices for the input and output (π and π_c) to its registered users. Now the initialization phase finishes. For the inference phase, once a user wants to use the inference service, it runs the embedding on the device to obtain x. Then the user transforms the embedding using the received input permutation matrix π by a super-lightweight operation $x\pi = x'$. Then the user sends x' to the cloud. The workload on the cloud platform has no change, compared with the normal Transformer model serving. The cloud just performs the $F_{\theta'}(x')$ computations and obtains the output o' in the permuted feature space. Once the user receives the returned o' from the cloud, it simply reverse-transform

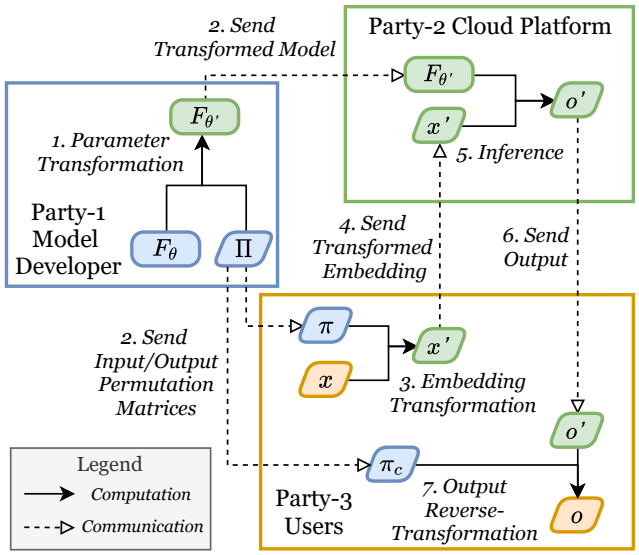


Figure 5: Overview of STIP.

Algorithm 1: Secure Transformer Inference Protocol

input : Number of Transformer Layers L

Initialization phase:

$[P_1]$ $\Pi \leftarrow \{\pi, \pi_c\} \cup \{\pi_{i,1}, \pi_{i,2}, \pi_{i,3} | i \in [L]\}$;

$[P_1]$ $F_{\theta'} \leftarrow$ `ParaTrans` (F_θ, Π);

$[P_1]$ `Send` $(F_{\theta'}, P_2)$ and `Send` $(\{\pi, \pi_c\}, P_3)$;

Inference phase:

$[P_3]$ $x' \leftarrow x\pi$ and `Send` (x', P_2);

$[P_2]$ $o' \leftarrow F_{\theta'}(x')$ and `Send` (o', P_3);

$[P_3]$ $o \leftarrow o'\pi_c^T$.

the output by $o = o'\pi_c^T$, which involves only memory movement operations and can be implemented super efficiently. Till now, one round of inference finished. Alg. 1 formally presents STIP protocol.

Production environment support. Production-level Transformer inference frameworks, such as DeepSpeed [4] and HuggingFace [64], incorporate many techniques to enhance efficiency. For example, they use KV-cache mechanism to alleviate redundant computations by storing intermediate results from previous attention calculations. STIP maintains a non-intrusive approach, transforming parameter values while keeping the underlying Transformer architecture unchanged. From the cloud perspective, STIP involves switching to a distinct set of weights, with no change in the inference computation process. As a result, STIP seamlessly aligns with production-level frameworks, and we have implemented STIP with the HuggingFace [64] library.

5.3 Security Analysis

Now we demonstrate that STIP can protect model parameters and user data from various attacks and quantify the bound of privacy leakage risk using distance correlation measure.

Random permutation resists brute-force attacks. To begin, let's consider P_1 as the attacker attempting to access user data x, o. Due to the inaccessible nature of $x\pi$ and $o\pi_c$, P_1 is unable to recover x, o with possessing π, π_c. Next, consider P_2 as the attacker targeting model parameters θ and user data x, o. Given that P_2 possesses permuted parameters and $x\pi$, the probability of correctly guessing $\pi_{d\times d}$ is $1/(d!)$, where d is typically larger than 512 in practical applications such as $d = 4096$ in LLaMA2-7b [60]. This renders the likelihood of a successful attack negligible. Notably, permutation-based protection schemes often exhibit a weakness in preserving the set of elements (e.g., English vocabulary) [75]. Fortunately, STIP avoids this vulnerability by applying permutation to intermediate activations rather than the original data. Thirdly, consider P_3 as the attacker against model parameters θ. Since P_3 lacks access to θ', it cannot recover θ despite having π, π_c. As STIP requires deploying the embedding model on the device, the weights of the embedding are exposed to P_3. However, the embedding module alone cannot perform valuable tasks and is therefore not sensitive (e.g., OpenAI has released its embedding module [45]).

Semi-symmetrical scheme to resist known-plaintext attack. A known-plaintext attack (KPA) is a cryptographic attack where the adversary possesses both the ciphertext (encrypted data) and the corresponding plaintext (unencrypted data). The goal of KPA is to uncover the encryption key or algorithm used to encrypt the data. In our context, if the plaintext of the model developer's parameters has been leaked, there is no need to continue attacking the protection scheme. Therefore, the focus of KPA consideration lies exclusively on user data. Assuming P_2 knows both x and $x\pi$, it can recover π with d times column matching, unless there are two or more exactly identical columns. Consequently, if parameters on the cloud rely solely on π for protection, all of them are at risk of being leaked. This underscores the rationale behind our adoption of a semi-symmetric protection scheme, wherein layer parameters are permuted using two matrices. One is exclusively owned by the model developer, while the other is shared with the user. This design in STIP makes the model parameters resistant to KPA. For a specific user, uncovering π would lead to all subsequent embeddings being exposed to P_2. To address this vulnerability, we implement a strategy of periodically changing the set of permutation matrices (in extreme cases, using one-time transformation), a practice commonly employed to resist KPA [8, 73].

Social engineering attack. Setting aside our semi-honest assumption, in a scenario where the cloud platform deceitfully pretends as a user and acquires the embedding model along with π, π_c, it can potentially uncover the data of other users

who share the same permutation matrices. To counteract this risk, the model developer can deploy multiple instances of the model, each employing distinct transformations. Users can then be randomly assigned to share a model instance (in extreme cases, each user may have an exclusive model instance), effectively mitigating the risk of data leakage through this social engineering attack. It's noteworthy that parameters remain resistant to this attack for the same reasons as the KPA we discussed above.

Distance correlation bound. Above we analyzed that STIP can ensure that data values cannot be uncovered. An important aspect to investigate is the degree of correlation between the original and permuted data. To quantify the risk of privacy leakage, we employ distance correlation [58], a statistical measure of dependency between two random vectors. Let Corr denote the distance correlation. Based on existing theorem [75], it has been proven that:

$$\mathop{\mathbb{E}}_{\pi_{d\times d}, A\in\mathbb{R}^{d\times d}}[\mathrm{Corr}(x, xA\pi)] \leq \mathop{\mathbb{E}}_{B\in\mathbb{R}^{d\times 1}}[\mathrm{Corr}(x, xB)].$$

In simpler terms, this implies that the privacy leakage resulting from the application of random permutation on intermediate activations is bounded by the leakage observed in one-dimensional reduction, which has demonstrated practical privacy-preserving capabilities [44, 63].

Model split considerations. By default, STIP only deploys the embedding module on user devices, as shown in Fig. 3. This decision is motivated by the fact that splitting the model before the embedding poses security challenges. In such a scenario, the device would be required to transmit tokenized one-hot vectors (a matrix $\in \mathbb{Z}^{n\times s}$, where s denotes the vocabulary size) to the cloud. While the matrix can be randomly permuted, the inherent one-hot nature of the vectors makes it susceptible to easy recovery of the permutation by the cloud. On the flip side, distributing more layers onto the device is also not a prudent choice. This is primarily because exposing additional parameters to end devices compromises efficiency.

5.4 TEE Integration for Enhanced Security

Now we discuss how to use TEE in STIP to further resist model extraction and finetuning-based attacks when the cloud and the user degenerate into one party.

Model functionality leakage. The above-mentioned STIP scheme utilizes random permutation and semi-symmetric mechanisms to prevent the cloud platform from obtaining the values of the original parameters. However, once the cloud obtains the capabilities of the user party (for example, by social engineering attacks), the models deployed on the cloud can still be used to perform inference computations. We refer to this risk as *model functionality leakage*.

Leverage TEE for authorized inference. Trusted execution environment (TEE) [31, 43] are secure areas within a processor that provide isolated environments for executing

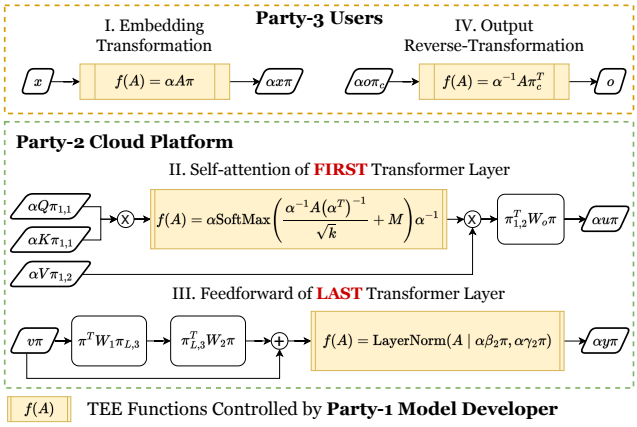


Figure 6: Integration of TEE into STIP protocol.

code and protecting sensitive data from unauthorized access. Due to the sequential nature of the Transformer's feedforward computation, authorization of inference can be implemented by deploying part of the model (such as the embedding and the classifier) into the TEE. Recent work explored utilizing parameter permutation and TEE to perform authorized Transformer inference [38]. However, our experiments (see Sec. 7.6) show that attackers can still use authorized input-output samples to perform model extraction [34] and finetuning attacks [37] on the parameters protected by TEE.

Introducing one-time randomness. Existing work [68, 76] has shown that model extraction attacks can be effectively resisted by introducing randomness on the input or output. Therefore, we design an approach based on left-multiplied random diagonal matrices to enhance the security of STIP against such attacks on model parameters. As shown in Fig. 6, we deploy four functions using enclave. Let A denote the input of the TEE functions controlled by the model developer. First, after getting the embedding, we additionally transform the embedding x using a one-time random diagonal matrix α: $f(A) = \alpha A\pi$. Second, in the first Transformer layer on the cloud, we put the masking and softmax operations into the TEE. Formally, we deploy the function as follows:

$$f(A) = \alpha\mathrm{SoftMax}(\frac{\alpha^{-1}A(\alpha^T)^{-1}}{\sqrt{k}} + M)\alpha^{-1}.$$

Third, in the last Transformer layer's feedforward sub-block, we put the LayerNorm operation into TEE for execution. Specifically, we deploy the following function: $f(A) = \mathrm{LayerNorm}(A|\alpha\beta_2\pi, \alpha\gamma_2\pi)$. Fourth, the user executes the reverse transformation of the inference result in TEE: $f(A) = \alpha^{-1}A\pi_c^T$. Let $F_{\theta',TEE}$ denote the Transformer model with transformed parameters and TEE integration. We prove that after integrating TEE, the computational equivalence property still holds:

Theorem 2. $\alpha^{-1}F_{\theta',TEE}(\alpha x\pi)\pi_c^T = F_\theta(x)$.

Due to page limitations, the proof is placed in Appendix A.

6 STIP for Transformer Variants

This section discusses how STIP supports various Transformer variants, including language models (§ 6.1), multi-modal models (§ 6.2), and mixture-of-experts models (§ 6.3). Following that, we establish generalized rules and claim the application scope of STIP (§6.4).
Due to page limitations, all proofs are placed in Appendix. A.

6.1 Language Models

Pre-LayerNorm. The first version of GPT [50] directly adopts the original Transformer decoder. GPT-2 [52] introduces Pre-LayerNorm, relocating layer normalization to the input of self-attention and feedforward sub-blocks, formally:

$$v = \text{Attn}(\text{LayerNorm}(x)) + x,$$
$$y = \text{ReLU}(\text{LayerNorm}(v)W_1)W_2 + v,$$

where Attn denotes the self-attention sub-block. From the proof of Theorem 1, we can easily prove that this theorem still holds for the Pre-LayerNorm structure, using the same parameter transformation. For TEE integration, we need to additionally put the ReLU activation of the feedforward sub-block of the last Transformer layer into the enclave for execution. Formally, function $f(A) = \alpha\text{ReLU}(\alpha^{-1}A)$ is deployed.

RMSNorm. LLaMA series [60] use RMSNorm [71] to replace LayerNorm. To support RMSNorm operator, STIP transforms its weight γ by $\gamma\pi$, then we can prove that

$$\text{RMSNorm}(x\pi;\gamma\pi) = \text{RMSNorm}(x;\gamma)\pi.$$

GeLU. GPT uses GeLU to replace ReLU in feedforward sub-blocks. Analogous to ReLU, GeLU involves element-wise computation without learnable parameters, hence we have $\text{GeLU}(x\pi) = \text{GeLU}(x)\pi$ and theorem 1 holds.

SwiGLU feedforward. LLaMA [60] uses SwiGLU [57] instead of ReLU in feedforward layers. Let $\text{FFN}_{\text{SwiGLU}}$ denote the feedforward layers using SwiGLU, with the definition:

$$\text{FFN}_{\text{SwiGLU}}(x) = (xW_1\,\text{Sigmoid}(xW_1)xW_3)\,W_2,$$
$$W_1, W_3 \in \mathbb{R}^{d\times m}, W_2 \in \mathbb{R}^{m\times d}.$$

We transform parameters as follows:

$$W_1' = \pi^T W_1,\quad W_3' = \pi^T W_3 \pi_{i,3},\quad W_2' = \pi_{i,3}^T W_2 \pi,$$

where $\text{FFN}'_{\text{SwiGLU}}$ denote the transformed feedforward sub-block. And we prove that $\text{FFN}'_{\text{SwiGLU}}(x\pi) = \text{FFN}_{\text{SwiGLU}}(x)\pi$.

Sparse attention. GPT-3 [9] adopts sparse attention patterns in the Transformer layer [12]. Similarly, Longformer [6] was proposed to improve the memory efficiency for long context. These alterations in attention are equivalent to modifying the masking M matrix. According to our proof, Theorem 1 holds for any M matrix, obviating the need for adjusting the parameter transformation.

6.2 Multi-Modal Models

ViT [19] divides an image into non-overlapping patches, and each patch is linearly embedded to create a sequence of token embeddings. These token embeddings serve as the input to the Transformer model. Since STIP does not rely on the pre-processing of the original data, it can seamlessly support ViT. LLaVA [39] takes both text and an image as inputs. It employs a visual transformer to embed the image and subsequently connects them with the embeddings of the text input using a linear projection $x_v W$, where x_v denotes the visual embedding. To integrate LLaVA with STIP, we only need to transform the projection weight W by $\pi_v^T W \pi_t$, where π_v and π_t denote the permutation matrices used for visual and textual transformer features, respectively.

6.3 Mixture-of-Experts Models

Mixtral [32] integrates mixture-of-experts (MoE) into Transformer by constructing multiple feedforward sub-blocks (referred to as experts) in parallel, complemented by a router (or gating layer). The router determines the weights for the experts through $g(x) = xW_g$, where $W_g \in \mathbb{R}^{d\times e}$ and e represents the number of experts. To support MoE, a simple transformation of W_g suffices, accomplished by $\pi^T W_g$.

6.4 Application Scope

For layers with learnable parameters, STIP requires them only to involve global matrix multiplication (e.g., linear, self-attention and feedforward) or token-wise aggregation (e.g., LayerNorm). To give some counterexamples, STIP cannot be extended to convolutional and recurrent layers.

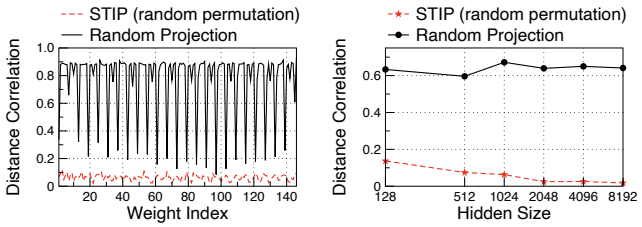
For layers without learnable parameters, STIP requires them to meet $f(x\pi) = f(x)\pi$ property, i.e., column-wise permutation equivariance. For example, ReLU, GeLU, SoftMax, and Sigmoid activation layers.

7 Evaluation

We evaluate STIP on various Transformer inference tasks using both real systems and public datasets. Our key findings are outlined as follows:
• STIP demonstrates practical security concerning model parameters and user data and has no loss of accuracy (§ 7.3).
• STIP 's throughput is comparable to unprotected full-cloud inference, outperforming state-of-the-art secure two-party protocols [11, 17, 26, 28, 40] by millions of times (§ 7.4).
• STIP exhibits efficiency in evaluating various microbenchmarks, including communication overhead, memory footprint, and effect of model split (§ 7.5).
• STIP shows effective resistance to model extraction and finetuning-based attacks by integrating TEE (§ 7.6).

Table 3: Summary of Testbeds and Transformer Models

Testbeds	Modality	Transformers
Campus Security Chatbot (CHAT)	Text	GPT2/LLaMA2
Vehicle Cabin Scene Understanding (CABIN)	Image	ViT/LLaVA
Simulator (SIMU)	Text	BERT/Mixtral



(a) Parameters of GPT2 (b) Embeddings

Figure 7: Privacy leakage measurement: distance correlation.

7.1 Implementation

We implemented STIP using PyTorch and HuggingFace [64] libraries. Modern deep learning frameworks, including PyTorch, adopt a row-major memory layout. To align with the memory layout, PyTorch performs matrix multiplication in the linear layer as xW^T instead of xW. Consequently, we transposed the previously introduced parameter transformation for implementation. For permutation operations, we opted to generate a random permutation vector π_v instead of a matrix π_m. This vector is then used to index rows or columns, as in $W[:,\pi_v]$, which achieves the same effect as $W\pi_m$. The indexing-based approach is more computationally efficient than matrix multiplication. See Appendix B for a code example for transforming GPT2 model parameters using HuggingFace implementation. We will release the code and documentation after this paper is accepted.

7.2 Experimental Setup

Testbeds and Transformer models. We use three testbeds and six representative Transformer models for evaluation, see Tab. 3. (1) Campus Security Chatbot (CHAT). To support natural language Q&A for campus security, we developed a large language model-based chatbot. We select pretrained LLaMA2-7b [60] to host this service at ANONYMOUS University. To scale the evaluation, we also deployed GPT2-124m/355m/774m/1.5b [52] and LLaMA2-13b/70b models[2]. (2) Vehicle Cabin Scene Understanding (CABIN). At ANONYMOUS automotive company, we use LLaVA-13b [39] to implement the cabin scene understanding function. LLaVA model takes in-cabin video frames and a preset prompt as inputs to generate scene descriptions. We also deployed ViT-86m/307m/632m models [19] for comprehensive experiments. (3) Simulator (SIMU). To further evaluate STIP on BERT series [15] and Mixtral [32] models, we build a simulation testbed and test with randomly generated data.

Baselines. For comparisons, we consider four approaches: (1) Full-cloud. Transformer models with original parameters are deployed on the cloud and the device sends raw data (plaintext) to the cloud for inference. (2-4) Iron [26], THE-X [11], and CipherGPT [28]. They propose secure two-party protocols for serving BERT series and GPT-2 models. (5-6) Bumblebee [40] and Puma [17]. They support secure two-party inference for LLaMA series models.

Devices. For all cases, we use a server with 4 NVIDIA A100 GPUs as the model server. In the CHAT testbed, we use a PC with 8-core Intel Core i7 CPUs as the user device. In the CABIN testbed, we use an NVIDIA Orin development board as the user device. And for SIMU, we use a MacBook Pro laptop with 4-core Intel Core i5 CPUs as the user device.

7.3 Security and Accuracy Guarantee

First, we evaluate the previously proven security and computational equivalence through experiments.

Distance correlation. In our privacy analysis, we employ distance correlation [58] as the metric for assessing privacy leakage. As a baseline, we utilize random linear projection on both parameters and embeddings, referred to as Random Projection. In Fig. 7a, we present the distance correlation between the original and transformed parameters of the GPT2-1.5b model. Notably, STIP demonstrates a significantly lower distance correlation compared to Random Projection. On average, Random Projection yields a distance correlation of 0.76, while STIP achieves a markedly lower value of 0.062. To evaluate the security of on-device data, we apply transformations to embeddings with various hidden sizes ranging from 128 to 8192. The resulting distance correlations are depicted in Fig. 7b. In the case of Random Projection, the transformed data maintains a correlation higher than 0.6 on average. Conversely, the distance correlation of STIP diminishes with increasing hidden sizes, ranging from 0.14 to 0.017. This showcases the effectiveness of STIP in reducing privacy leakage associated with transformed data. Our experimental findings affirm the low privacy leakage of permutation-based transformed data and parameters, providing validation for our bound analysis in Sec. 5.3.

On-cloud parameter unavailability. In addition to quantifying the correlation, we also assess the practical unavailability of on-cloud parameters by generating tokens using transformed parameters. We use identical prompts and feed them to the GPT2-1.5b model with original and STIP transformed parameters. Tab. 4 demonstrates the results. With the

[2]Note that the number after the connector - refers to the parameter amount.

Table 4: Unauthorized use of on-cloud transformed model generates meaningless tokens.

Real Generation	On-cloud Generation
Prompt: I'm a language model,	
but what I do in that role is to change everything in our lives.	examines Blazers consolationtechorate applicationkiJanuary PLANkikiorate Blazers consolation Beyondki

prompt "I'm a language model,", tokens generated by on-cloud transformed parameters are completely meaningless, highlighting the practical unavailability of deployed parameters. This observation emphasizes the effectiveness of STIP in securing parameters from unauthorized use.

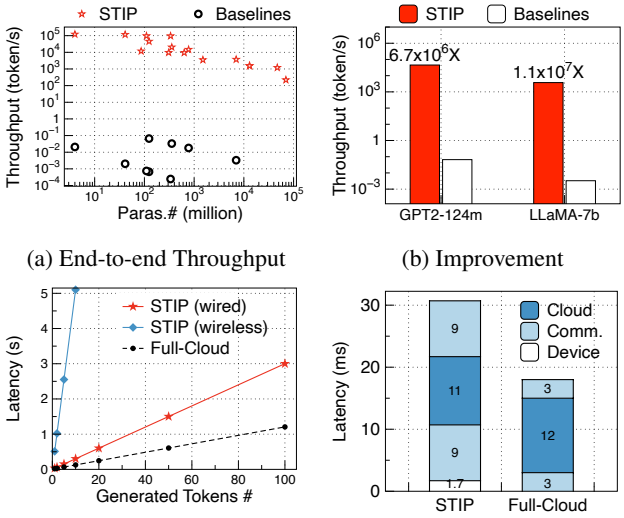
No loss of accuracy. A key advantage of STIP lies in its computational equivalence, ensuring that serving Transformer models with STIP incurs no loss of accuracy. We assess this by examining two metrics: the sum of absolute differences in predictions and top-1 token classification accuracy. We conducted tests on all six selected model series, ranging from 4 million to 70 billion parameters, using 10000 samples each. As depicted in Table 5, STIP consistently achieves 100% accuracy across all models. It's worth noting that the slight non-zero absolute difference is attributable to inherent floating-point operation errors rather than any loss of accuracy introduced by STIP .

7.4 Inference Efficiency

Next, we evaluate the inference efficiency of STIP. The results are tested on the testbed devices associated with the model, and we will not make additional explanations.

End-to-end throughput and scalability with parameter size. We conducted tests to evaluate the end-to-end throughput of serving Transformer models with STIP. The batch size was set to 100, and the number of tokens per sample was set to 100. As illustrated in Fig. 8 (a), STIP demonstrates orders of magnitude higher throughput compared to baselines [11,17,26,28,40]. Additionally, we performed full-cloud inference tests, but the results were close to STIP, causing overlap of markers and, consequently, were omitted for clarity. For GPT2-124m and LLaMA2-7b throughput, Puma reported 6.7e-2 and 3.3e-3 token/s, whereas STIP achieves 45,366 and 3738 token/s, showcasing an improvement of 0.67 and 1.1 million times. Fig. 8 (b) summarizes the throughput improvements. On the other hand, our experiments with STIP have a parameter size of up to 70 billion, which, to the best of our knowledge, is the largest in the literature.

Autoregressive generation. In addition to a single-round feedforward pass, we conducted tests on autoregressive generation with STIP, considering both wired and wireless network connections for STIP communication. The average communication latency for wired connections is approximately 10ms, while for wireless connections, it is around 250ms. With a batch size of 1 and 128 input prompts, Fig. 8 (c) presents the results for the LLaMA2-7b model. The latency for all serving approaches exhibits a linear increase with the number of generated tokens. The slopes for the result lines of Full-cloud, STIP wired, and STIP wireless are approximately 12, 30, and 510, respectively. As discussed in Sec. 5.2, the communication cost per generated token is inevitable to ensure output privacy protection. Considering the practical security that STIP introduces compared to unprotected full-cloud inference, the slightly higher latency (e.g., 2s more for 100 tokens) is deemed acceptable.



(a) End-to-end Throughput (b) Improvement

(c) Autoregressive Generation (d) Latency Breakdown

Figure 8: Efficient serving Transformers with STIP.

Latency breakdown. To gain deeper insights into the overhead introduced by STIP, we conducted an analysis of latency breakdown, comparing it against full-cloud inference. Illustrated in Fig. 8d, our evaluation reveals that STIP introduces an additional 1.7ms latency on the device while concurrently reducing on-cloud latency from 12ms to 11ms. A crucial factor contributing to the slower performance of STIP compared to full-cloud is the communication phase. This arises from the necessity of transmitting intermediate embeddings, a BATCH $\times n \times d$ tensor, which typically exceeds the size of plaintext words transmitted in full-cloud serving. While prior efforts [69] have investigated techniques to compress intermediate activations and enhance communication efficiency in model splitting scenarios, it is noteworthy that our work imposes strict requirements for lossless accuracy, rendering these compression techniques beyond the current design scope. Integrating such compression methods with STIP is a promising direction for future research.

Table 5: STIP has the guarantee of lossless inference accuracy. Numerical differences arise from floating-point arithmetic errors.

Model	GPT-2	LLaMA2	ViT	BERT	LLaVA	Mixtral
Paras.	124m/355m/774m/1.5b	7b/13b/70b	86m/307m/632m	4m/41m/110m/336m	13b	47b
Abs. Diff.	0.021/0.033/0.0478/0.051	0.009/0.012/0.012	3e-4/3e-4/3e-4	5e-3/8e-3/9e-3/9e-3	0.016	8e-3
Class Acc.	100%	100%	100%	100%	100%	100%

7.5 Micro-Benchmarks

Device-cloud communication traffic. The communication traffic induced by STIP is influenced by three factors: the number of input tokens, hidden size, and output vocabulary size. To illustrate, considering the GPT2-124m model, a single-round inference operation causes 5.8 MiB and 7.5 MiB of traffic for input embedding and output activations, respectively. As depicted in Fig. 2, the communication traffic incurred by STIP is markedly lower compared to CipherGPT, 95,151 MiB. This substantial reduction in traffic highlights STIP's ability to achieve security at a modest cost.

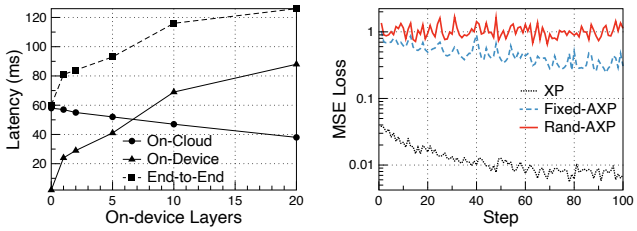
On-device memory footprint. In light of the diverse range of devices that may be employed for Transformer-based services, we assess the on-device memory footprint. For the tokenizer component, LLaMA2 and ViT models exhibit memory footprints of 18 MiB and 3.1 MiB, respectively. In the case of the embedding part, the memory allocation depends on the hidden size parameter. LLaMA2-70b, utilizing a large hidden size of 8192, incurs a memory cost of 903 MiB. In contrast, the ViT models exhibit more modest memory requirements, ranging from 3.9 MiB to 4.9 MiB. This implies that the on-device memory demands of STIP, even for models with substantial hidden sizes, remain feasible for contemporary end devices.

Effect of model split. We vary the number of on-device Transformer layers from 0 to 20 and analyze the corresponding impact on inference latency. As depicted in Fig.9a, the latency of end-to-end inference rises proportionally with an increasing number of on-device layers. This latency increase is attributed to the relatively lower computing power of devices compared to the cloud. As discussed in Sec. 5.3, deploying more layers on the device not only results in higher latency but also exposes additional parameters to the user, thereby introducing privacy risks. In light of these considerations, our analysis indicates that deploying only the embedding module on the device represents the optimal choice. This configuration minimizes latency while mitigating the potential privacy risks of exposing more layers to the user.

7.6 TEE Integration

Now we evaluate our TEE integration designs. Specifically, we test the resistance to model extraction and fine-tuning attacks, and the inference latency.

Model extraction attack. We assume that the attacker uses



(a) GPT2-1.5b (b) GPT2-124m on Wikitext-2

Figure 9: Inference latency with different model splits (a) and resistance to model extraction attacks (b).

authorized inference to obtain the input and output samples of the initial transformation (module I in Fig. 6) executed in the TEE. We consider that the attacker trains a linear model to learn a mapping from the embedding x to three outputs: (1) XP: column-wise permuted embedding; (2) Fixed-AXP: transformation with a fixed diagonal matrix; (3) Rand-AXP: transformation with random diagonal matrices, which is also the approach used by STIP. We use the pre-trained GPT2-124m model's embedding and the WikiText-2 dataset [42] to train linear models. As shown in Fig. 9b, using only permutation for protection is easily compromised by a linear fitting. When we use a fixed linear transformation, the loss also has a slow downward trend. Increasing the sample size or enlarging the capacity of the fitting model may lead to a certain degree of risk. Our design introduces one-time randomness, therefore the loss is in an oscillating state and has no downward trend, effectively preventing model extraction attacks.

Fine-tuning attack. Another possible attack approach is to use the obtained output tokens to fine-tune the transformed model, aiming to restore the original model performance. Specifically, we consider two attacks: (1) Train Transformed: Fine-tune all parameters on the STIP-transformed model; (2) Train Linear: Freeze transformed parameters, then add and train a linear layer after embedding. For comparison, we consider two baselines: (3) Finetune: Fine-tune all parameters on the normal model; (4) Train from Scratch: Train all parameters on a randomly initialized model; We use the GPT2 series models and the WikiText-2 dataset [42] for experiments. As shown in Tab. 6, neither fine-tuning all parameters nor training additional linear layers can effectively restore the model performance (worse than training a model from scratch). We also compare the fine-tuning results of models with different

Table 6: Resistance to finetuning-based attacks using GPT2-124m model. PPL denotes the perplexity metric. The values before and after the / symbol represent the experiment results using the original and the finetuned parameters, respectively.

Approach	Train Loss	Test Loss	Test Acc.	Test PPL
Original	-	3.42	0.38	31
Finetune	3.13	3.05	0.43	21
Train from Scratch	6.59	6.29	0.18	538
Train Transformed	7.25 /7.16	6.93 /6.82	0.14 /0.15	1028 /919
Train Linear	10.23 /10.25	9.36 /9.54	0.11 /0.11	11640 /13963

Table 7: Inference latency with and without TEE in STIP.

Model	10x100 tokens inference latency (s) w/o TEE	w/ TEE (all)	w/ TEE (io)
ViT-86m	0.76	1.15	0.78 (↓2.63%)
GPT2-124m	0.86	1.27	0.93 (↓8.14%)
GPT2-1.5b	0.98	2.26	1.03 (↓5.10%)
LLaMA2-7b	1.81	3.38	1.96 (↓8.29%)
LLaMA2-70b	8.64	31.32	9.45 (↓9.38%)

parameter sizes. Experimental results show that the effect of attacking the 1.5b model is much worse than training the pre-trained 124m model. We can conclude that in practical applications, instead of attacking the model protected by STIP, it is better for attackers to directly train an open-source model.

Inference latency. We use NVIDIA A100 GPUs and Intel Xeon 6330 CPUs to simulate TEE execution. We compare three execution approaches: (1) w/o TEE: All calculations are performed in the GPU. (2) w/ TEE (all): For every Transformer layer, TEE is used. (3) w/ TEE (io): TEE is used only for the first (input) and last (output) Transformer layers, which is also the default design used by STIP. Tab. 7 shows the latency of six Transformer models with parameters ranging from 86m to 70b. Experimental results show that with our carefully designed integration approach, using TEE only reduces the efficiency by 2.6-9.4%. Considering the enhanced security of model parameters brought by integrating TEE, this efficiency loss is completely acceptable.

8 Discussion

Supported two-party use cases. Recall that our threat model only assumes that the model developer and the cloud cannot collude. For the cases where (1) the cloud and the device are one party, and (2) the model developer and the device are one party, STIP can still be applied. The first case is common in all-in-one solutions that include models and hardware. For example, a company deploys its own developed model on its own hardware and delivers the entire solution to users, who then perform inference on their own data. The second use case is more common in academia, for example, a laboratory develops its own model and then deploys the model on a cloud platform to process private data.

Training support extension. STIP is designed to address the forward pass, and model training introduces the complexity of backward gradient propagation. We envision that the principles underlying STIP could be promisingly extended to support privacy-preserving training. However, this extension entails studying the communication overhead associated with gradients and the additional privacy leakage risks introduced during gradient propagation [62], e.g., data poisoning and membership inference attacks [67]. Future ongoing exploration is needed to fully realize the potential of STIP in the context of Transformer training and finetuning.

9 Related Work

Neural network split. The practice of splitting neural network layers and distributing them between the device and server has been explored as a means to protect raw on-device data while preserving efficiency [33,40,47,48,59,70]. Recent research [66] proposes using permutations to enhance protection further. However, despite this split, the potential for reverse-engineering sensitive information from intermediate activations [2], such as text embeddings [46], remains a concern. STIP builds upon the concept of model split and goes a step further by incorporating random permutation, offering theoretically enhanced security measures.

Secure Transformer inference. In the context of a two-party setting, prior efforts [11, 17, 26, 28, 40, 67] have explored the combination of homomorphic encryption and multi-party computation techniques to devise secure protocols for Transformer inference. In addition to HE techniques, function secret sharing [25] can also be used for secure Transformer inference. These approaches customize and optimize computation protocols for specific layers within Transformer models, such as non-linear activation and layer normalization. In contrast to these two-party systems, STIP introduces a novel three-party threat model and employs a semi-symmetrical permutation scheme to enhance security.

10 Conclusion

In this paper, we studied security concerns in Transformer inference. We proposed a three-party threat model and presented the design of STIP, a secure transformer inference protocol based on our semi-symmetrical permutation scheme. Theoretical analysis and experiments in real systems evaluated STIP's practical security, computational equivalence, and computational efficiency.

Ethics Considerations and Compliance with the Open Science Policy

Disclosures. The data and parameter security issue discussed in this paper is a classic, long-standing multi-party computation problem with no disclosure risk.

Innovations with both positive and negative potential outcomes. Our secure inference protocol fairly protects user data privacy and model developers' proprietary parameters without favoring the interests of any one party. Therefore, we believe that integrating STIP will not have direct negative outcomes.

The law. STIP was created to make the existing Transformer inference services more compliant with laws and regulations, such as GDPR. Our research in this paper complies with laws and regulations.

Open science: Availability, functionality, and reproducibility. The data, model structure, and model weights used in this paper are all public. The algorithm details have been clearly described and we will open-source the algorithm code after the paper is accepted. Specifically, we will release:

- Code for performing parameter transformation for GPT2, LLaMA2, ViT, LLaVA, BERT, Mixtral models (Tab. 3);
- Code for measuring distance correlation, see Fig. 7b and Fig, 7a;
- Code for testing inference throughput (Fig. 8) and latency (Fig. 8d);
- Code for testing the absolute difference and accuracy of inference results (Tab. 5);
- Code for performing model extraction attack (Fig. 9b) and fine-tuning attacks (Tab. 6);
- Code for performing TEE simulation inference and latency tests (Tab. 7).

References

[1] Microsoft: We're bringing chatgpt to the azure cloud-computing service. https://www.zdnet.com/article/microsoft-were-bringing-chatgpt-to-the-azure-openai-cloud-computing-service/, 2023.

[2] Sharif Abuadbba, Kyuyeon Kim, Minki Kim, Chandra Thapa, Seyit A Camtepe, Yansong Gao, Hyoungshick Kim, and Surya Nepal. Can we use split learning on 1d cnn models for privacy preserving training? In *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*, pages 305–318, 2020.

[3] Samuel K Ainsworth, Jonathan Hayase, and Siddhartha Srinivasa. Git re-basin: Merging models modulo permutation symmetries. *arXiv preprint arXiv:2209.04836*, 2022.

[4] Reza Yazdani Aminabadi, Samyam Rajbhandari, Ammar Ahmad Awan, Cheng Li, Du Li, Elton Zheng, Olatunji Ruwase, Shaden Smith, Minjia Zhang, Jeff Rasley, et al. Deepspeed-inference: enabling efficient inference of transformer models at unprecedented scale. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15. IEEE, 2022.

[5] Azure. Chatgpt is now available in azure openai service. https://azure.microsoft.com/en-us/blog/chatgpt-is-now-available-in-azure-openai-service/, 2023.

[6] Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

[7] Umang Bhargava, Aparna Sharma, Raghav Chawla, and Prateek Thakral. A new algorithm combining substitution & transposition cipher techniques for secure communication. In *2017 International Conference on Trends in Electronics and Informatics (ICEI)*, pages 619–624. IEEE, 2017.

[8] Tiziano Bianchi, Valerio Bioglio, and Enrico Magli. Analysis of one-time random projections for privacy preserving compressed sensing. *IEEE Transactions on Information Forensics and Security*, 11(2):313–327, 2015.

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

[10] Samuel Carreira, Tomás Marques, José Ribeiro, and Carlos Grilo. Revolutionizing mobile interaction: Enabling a 3 billion parameter gpt llm on mobile. *arXiv preprint arXiv:2310.01434*, 2023.

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

[12] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

[13] Google DeepMind. Gemini. https://deepmind.google/technologies/gemini/, 2023.

[14] Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Llm. int8 (): 8-bit matrix multiplication for transformers at scale. *arXiv preprint arXiv:2208.07339*, 2022.

[15] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[16] Xiaoqiang Di, Jinqing Li, Hui Qi, Ligang Cong, and Huamin Yang. A semi-symmetric image encryption scheme based on the function projective synchronization of two hyperchaotic systems. *PloS one*, 12(9):e0184586, 2017.

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

[18] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[19] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *ICLR*, 2021.

[20] Ozlem Durmaz Incel and Sevda Özge Bursa. On-device deep learning for mobile and wearable sensing applications: A review. *IEEE Sensors Journal*, 23(6):5501–5512, 2023.

[21] N Fares and Shavan Askar. A novel semi-symmetric encryption algorithm for internet applications. *Journal of University of Duhok*, 19(1):1–9, 2016.

[22] Forbes. Samsung bans chatgpt among employees after sensitive code leak. https://www.forbes.com/sites/siladityaray/2023/05/02/samsung-bans-chatgpt-and-other-chatbots-for-employees-after-sensitive-code-leak/, 2023.

[23] Karan Ganju, Qi Wang, Wei Yang, Carl A Gunter, and Nikita Borisov. Property inference attacks on fully connected neural networks using permutation invariant representations. In *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, pages 619–633, 2018.

[24] Google. Bard. https://bard.google.com/chat, 2023.

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

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

[27] Qijian He, Wei Yang, Bingren Chen, Yangyang Geng, and Liusheng Huang. Transnet: Training privacy-preserving neural network over transformed layer. *Proceedings of the VLDB Endowment*, 13(12):1849–1862, 2020.

[28] Xiaoyang Hou, Jian Liu, Jingyu Li, Yuhan Li, Wen-jie Lu, Cheng Hong, and Kui Ren. Ciphergpt: Secure two-party gpt inference. *Cryptology ePrint Archive*, 2023.

[29] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *CoRR*, abs/2106.09685, 2021.

[30] HuggingFace. Open-llm-leaderboard, 2023.

[31] Patrick Jauernig, Ahmad-Reza Sadeghi, and Emmanuel Stapf. Trusted execution environments: properties, applications, and challenges. *IEEE Security & Privacy*, 18(2):56–60, 2020.

[32] Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mixtral of experts, 2024.

[33] Penghao Jiang, Ke Xin, Chunxi Li, and Yinsi Zhou. High-efficiency device-cloud collaborative transformer model. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2203–2209, 2023.

[34] Wenbo Jiang, Hongwei Li, Guowen Xu, Tianwei Zhang, and Rongxing Lu. A comprehensive defense framework against model extraction attacks. *IEEE Transactions on Dependable and Secure Computing*, 21(2):685–700, 2023.

[35] Oleksii Kuchaiev, Jason Li, Huyen Nguyen, Oleksii Hrinchuk, Ryan Leary, Boris Ginsburg, Samuel Kriman, Stanislav Beliaev, Vitaly Lavrukhin, Jack Cook, et al.

Nemo: a toolkit for building ai applications using neural modules. *arXiv:1909.09577*, 2019.

[36] Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *International conference on machine learning*, pages 3744–3753. PMLR, 2019.

[37] Taesung Lee, Benjamin Edwards, Ian Molloy, and Dong Su. Defending against neural network model stealing attacks using deceptive perturbations. In *2019 IEEE Security and Privacy Workshops (SPW)*, pages 43–49. IEEE, 2019.

[38] Qinfeng Li, Zhiqiang Shen, Zhenghan Qin, Yangfan Xie, Xuhong Zhang, Tianyu Du, and Jianwei Yin. Translinkguard: Safeguarding transformer models against model stealing in edge deployment. *arXiv preprint arXiv:2404.11121*, 2024.

[39] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning, 2023.

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

[41] Takahiro Maekawa, Ayana Kawamura, Yuma Kinoshita, and Hitoshi Kiya. Privacy-preserving svm computing in the encrypted domain. In *2018 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, pages 897–902. IEEE, 2018.

[42] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models, 2016.

[43] NVIDIA. Confidential computing. https://www.nvidia.com/en-us/data-center/solutions/confidential-computing/, 2024.

[44] Stanley RM Oliveira and Osmar R Zaiane. Privacy-preserving clustering by object similarity-based representation and dimensionality reduction transformation. In *Proceedings of the 2004 ICDM Workshop on Privacy and Security Aspects of Data Mining*, pages 40–46, 2004.

[45] OpenAI. tiktoken. https://github.com/openai/tiktoken, 2024.

[46] Xudong Pan, Mi Zhang, Shouling Ji, and Min Yang. Privacy risks of general-purpose language models. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 1314–1331. IEEE, 2020.

[47] Dario Pasquini, Giuseppe Ateniese, and Massimo Bernaschi. Unleashing the tiger: Inference attacks on split learning. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 2113–2129, 2021.

[48] Ngoc Duy Pham, Alsharif Abuadbba, Yansong Gao, Tran Khoa Phan, and Naveen Chilamkurti. Binarizing split learning for data privacy enhancement and computation reduction. *IEEE Transactions on Information Forensics and Security*, 2023.

[49] Manoj Prabhakaran and Mike Rosulek. Cryptographic complexity of multi-party computation problems: Classifications and separations. In David Wagner, editor, *Advances in Cryptology – CRYPTO 2008*, pages 262–279, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[50] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

[51] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

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

[53] REUTERS. Chatgpt sets record for fastest-growing user base - analyst note. https://www.reuters.com/technology/chatgpt-sets-record-fastest-growing-user-base-analyst-note-2023-02-01/, 2023.

[54] Francisco Romero, Qian Li, Neeraja J Yadwadkar, and Christos Kozyrakis. Infaas: Automated model-less inference serving. In *2021 USENIX Annual Technical Conference*, pages 397–411, 2021.

[55] Jingqing Ruan, Yihong Chen, Bin Zhang, Zhiwei Xu, Tianpeng Bao, Guoqing Du, Shiwei Shi, Hangyu Mao, Ziyue Li, Xingyu Zeng, and Rui Zhao. Tptu: Large language model-based ai agents for task planning and tool usage, 2023.

[56] Alexander M Rush. The annotated transformer. In *Proceedings of workshop for NLP open source software (NLP-OSS)*, pages 52–60, 2018.

[57] Noam Shazeer. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

[58] Gábor J Székely, Maria L Rizzo, and Nail K Bakirov. Measuring and testing dependence by correlation of distances. 2007.

[59] Chandra Thapa, Pathum Chamikara Mahawaga Arachchige, Seyit Camtepe, and Lichao Sun. Splitfed: When federated learning meets split learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 8485–8493, 2022.

[60] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

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

[62] Junxiao Wang, Song Guo, Xin Xie, and Heng Qi. Protect privacy from gradient leakage attack in federated learning. In *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*, pages 580–589. IEEE, 2022.

[63] Yining Wang, Yu-Xiang Wang, and Aarti Singh. A theoretical analysis of noisy sparse subspace clustering on dimensionality-reduced data. *IEEE Transactions on Information Theory*, 65(2):685–706, 2018.

[64] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Huggingface's transformers: State-of-the-art natural language processing, 2020.

[65] Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning*, pages 38087–38099. PMLR, 2023.

[66] Hengyuan Xu, Liyao Xiang, Hangyu Ye, Dixi Yao, Pengzhi Chu, and Baochun Li. Permutation equivariance of transformers and its applications. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5987–5996, 2024.

[67] Biwei Yan, Kun Li, Minghui Xu, Yueyan Dong, Yue Zhang, Zhaochun Ren, and Xiuzheng Cheng. On protecting the data privacy of large language models (llms): A survey. *arXiv preprint arXiv:2403.05156*, 2024.

[68] Fan Yang, Zhiyuan Chen, and Aryya Gangopadhyay. Using randomness to improve robustness of tree-based models against evasion attacks. In *Proceedings of the ACM International Workshop on Security and Privacy Analytics*, pages 25–35, 2019.

[69] Shuochao Yao, Jinyang Li, Dongxin Liu, Tianshi Wang, Shengzhong Liu, Huajie Shao, and Tarek Abdelzaher. Deep compressive offloading: Speeding up neural network inference by trading edge computation for network latency. In *Proceedings of the 18th conference on embedded networked sensor systems*, pages 476–488, 2020.

[70] Liekang Zeng, Xu Chen, Zhi Zhou, Lei Yang, and Junshan Zhang. Coedge: Cooperative dnn inference with adaptive workload partitioning over heterogeneous edge devices. *IEEE/ACM Transactions on Networking*, 29(2):595–608, 2020.

[71] Biao Zhang and Rico Sennrich. Root mean square layer normalization. *Advances in Neural Information Processing Systems*, 32, 2019.

[72] Chengliang Zhang, Junzhe Xia, Baichen Yang, Huancheng Puyang, Wei Wang, Ruichuan Chen, Istemi Ekin Akkus, Paarijaat Aditya, and Feng Yan. Citadel: Protecting data privacy and model confidentiality for collaborative learning. In *Proceedings of the ACM Symposium on Cloud Computing*, SoCC '21, page 546–561, New York, NY, USA, 2021. Association for Computing Machinery.

[73] Tieyu Zhao, Qiwen Ran, Lin Yuan, Yingying Chi, and Jing Ma. Information verification cryptosystem using one-time keys based on double random phase encoding and public-key cryptography. *Optics and Lasers in Engineering*, 83:48–58, 2016.

[74] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 2023.

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

[76] Shuai Zhou, Tianqing Zhu, Dayong Ye, Wanlei Zhou, and Wei Zhao. Inversion-guided defense: Detecting model stealing attacks by output inverting. *IEEE Transactions on Information Forensics and Security*, 2024.

Appendices are supporting material that has not been peer-reviewed.

A Proofs

Proof of Theorem 1:

$$F_{\theta'}(x\pi)\pi_c^T = F_\theta(x).$$

Proof. First, since the calculation of non-linear activation is element-wise, they are permutation equivalent, i.e., ReLU($x\pi$)=ReLU(x)π and SoftMax($x\pi$)=SoftMax(x)π.
Next, we prove that:
LayerNorm($x\pi;\gamma\pi,\beta\pi$) =LayerNorm($x;\gamma,\beta$)π.
The LayerNorm function is defined for $x \in \mathbb{R}^{n\times d}$ by

$$\text{LayerNorm}(x;\gamma,\beta) = \gamma \circ \frac{x-\mu_x}{\sigma_x} + \beta, \qquad \gamma,\beta \in \mathbb{R}^d,$$

where $\circ$ denotes the Hadamard (element-wise) product operator. Since μ_x and σ_x are computed by rows, $\mu_{x\pi} = \mu_x$ and $\sigma_{x\pi} = \sigma_x$. Therefore,

$$\begin{aligned}\text{LayerNorm}(x\pi;\gamma\pi,\beta\pi) &= \gamma\pi \circ \frac{x\pi-\mu_x}{\sigma_x} + \beta\pi \\ &= \left(\gamma \circ \frac{x-\mu_x}{\sigma_x} + \beta\right)\pi \\ &= \text{LayerNorm}(x;\gamma,\beta)\pi.\end{aligned}$$

Then, since $\forall\pi, \pi\pi^T = I$:

$$\begin{aligned}
Q' &= x\pi\pi^T W_q\pi_{i,1} = xW_q\pi_{i,1} = Q\pi_{i,1},\\
K' &= x\pi\pi^T W_k\pi_{i,1} = xW_k\pi_{i,1} = K\pi_{i,1},\\
V' &= x\pi\pi^T W_v\pi_{i,2} = xW_v\pi_{i,2} = V\pi_{i,2},\\
u' &= \text{SoftMax}\left(\frac{Q'K'^T}{\sqrt{k}} + M\right)V'\pi_{i,2}^T W_o\pi\\
&= \text{SoftMax}\left(\frac{Q\pi_{i,1}\pi_{i,1}^T K^T}{\sqrt{k}} + M\right)V\pi_{i,2}\pi_{i,2}^T W_o\pi\\
&= \text{SoftMax}\left(\frac{QK^T}{\sqrt{k}} + M\right)VW_o\pi = u\pi,\\
v' &= \text{LayerNorm}(u' + x\pi;\gamma_1',\beta_1')\\
&= \text{LayerNorm}(u\pi + x\pi;\gamma_1\pi,\beta_1\pi)\\
&= \text{LayerNorm}((u+x)\pi;\gamma_1\pi,\beta_1\pi) = v\pi,\\
z' &= \text{ReLU}(v'W_1')W_2' = \text{ReLU}(v\pi\pi^T W_1\pi_{i,3})\pi_{i,3}^T W_2\pi\\
&= \text{ReLU}(vW_1)W_2\pi = z\pi,\\
y' &= \text{LayerNorm}(z' + v';\gamma_2',\beta_2')\\
&= \text{LayerNorm}(z\pi + v\pi;\gamma_2\pi,\beta_2\pi)\\
&= \text{LayerNorm}((z+v)\pi;\gamma_2\pi,\beta_2\pi) = y\pi,\\
o' &= y'W_c' = y\pi\pi^T W_c\pi_c = o\pi_c.
\end{aligned}$$

Therefore, $F'_\theta(x\pi)\pi_c^T = o'\pi_c^T = o\pi_c\pi_c^T = o = F_\theta(x)$. □

Proof of Theorem 2:

$$\alpha^{-1}F_{\theta',TEE}(\alpha x\pi)\pi_c^T = F_\theta(x).$$

Proof. First, we prove that LayerNorm eliminates the linear transformation caused by α:

$$\text{LayerNorm}(\alpha x) = \text{LayerNorm}(x).$$

Since α performs a linear transformation on each row, the mean and standard deviation calculated for each row also retain the linear transformation, that is, $\mu_{\alpha x} = \alpha\mu_x$ and $\sigma_{\alpha x} = \alpha\sigma_x$. Therefore:

$$\begin{aligned}\text{LayerNorm}(\alpha x;\gamma,\beta) &= \gamma \circ \frac{\alpha x - \alpha\mu_x}{\alpha\sigma_x} + \beta\\ &= \left(\gamma \circ \frac{x-\mu_x}{\sigma_x} + \beta\right)\\ &= \text{LayerNorm}(x;\gamma,\beta).\end{aligned}$$

Then, since $\alpha\alpha^{-1} = I, \pi\pi^T = I$, by reusing the notations used in the proof of Theorem 1, the inference process of a Transformer model with TEE integration is as follows:

$$\begin{aligned}
Q'' &= \alpha x W_q\pi_{i,1} = \alpha Q',\\
K'' &= \alpha x W_k\pi_{i,1} = \alpha K',\\
V'' &= \alpha x W_v\pi_{i,2} = \alpha V',\\
u'' &= \alpha\text{SoftMax}\left(\frac{\alpha^{-1}Q''K''^T(\alpha^T)^{-1}}{\sqrt{k}} + M\right)\alpha^{-1}V''\pi_{i,2}^T W_o\pi\\
&= \alpha\text{SoftMax}\left(\frac{\alpha^{-1}\alpha Q'K'^T\alpha^T(\alpha^T)^{-1}}{\sqrt{k}} + M\right)\alpha^{-1}\alpha VW_o\pi\\
&= \alpha\text{SoftMax}\left(\frac{QK^T}{\sqrt{k}} + M\right)VW_o\pi\\
&= \alpha u\pi,\\
v'' &= \text{LayerNorm}(u'' + \alpha x\pi;\gamma_1',\beta_1')\\
&= \text{LayerNorm}(\alpha u\pi + \alpha x\pi;\gamma_1\pi,\beta_1\pi)\\
&= \text{LayerNorm}((u+x)\pi;\gamma_1\pi,\beta_1\pi)\\
&= v\pi,\\
z'' &= \text{ReLU}(v''W_1')W_2' = z\pi,\\
y'' &= \text{LayerNorm}(z'' + v'';\gamma_2',\beta_2')\\
&= \text{LayerNorm}(z\pi + v\pi;\alpha\gamma_2\pi,\alpha\beta_2\pi)\\
&= \alpha\text{LayerNorm}((z+v)\pi;\gamma_2\pi,\beta_2\pi)\\
&= \alpha y\pi,\\
o'' &= y''W_c' = \alpha y\pi\pi^T W_c\pi_c = \alpha o\pi_c.
\end{aligned}$$

Therefore,

$$\alpha^{-1}F_{\theta',TEE}(\alpha x\pi)\pi_c^T = \alpha^{-1}\alpha o\pi_c\pi_c^T = o = F_\theta(x).$$

That is, after integrating TEE, the original inference results can still be restored equivalently. □

Proof of Pre-LayerNorm.

Proof. From the proof of Theorem.1, we can see the permutation equivalence property holds for the self-attention sub-block, i.e., $\text{Attn}(x\pi) = \text{Attn}(x)\pi$. So

$$\begin{aligned}
v' &= \text{Attn}(\text{LayerNorm}'(x\pi)) + x\pi \\
&= \text{Attn}(\text{LayerNorm}(x)\pi) + x\pi \\
&= \text{Attn}(\text{LayerNorm}(x))\pi + x\pi \\
&= (\text{Attn}(\text{LayerNorm}(x)) + x)\pi = v\pi, \\
y' &= \text{ReLU}(\text{LayerNorm}'(v')W_1')W_2' + v' \\
&= \text{ReLU}(\text{LayerNorm}'(v\pi)\pi^T W_1 \pi_{i,3})\pi_{i,3}^T W_2\pi + v\pi \\
&= (\text{ReLU}(\text{LayerNorm}(v)W_1)W_2 + v)\pi = y\pi,
\end{aligned}$$

where $\text{LayerNorm}'$ denotes layer normalization with transformed parameters.

Therefore, $F_\theta'(x\pi)\pi_c^T = F_\theta(x)$ still holds.

For TEE integration, the feedforward sub-block takes $\alpha v'$ as the input. Therefore:

$$\begin{aligned}
y'' &= \alpha\text{ReLU}(\alpha^{-1}\alpha\text{LayerNorm}'(\alpha v')W_1')W_2' + \alpha v' \\
&= \alpha(\text{ReLU}(\text{LayerNorm}(v)W_1)W_2 + v)\pi = \alpha y\pi,
\end{aligned}$$

Therefore, theorem 2 still holds. □

Proof of RMSNorm.

Proof. The RMSNorm function is defined for $x \in \mathbb{R}^{n\times d}$ by

$$\text{RMSNorm}(x;\gamma) = \gamma \circ \frac{x}{\sqrt{\frac{1}{n}\sum_i x_i^2}}, \qquad \gamma \in \mathbb{R}^d,$$

where $\circ$ denotes the Hadamard (element-wise) product operator. Since $\sum_i x_i^2$ is computed by rows, $\sum_i (x\pi)_i^2 = \sum_i x_i^2$. Therefore,

$$\begin{aligned}
\text{RMSNorm}(x\pi;\gamma\pi) &= \gamma\pi \circ \frac{x\pi}{\sqrt{\frac{1}{n}\sum_i (x\pi)_i^2}} \\
&= \left(\gamma \circ \frac{x}{\sqrt{\frac{1}{n}\sum_i x_i^2}}\right)\pi \\
&= \text{RMSNorm}(x;\gamma)\pi.
\end{aligned}$$

□

Proof of SwiGLU feedforward.

Proof. By definition,

$$\begin{aligned}
&\text{FFN}'_{\text{SwiGLU}}(x\pi) = (x\pi W_1' \text{Sigmoid}(x\pi W_1')x\pi W_3)W_2' \\
&= (x\pi\pi^T W_1 \text{Sigmoid}(x\pi\pi^T W_1)x\pi\pi^T W_3 \pi_{i,3})\pi_{i,3}^T W_2\pi \\
&= (xW_1 \text{Sigmoid}(xW_1)xW_3)W_2\pi \\
&= \text{FFN}_{\text{SwiGLU}}(x)\pi.
\end{aligned}$$

□

B Parameter Transformation Code Example

Below we give a parameter transformation code for the HuggingFace implementation of the GPT2 model using PyTorch.

```
import torch
import numpy as np
def permute_gpt2(model, p, p_out):
    for name, para in model.transformer.
    named_parameters():
        t = name.split(".")
        if t[0] in ["wte", "wpe"]:
            continue
        if t[0].startswith("ln"):
            para.data = para.data[p]
            continue
        if t[2].startswith("ln"):
            para.data = para.data[p]
        if t[3]=="c_attn" and t[-1]=="weight":
            para.data = para.data[p]
        if t[2]=="attn" and t[3]=="c_proj":
            if t[-1]=="weight":
                para.data = para.data[:, p]
            if t[-1]=="bias":
                para.data = para.data[p]
        if t[3]=="c_fc" and t[-1]=="weight":
                para.data = para.data[p]
        if t[2]=="mlp" and t[3]=="c_proj":
            if t[-1]=="weight":
                para.data = para.data[:, p]
            if t[-1]=="bias":
                para.data = para.data[p]
    for name, para in model.lm_head.
    named_parameters():
        if name=="weight":
            para.data = para.data[p_out][:, p]
    return model
if __name__=="__main__":
    from transformers import GPT2LMHeadModel
    DMODEL = 768
    DOUT = 50257
    p = np.random.permutation(DMODEL)
    p_out = np.random.permutation(DOUT)
    model = GPT2LMHeadModel.from_pretrained("
    gpt2/")
    x = torch.from_numpy(np.random.rand(1, 1,
    DMODEL)).float()
    x_new = x[:, :, p]
    with torch.no_grad():
        y = model(inputs_embeds=x)
        m_new = permute_gpt2(model, p, p_out)
        y_new = m_new(inputs_embeds=x_new)
        y_pout = y[:, :, p_out]
        abs_diff = np.abs(y_new - y_pout).sum()
        print("abs_diff=", abs_diff)
```

Our implementation does not need to perform matrix multiplication but only needs to perform re-index operations to achieve the permutation operations, so it is very efficient. Note that for code simplicity, we have omitted the $\{\pi_{i,1}, \pi_{i,2}, \pi_{i,3} \mid i \in [L]\}$ transformations and only kept the transformations of π and π_c.