STIP: Three-Party Privacy-Preserving and Lossless Inference for Large Transformers in Production

Mu Yuan*, Lan Zhang†‡, Yihang Cheng†, Miao-Hui Song†, Guoliang Xing*, Xiang-Yang Li†
*The Chinese University of Hong Kong
†University of Science and Technology of China
‡Institute of Artificial Intelligence, Hefei Comprehensive National Science Center
Email: muyuan@cuhk.edu.hk, zhanglan@ustc.edu.cn, yihangcheng@mail.ustc.edu.cn,
songmiaohui@mail.ustc.edu.cn, xiangyangli@ustc.edu.cn, glxing@ie.cuhk.edu.hk

***Abstract*—The privacy of model parameters and user data is crucial for Transformer-based cloud services, such as online chatbots. While recent advances in secure multi-party computation and homomorphic encryption provide strong cryptographic guarantees, their computational overhead makes them infeasible for real-time inference with large-scale Transformer models. In this work, we propose a practical alternative that balances privacy and efficiency in real-world deployments. We introduce a three-party threat model involving a model developer, a cloud model server, and a data owner, capturing the trust assumptions and deployment conditions of practical AI services. Within this framework, we design a semi-symmetric permutation-based protection mechanism and present STIP, the first three-party privacy-preserving inference system for large Transformers deployable on commodity hardware. STIP formally bounds privacy leakage while preserving lossless inference accuracy. To further safeguard model parameters, STIP integrates trusted execution environments to resist model extraction and fine-tuning attacks. We evaluate STIP on six representative Transformer model families, including models with up to 70 billion parameters, under three deployment settings. STIP's efficiency is comparable to unprotected full-cloud inference, for example, STIP achieves 31.7 ms latency on LLaMA2-7B model. STIP also shows effective resistance to various attacks against user data and model parameters. STIP has been deployed in a production environment on our proprietary 70B model. In a three-month online test, STIP brings only 12% additional latency and no privacy incidents were reported, demonstrating its practicality and robustness for production-scale AI systems.**

I. Introduction

Transformer [1] is a neural network architecture that captures relationships between sequential input elements (i.e., tokens) using the self-attention mechanism. This mechanism is based entirely on global matrix multiplication that allows parallel computation [2], making the Transformer architecture computationally scalable. Scaling up Transformer model parameters to billions brings emergent abilities [3], such as performing arithmetic tasks and zero-shot cross-domain question answering. Large Transformer models, e.g., OpenAI's GPT [4], Google's BERT [5], and Meta's LLaMA [6], have become the cornerstone of modern artificial intelligence, enabling next-generation applications such as long-context chatbot [7], [8], software copilot [9], and video generation [10].

Deploying these Transformer models for inference is a typical instance of Machine Learning as a Service (MLaaS) [11]. Take ChatGPT as an example, a *model owner* (OpenAI) hosts its proprietary model (GPT-4o), and a *data owner* (user) sends input text to the model and receives outputs. Large Transformer models are increasingly being deployed in domains including healthcare, finance, and personalized assistants, where privacy is critical. However, serious privacy risks exist in the two typical deployment modes of MLaaS:

- **Public-cloud** deployment, where model parameters are hosted on the model owner's cloud servers. The user's private inputs and outputs are fully exposed to the model owner. This mode may be completely unacceptable for privacy-sensitive areas, e.g., Samsung banned all online chatbots after a sensitive code leak [12].
- **On-device** deployment, where model parameters are hosted on the user's device. The model owner's proprietary parameters are directly exposed in the device's plaintext storage and memory. Potential parameter leakage risks (such as model extraction attacks [13], [14], [15]) can cause billions of losses to model owners.

Given the privacy issues of the above two deployments, a trade-off solution of device-cloud collaborative inference is proposed:

- **Model splitting** deployment [16], [17], [18] strategically distributes neural network layers between the device and the cloud. The device sends intermediate activations to the cloud to continue inference. Model splitting inference only deploys a small number of parameters on the device and avoids uploading the raw input data to the cloud [19], [20], which protects the privacy of both parties to a certain extent.

However, privacy concerns still arise as subsequent research reveals the potential for reverse-engineering sensitive information from intermediate activations [21], [22]. This results in this deployment being rarely adopted in practice.

Lan Zhang is the corresponding author.

Network and Distributed System Security (NDSS) Symposium 2026
23-27 February 2026, San Diego, CA, USA
ISBN 979-8-9919276-8-0
https://dx.doi.org/10.14722/ndss.2026.230035
www.ndss-symposium.org

A line of research has turned to cryptographic approaches for strong theoretical guarantees, especially for the Transformer architecture:

- **HE+2PC** protocols. By combining homomorphic encryption (HE) and secure two-party computation (2PC) techniques, many protocols [23], [24], [25], [26], [27], [28], [29], [30] have been proposed for Transformer models, which provide theoretical security for both private user data and model parameters.

However, when deploying generative models with billions of parameters, there are still two key technical challenges:

Challenge-1: Balancing privacy with exact accuracy. In the highly competitive market of large Transformer models, even a slight degradation in accuracy can result in lagging behind competing products [31]. Many privacy-preserving inference methods adopt approximations (e.g., polynomial activation functions, truncated precision) that introduce minor yet non-negligible accuracy drops. While such approximations may be acceptable in non-critical applications, they pose challenges in domains such as healthcare, finance, or regulated services, where output consistency and auditability are paramount. This raises the question: Can we achieve strong privacy guarantees without compromising model accuracy? Designing such a solution is challenging due to the inherently non-linear and coupled operations within Transformer blocks, which resist obfuscation through reversible transformations.

Challenge-2: high sensitivity to communication costs. Communication costs are increasingly recognized as a significant bottleneck in machine learning systems [34], particularly in multi-party settings. Existing HE+2PC protocols [23], [24], [25], [26], [27], [28], [29] incur prohibitive device-cloud communication overheads. For example, CipherGPT [27] costs a 25-minute processing time and 90 GiB traffic for generating a single token with GPT2 while Puma [25] takes around 5 minutes and 2 GiB communication for LLaMA2-7b. This communication cost and latency make these cryptographic protocols commercially infeasible given the bandwidth limitation of many user devices and basic quality of experience requirements.

While ongoing efforts aim to optimize these protocols (e.g., Rhombus (CCS'24) [35], NEXUS (NDSS'25) [30]), an alternative line of work explores a more deployment-friendly threat model involving **three parties**: model developer, model server, and data owner. This setting decomposes the original model owner into two distinct parties, model developer and model server. Apple's Private Cloud Compute (PCC) [33] exemplifies this approach, which has been used for privacy-preserving inference in production.

- **Private-cloud** deployment, where the model developer (e.g., Apple) hosts its proprietary parameters on the model server (e.g., PCC), and the data owner sends inputs to the model server and receives outputs.

PCC is claimed to guarantee that user data sent to PCC isn't accessible to anyone other than the user, not even to Apple [33]. However, PCC is tied to custom Apple silicon and a hardened operating system. Private-cloud deployment is promising in terms of both data privacy and inference efficiency, but it cannot support the massive commodity hardware in existing public clouds.

Goal and uniqueness. In this work, we aim to bring PCC-like privacy guarantees to commodity cloud infrastructure, without relying on heavy cryptographic primitives. We adopt the three-party setting and present STIP, a privacy-preserving Transformer inference system that: achieves lossless accuracy, imposes a theoretical bound on privacy leakage, and delivers real-time efficiency on large models. Tab. I summarizes Transformer inference methods in both two-party and three-party settings, highlighting the uniqueness of STIP.

STIP design. Since the inference computation is executed on untrusted servers, both parameters and user data must be transformed before uploading to the cloud to preserve privacy. To achieve practical communication efficiency, we avoid the use of heavy cryptographic primitives such as homomorphic encryption (HE) and secure multiparty computation (MPC). Instead, we propose a transformation scheme tailored to the Transformer architecture, based on random permutation and linear scaling. Specifically, for each weight matrix in the Transformer, we generate a pair of random permutation matrices to permute its rows and columns, respectively. These matrices are generated by the model developer. Only one of each pair (used in the first and last layers) needs to be shared with the data owner to enable input/output transformation, while the intermediate-layer matrices are kept private to the model developer. This forms what we term a semi-symmetric scheme, inspired by the sequential nature of neural networks, where only the endpoints require shared transformation knowledge. One advantage of STIP is that it preserves the full computational semantics of the Transformer model. Unlike cryptographic approaches based on HE or MPC that rely on polynomial approximations of non-linear functions (e.g., ReLU, SoftMax, LayerNorm) [23], [32], our transformation avoids modifying the model's computational graph. We theoretically prove the computation equivalence of the transformed network, ensuring that STIP produces outputs that are identical to those of the original plaintext model.

We demonstrate the privacy-preserving capability of STIP by deriving a strict upper bound on information leakage using distance correlation [36], and by analyzing its resistance to brute-force and known-plaintext attacks. To strengthen protection of model parameters during execution, STIP integrates a lightweight Trusted Execution Environment (TEE)-enhanced execution path, confining only four simple operations to the enclave. This minimal TEE reliance ensures both practical efficiency and robustness against model extraction and fine-tuning attacks, with just a 9.38% latency increase even on a 70B model. Overall, STIP achieves a unique Pareto-optimal point in the privacy–efficiency–accuracy trade-off space: strong privacy protection, real-time inference efficiency, and zero degradation in model outputs.

Contributions of this work are summarized as follows:

• We developed STIP, the first three-party inference system for

TABLE I: Comparison of our proposed STIP and existing Transformer inference methods. In the "Msg. Complexity" column, T denotes the number of generated tokens and L denotes the number of Transformer layers.

Method	Privacy-Preserving	Lossless	Commodity Hardware	Msg. Complexity	Tested Models
Two-Party: Model Owner, Data Owner					
Public-Cloud, On-Device	✗	✓	✓	O(1)	All
Model Splitting [16]	✗	✓	✓	O(T)	All
Iron [28], THE-X [29], BOLT [23]	✓	✗	✓	O(TL)	BERT
CipherGPT [27]	✓	✗	✓	O(TL)	GPT-2
Bumblebee [26], Puma [25], SIGMA [32], NEXUS [30]	✓	✗	✓	O(TL)	GPT/LLaMA/BERT
Three-Party: Model Developer, Model Server, Data Owner					
Private-Cloud [33]	✓	✓	✗	O(1)	All
STIP	✓	✓	✓	O(T)	GPT/LLaMA/ ViT/LLaVA/ BERT/Mixtral

large Transformer models using commodity hardware, with the theoretical bound of privacy leakage, guarantee of no loss of accuracy, and millisecond-level inference latency.

• We implemented STIP and conducted evaluations on six representative series of Transformer models. Experiments show the privacy protection of STIP against various attacks, including model extraction and fine-tuning attacks. In terms of efficiency, STIP achieves throughput and latency comparable to the unprotected and private-cloud [33] deployments. Just for reference (due to different settings), STIP achieves efficiency levels orders of magnitude higher than cryptography-based solutions [27], [28], [29], [25], [26], [23], [30].

• STIP has been deployed in a nationwide production environment on our proprietary 70B model, running on a major cloud platform, bringing only 12% additional latency and no major privacy incidents were reported in three-month traces. This deployment demonstrates the real-world viability of STIP.

II. Background

A. Transformer Architecture

We use the original Transformer architecture [1] to introduce the inference computation, without loss of generality, advanced Transformer variants (GPT[4], LLaMA [6], ViT [37] and Mixtral [38]) will be discussed in Sec. V.

Input embedding. In Transformer models, the embedding operation is the initial step that maps discrete inputs (such as words or images) into continuous vectors [1]. This operation typically consists of tokenization, linear transformation, and positional encoding [1]. For text inputs, words are first tokenized into token IDs which are represented as one-hot vectors (where each token is a vector with all zeros except for a single one at the index corresponding to the token's position in the vocabulary). These one-hot vectors are multiplied by a learned weight matrix, transforming them into continuous, lower-dimensional vectors, known as token embeddings. Then token embeddings are combined with positional embeddings to convey the position of each token in the sequence.

Transformer layer forward pass. As shown in Fig. 1, the 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:

$$Q = xW_q, \quad K = xW_k, \quad V = xW_v, \qquad W_q, W_k, W_v \in \mathbb{R}^{d\times d},$$

$$u = \text{SoftMax}\left(\frac{QK^T}{\sqrt{k}} + M\right)VW_o, \qquad M \in \mathbb{R}^{n\times n}, W_o \in \mathbb{R}^{d\times d},$$

$$v = \text{LayerNorm}(u + x; \gamma_1, \beta_1), \qquad \gamma_1, \beta_1 \in \mathbb{R}^d,$$

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:

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

Use of causal 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 [1], 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

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

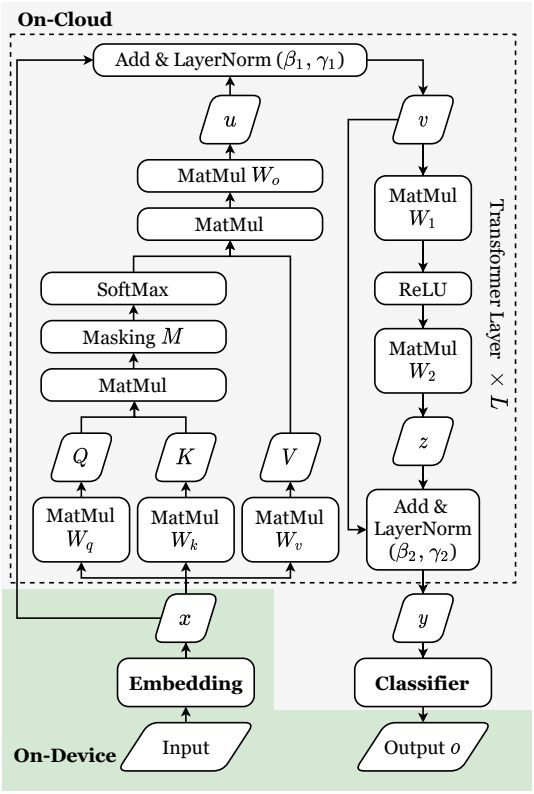


Fig. 1: Inference workflow of original Transformer.

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 (§ IV-A).

B. Auto-regressive Generation

After the classifier, the post-processing of output o depends on the specific task. For visual classification, we just select the class with the highest probability. In this paper, we are more concerned with autoregressive generation.

Auto-regressive generation is a widely-used approach in sequence modeling tasks, where the probability of a sequence $\mathbf{x} = (x_1, x_2, \ldots, x_T)$ is factorized using the chain rule of probability as $P(\mathbf{x}) = \prod_{t=1}^{T} P(x_t \mid x_1, x_2, \ldots, x_{t-1})$. During generation, tokens are sampled iteratively: at each step t, the model predicts the conditional probability $P(x_t \mid x_1, x_2, \ldots, x_{t-1})$, and the process continues until an end-of-sequence token (a predefined special token) is produced or a maximum length is reached. Transformer-based auto-regressive models, such as GPT, compute these conditional probabilities using a decoder-only architecture with causal masking to ensure that predictions at step t depend only on tokens up to $t-1$. This step-by-step approach enables flexible and expressive sequence generation but requires sequential decoding, which can be computationally intensive for long outputs.

C. Secure Two-Party 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 [27], [28], [29]. However, these approaches incur significant computational overheads and inevitable accuracy loss 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 and 90 GiB traffic to generate a single token with GPT2 model [27].

D. Align with Real-World Applications: Three-Party Inference

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 [39], 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 services.

Service-1: campus chatbot. At USTC, 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 the chatbot questions like, "Did any abnormal behavior occur during a certain period?" and get responses in natural language.

Service-2: vehicle cabin assistant. Collaborating with NIO 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 open-sourced parameters [6], [40] with our collected data. The computing power of our on-campus lab and company can afford offline model training, but cannot support large-scale long-term services. The campus chatbot has approximately a few hundred users, and the vehicle assistants need to serve hundreds of thousands of users. We, as model developers, need to resort to third-party cloud platforms to serve our Transformer models. The public cloud is trusted

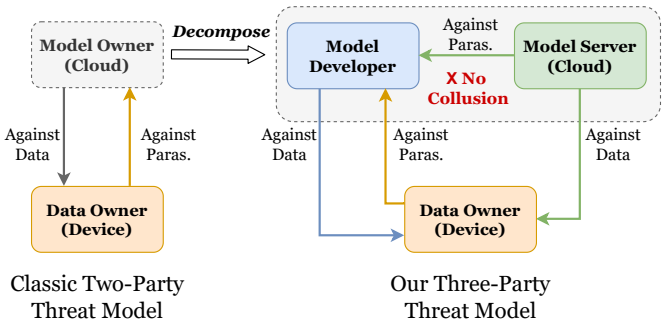


Fig. 2: Three-Party Threat Model. Paras. denotes model parameters.

and parameters are proprietary, there will be no collusion between the model developer and the cloud, see Fig. 2.

PCC: a three-party solution. Apple's Private Cloud Compute (PCC) [33] is a sophisticated architecture designed to enhance data privacy while enabling advanced inference functionalities across its devices. PCC ensures that user data remains inaccessible even to Apple administrators, theoretically, it is equivalent to PCC and Apple being two non-colluding parties. Therefore, PCC aligns with our three-party setting that involves Apple as the model developer, PCC cluster as the model server, and users as the data owner.

III. System Overview

Scope. This work focuses on achieving privacy-preserving and efficient inference for Transformer models, i.e., the end-to-end forward pass, under the three-party setting. Inference results can serve as input for downstream applications, e.g., AI agents that automatically call external APIs [41]. Potential privacy risks and efficiency issues in downstream applications are out of the scope of this work.

Design goals. Our system STIP has four main design goals:
(1) Data and parameter privacy. The foremost objective is to ensure the privacy of user data and model parameters.
(2) No accuracy loss. The system is required to perform accuracy-lossless inference, meaning there should be no approximation of any computations in Transformer models.
(3) Support production environments. The system must support inference frameworks used in production environments, incorporating techniques such as kv-cache for efficiency optimization [42], [43].
(4) Flexible extension to Transformer variants. Given the continuous evolution of Transformer models with various emerging variants [6], [4], [44], [38], [45], the system must possess the ability for flexible extension to accommodate Transformer variants. This ensures long-term availability without necessitating case-by-case adaptation.

A. Threat Model

For serving Transformer models, we consider three parties:
• Model developer (P_1) that trains and owns private Transformer model parameters.
• 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.

Given that the developed models are proprietary, model developers must safeguard their model parameters against attacks from untrusted public cloud [46]. We make the assumption that P_1 does not engage in collusion with P_2. In principle, our setting aligns with server-aided MPC [47] framework, which allows parties to securely outsource their computation to a third-party cloud provider.

We adopt the **semi-honest** setting where each party will correctly follow the algorithm but attempt to infer additional sensitive information from the observed messages. Semi-honest security is a widely adopted assumption for privacy-preserving inference [23], [27], [28], [48], [29].

The inference system should ensure that P_1 and P_2 are unaware of P_3's original input x_1 and inference output o. And P_1's original model parameters θ should be hidden to both P_2 and P_3. The parameter θ consists of L-layer attention weights ($\{W_q, W_k, W_v, W_o\}_L$), feedforward weights ($\{W_1, W_2\}_L$), normalization weights ($\{\gamma, \beta\}_L$), and classifier weights (W_c). In the context of Transformer, P_3's input can be text prompt [6], images [40], and a combination of multiple modalities [44] and the inference output is a probability vector of the last classification head [49].

B. Sub-Use Cases

Since we only assume that P_1 and P_2 do not collude, in addition to the above three-party scenario, STIP also subsumes the following two-party cases:

(1) $P_1 = P_3$, i.e., the model developer is also the data owner. Possible applications include a university laboratory that develops its own model and hopes to use a cloud platform to host the model for internal use by students. In this case, our system can prevent the cloud from obtaining the original model parameters and data.

(2) $P_2 = P_3$, i.e., the data owner is also the model server. This case fits a common scenario, e.g., a company develops a personal assistant model and deploys it on the user's mobile phone to process the on-device data locally. In this case, our system mainly protects private original parameters from being accessed.

Justification and implications of non-collusion. STIP's security relies only on the assumption that P_1 (model developer) and P_2 (model server) do not collude. If the assumption of no collusion between P_1 and P_2 fails, i.e., the developer and cloud server share their secrets, the privacy of P_3's input may be compromised. Such collusion reverts the system to a standard two-party setting, where stronger cryptographic protocols may be required. We acknowledge this limitation and view this as future work.

STIP is positioned for scenarios where P_1 and P_2 are distinct organizations with partially aligned incentives, a model adopted in a production-scale cloud operated by a different

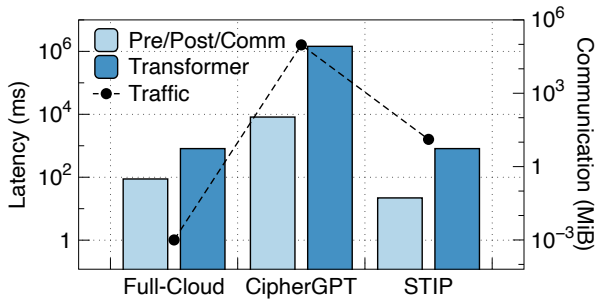


Fig. 3: Latency and communication overheads of GPT2-124m model under different inference systems. Pre/Post/-Comm refers to the latency costs incurred by pre-processing, post-processing and communication.

entity. In such cases, our semi-honest assumption and non-collusion model provide a pragmatic balance between privacy and performance.

IV. Design

This section first introduces how to perform equivalent inference of Transformer models with feature space permutation (§ IV-A). Next, we present the system workflow (§ IV-B), and analyze the privacy-preserving capability (§ IV-C, IV-D). Then we detail how to integrate TEE to further protect parameters from extraction attacks (§ IV-E).

A. Feature Space Permutation

Transformation of data and parameters is key to protection. Existing systems that combine HE and 2PC techniques have prohibitive computing and communication overheads. As shown in Fig. 3, CipherGPT [27] takes over 25 minutes and 90 GiB traffic to perform a single forward pass of GPT2 [4] 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 [50]. 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 [51] and the token-wise permutation invariance [52]. Drawing attention to the prevalent use of random permutation in addressing privacy concerns, including secure communication [53], machine learning [54], [48], [55], etc. In light of these observations, we present our first insight:

▷ ***Insight 1: Efficient feature-space permutation for privacy-preserving and lossless Transformer inference.***

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.

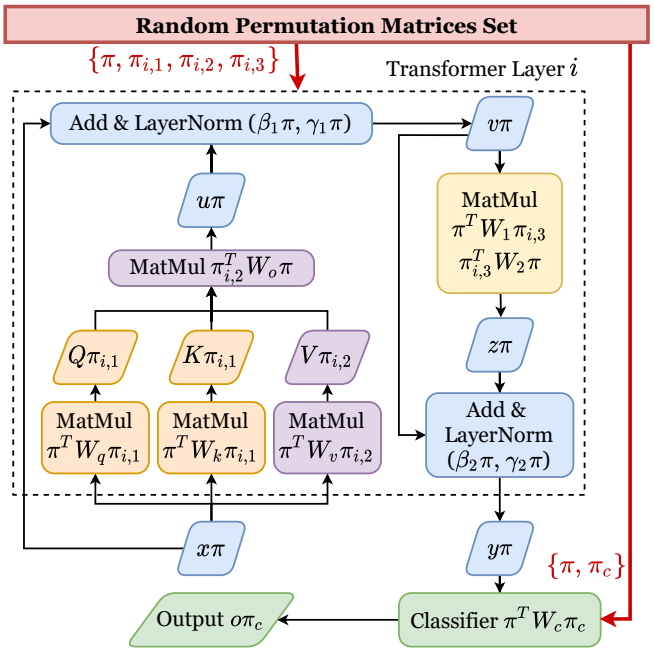


Fig. 4: Feature-space parameter transformation. Colors represent the use of different permutation matrices.

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 [52], [56]. 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 $\pi_{i,1}, \pi_{i,2}, \pi_{i,3}$:

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

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

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

Fig. 4 illustrates the parameter transformation procedure. The transformation can be efficiently implemented with $O(d)$-complexity movement of memory pointers, where d is the feature dimension. As Fig. 3 shows, our system STIP achieves orders of magnitude higher efficiency than CipherGPT, and the latency is close to full-cloud deployment.

TABLE II: 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 $\{\pi_1, ..., \pi_{3L}\}$	$d!$	$(d!)^{3L}$	✓	✓

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

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

This guarantees exact equivalence between original and transformed inference results. Due to page limitations, the proof is placed in Appendix A.

B. System Workflow

Semi-symmetrical sharing of permutation matrices. While random permutation-based schemes are efficient and accuracy-lossless, endowing them with robust privacy protection poses non-trivial challenges. A sequence-level permutation scheme [52] 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 that once the cloud gains a pair of known plaintexts of the original data and the transformed data, it can easily recover the permutation matrix, subsequently inverse-transforming all parameters and exposing sensitive information.

▷ ***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. Our proposed feature-space permutation scheme utilizes a set of matrices $\pi_1, ..., \pi_{3L}$, where L represents the number of layers. 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. Similar semi-symmetric protection schemes have been explored in domains such as image encryption [57] and online shopping [58]. Our analysis shows the resistance to BFA and KPA (§ IV-C). Tab. II summarizes the number of possible permutations for data and parameters and resistance to BFA and KPA attacks.

STIP workflow. Based on our proposed permutation-based transformation for Transformer models, we develop STIP system. Fig. 5 shows the workflow 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

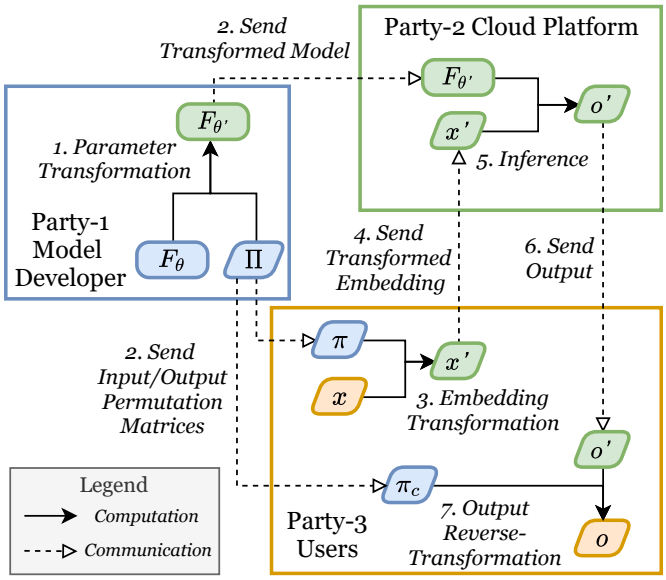


Fig. 5: Workflow of STIP.

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 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.

Production environment support. Production-level Transformer inference frameworks, such as DeepSpeed [42] and HuggingFace [59], 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. Importantly, STIP only transforms 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 code. As a result, STIP seamlessly aligns with production-level frameworks, and we have implemented STIP with the HuggingFace [59] library.

C. Attack Resistance

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 [6]. 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) [48]. 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 [60]).

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 [61], [62].

Malicious model server 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.

D. Privacy Leakage Upper Bound

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 privacy leakage, we employ distance correlation [36], a statistical measure of dependency between two random vectors. The distance correlation between u and v, is defined as

$$1 - \frac{(u - \bar{u}) \cdot (v - \bar{v})}{||(u - \bar{u})||_2 ||(v - \bar{v})||_2}$$

where $\bar{u}$ is the mean of the elements of u and $x \cdot y$ is the dot product of x and y. Let Corr denote the distance correlation. Based on existing theorem [48], it has been proven that:

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

Interpretation of privacy guarantee. The above distance correlation bound provides a quantitative measure of privacy: it guarantees that the dependency between the original data x and its permuted form $xA\pi$ is capped at a small value. In other words, an adversary cannot find much more statistical correlation between $xA\pi$ and x than they would between x and a random one-dimensional projection of x. This implies that only minimal information about x can leak through $xA\pi$. Practically, a distance correlation near zero means the transformed data reveals almost no linear or non-linear relationship to the original data, making inference of the original inputs infeasible in our threat model. We stress that this is a worst-case upper bound: it ensures any potential leakage is strictly limited. Such a guarantee is weaker than absolute cryptographic secrecy, but it has been shown to yield strong privacy in practice (a similar level of obfuscation is used in prior work for privacy-preserving data releases [63], [64]). In summary, our theoretical privacy guarantee indicates that STIP significantly reduces information leakage, quantifiably bounding an attacker's ability to correlate transformed data with the sensitive original inputs.

Model split considerations. By default, STIP only deploys the embedding module on user devices, as shown in Fig. 1. This decision is motivated by the fact that splitting the model before the embedding poses privacy 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 and does not reduce theoretical privacy leakage.

E. TEE Integration for Enhanced Protection

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.

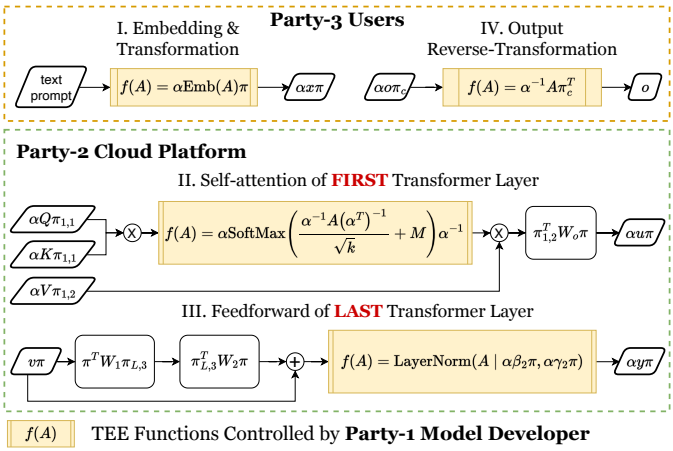


Fig. 6: Integration of TEE into STIP.

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) [65], [66] are secure areas within a processor that provide isolated environments for executing 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 [67]. However, our experiments (see Sec. VI-F) show that attackers can still use authorized input-output samples to perform model extraction [14] and fine-tuning attacks [68] on the parameters protected by TEE.

Introducing one-time randomness. Existing work [69], [70] 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 privacy protection 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, given the user's text prompt, we perform the embedding and additionally transform the embedding x using a one-time random diagonal matrix α: $f(A) = \alpha\text{Emb}(A)\pi$. 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:

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

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: $f(A) = \text{LayerNorm}(A|\alpha\beta_2\pi, \alpha\gamma_2\pi)$. 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 equivalence 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.

Preventing parameter leakage via TEE. In STIP, the embedding generation process, from raw input prompt to token embeddings x, is executed inside the TEE enclave. As a result, the user cannot access the plaintext x directly and only sees the transformed embedding $x' = \alpha x\pi$. The introduction of the one-time random diagonal matrix α serves a specific purpose: it thwarts attacks that attempt to use known prompts and their corresponding outputs to reconstruct the embedding module or permutation matrix. Without α, an attacker could provide a prompt p, observe the deterministic transformed embedding $x\pi$, and collect sufficient (prompt, output) pairs to approximate the embedding function or learn π through regression. However, with per-query randomness α, each prompt results in a randomly transformed embedding, even if the prompt is repeated. Thus, the attacker can no longer accumulate consistent training data, and the embedding function remains unrecoverable.

This design ensures that even though permutation π and scaling α transformations preserve some properties of embeddings (e.g., vector angle) between token vectors, the attacker cannot exploit this, because the underlying embeddings x are never exposed. The one-time randomness α effectively breaks any structure the server could learn from observing the transformed embedding, making surrogate model extraction infeasible. In summary, STIP eliminates parameter leakage by (1) isolating the embedding computation in the TEE, and (2) applying per-query randomness that destroys repeatability and structure across queries.

TEE Trust and Limitations. Our privacy guarantee of model functionality relies on a trusted execution environment. It is important to clarify the assumptions and limitations of this reliance. First, we assume the TEE itself is secure, i.e., the enclave's code and data cannot be accessed or influenced by the cloud server outside of the allowed interface. In practice, however, TEEs are known to be vulnerable to certain side-channel attacks (e.g., timing and memory access pattern leaks, cache side-channels, speculative execution vulnerabilities). A determined attacker with low-level access to the cloud server could exploit such channels to attempt to infer enclave-protected information. STIP design minimizes TEE usage to reduce this attack surface, but it does not entirely remove the risk. We thereby inherit the typical trust assumption of TEEs, including trusting the hardware manufacturer and the system's firmware to be free of known leaks. In a real deployment, one should ensure the enclave is kept up-to-date against vulnerabilities and consider side-channel defenses if needed.

We clarify that while the TEE integration boosts security (resisting direct model extraction and fine-tuning attacks), it is not a silver bullet: the approach provides a practical trade-off, improving privacy under a semi-honest threat model but still requiring trust in the TEE.

V. STIP for Transformer Variants

This section discusses how STIP supports various Transformer variants, including language models (§ V-A), multi-modal models (§ V-B), and mixture-of-experts models (§ V-C). Following that, we establish generalized rules and claim the application scope of STIP (§V-D).

Due to page limitations, all proofs are placed in Appendix. A.

A. Language Models

Pre-LayerNorm. The first version of GPT directly adopts the original Transformer decoder. GPT-2 [4] 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 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 in the feedforward sub-block of the last Transformer layer into the enclave. Formally, function $f(A) = \alpha\text{ReLU}(\alpha^{-1}A)$ is deployed.

RMSNorm. LLaMA series [6] 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 [6] uses SwiGLU [72] 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 $\text{FFN}'_{\text{SwiGLU}}(x\pi) = \text{FFN}_{\text{SwiGLU}}(x)\pi$.

B. Multi-Modal Models

ViT [40] 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 preprocessing of the original data, it can seamlessly support ViT. LLaVA [44] 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_vW, 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.

C. Mixture-of-Experts Models

Mixtral [38] 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 $\pi^T W_g$.

D. 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.

VI. Evaluation

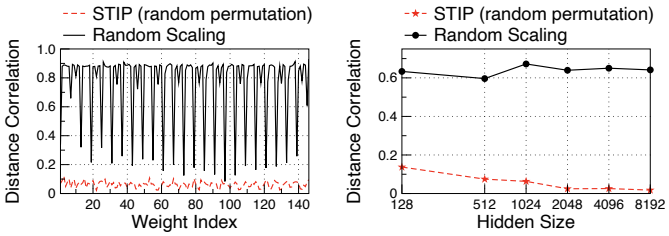
A. Implementation

We implemented STIP using PyTorch and HuggingFace [59] libraries, and open-sourced the code at https://github.com/yuanmu97/secure-transformer-inference. In addition, our collaborators provide a MindSpore-based implementation built on Ascend (https://gitee.com/mindspore/mindarmour), demonstrating that our design is framework-agnostic.

Modern deep learning frameworks, including PyTorch, adopt a row-major memory layout. To align with the layout, PyTorch performs matrix multiplication in the linear layer as $xW^\intercal$ instead of xW. Consequently, we transpose the previously introduced parameter transformation for implementation. For permutation operations, we generate a random permutation vector π_v instead of a matrix Π_m. This vector is then used to index rows or columns, e.g., $W[:, \pi_v]$, which achieves the same effect as $W\Pi_m$ but is more efficient than dense matrix multiplication. See Appendix B for a code example that transforms GPT-2 model parameters in a HuggingFace implementation.

TABLE III: Summary of Testbeds and Transformer Models

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



(a) Parameters of GPT2 (b) Embeddings

Fig. 7: Privacy leakage measurement: distance correlation.

B. Experimental Setup

Testbeds and Transformer models. We use three testbeds and six representative Transformer models for evaluation, see Tab. III. (1) Campus Security Chatbot (CHAT). We select pretrained LLaMA2-7b [6] to host this service at USTC. To scale the evaluation, we also deployed GPT2-124m/355m/774m/1.5b [4] and LLaMA2-13b/70b models[2]. (2) Vehicle Cabin Scene Understanding (CABIN). We use LLaVA-13b [44] 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 [40] for comprehensive experiments. (3) Chatbot. To further evaluate STIP on BERT series [5] and Mixtral [38] models, we build a chatbot testbed.

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 [28], THE-X [29], and CipherGPT [27]. They propose secure two-party protocols for serving BERT series and GPT-2 models. (5-7) Bumblebee [26], Puma [25], and NEXUS [30]. 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 Chatbot, we use a MacBook Pro laptop with 4-core Intel Core i5 CPUs as the user device.

C. Privacy and Accuracy

First, we evaluate the previously analyzed privacy protection and computational equivalence through experiments.

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

Bounded distance correlation. We employ distance correlation [36] as the metric for assessing privacy leakage. As a baseline, we utilize random element-wise scaling on both parameters and embeddings, referred to as Random Scaling:

$$x_{\text{scaled}} = r \odot x, \quad \text{where } r \sim \mathcal{U}(0,1)^d$$

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 Scaling. On average, Random Scaling yields a distance correlation of 0.76, while STIP achieves a markedly lower value of 0.062. To evaluate the privacy 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 Scaling, 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. IV-C.

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 IV, 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 .

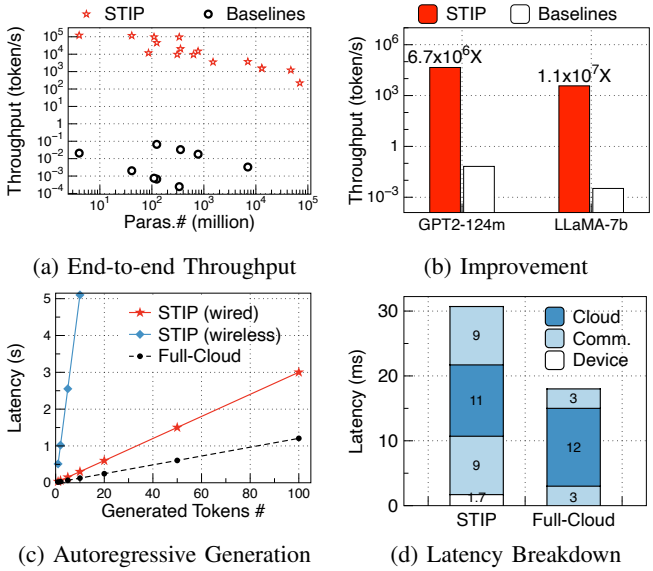
D. Inference Efficiency

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 [27], [28], [29], [25], [26]. 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

TABLE IV: STIP has the guarantee of lossless accuracy. Numerical differences arise from floating-point arithmetic errors. Class accuracy = 100% means that the transformed and original model yield identical predictions.

Model	GPT-2	LLaMA2	ViT	BERT	LLaVA	Mixtral
Num. of Parameters	124m/355m/774m/1.5b	7b/13b/70b	86m/307m/632m	4m/41m/110m/336m	13b	47b
Absolute Difference	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 Accuracy	100%	100%	100%	100%	100%	100%



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

(c) Autoregressive Generation (d) Latency Breakdown

Fig. 8: Efficient serving Transformers with STIP .

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. IV-B, the communication cost per generated token is inevitable to ensure output privacy protection. Considering the practical privacy protection that STIP introduces compared to unprotected full-cloud inference, the slightly higher latency (e.g., 2s more for 100 tokens) is deemed acceptable.

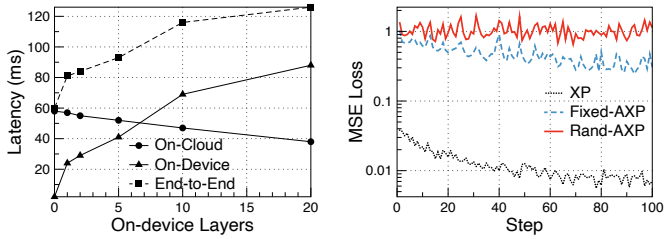
Latency breakdown. To gain deeper insights into the overhead introduced by STIP , we conducted an analysis of latency breakdown, comparing it against full-cloud inference. Illustrated in Fig. 8d, our evaluation reveals that STIP introduces an additional 1.7ms latency on the device while concurrently 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 [73] 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.

E. 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. 3, 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 privacy protection 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. IV-C, 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



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

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

minimizes latency while mitigating the potential privacy risks of exposing more layers to the user.

F. 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 authorized inference to obtain the input and output samples of the initial transformation 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 [74] 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. 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 [74] for experiments. As shown in Tab. V, neither fine-tuning all parameters nor training additional linear layers can effectively restore the model performance (worse than training a model from scratch).

Inference latency. We use NVIDIA A100 GPUs and Intel Xeon 6330 CPUs for 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. VI 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 protection of model parameters brought by integrating TEE, this efficiency loss is completely acceptable.

TABLE V: 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 VI: Inference latency with and without TEE in STIP .

Model	10x100 tokens inference latency (s)		
	w/o TEE	w/ TEE (all)	w/ TEE (IO-only)
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%)

G. Real-World Deployment

We have deployed STIP in a production environment to serve real users through a commercial cloud platform. The deployment leverages a proprietary 70B Transformer model, integrated into a chatbot service system that supports private document Q&A. In the production setting, STIP runs entirely on commodity cloud hardware (A100 servers). To support multi-request concurrency, we maintained 10 independent model transformation contexts, each corresponding to a distinct permutation configuration. Over a three-month evaluation period, STIP processed more than 100 million user tokens. The average end-to-end latency per token was approximately 105 ms. Compared to the plaintext version of the service, the deployment incurred only a 12% increase in latency, well within acceptable bounds for interactive applications. The system has been operating continuously for several months without any reported privacy incidents, demonstrating STIP's robustness, scalability, and suitability for production-grade AI workloads.

VII. 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 [26], [16], [17], [18], [19], [20]. Recent research [56] proposes using permutations to enhance

protection further. However, despite this split, the potential for reverse-engineering sensitive information from intermediate activations [21], such as text embeddings [22], remains a concern. STIP builds upon the concept of model split and goes a step further by incorporating random permutation, offering theoretically enhanced privacy protection.

Secure Transformer inference. In the context of a two-party setting, prior efforts [24], [26], [25], [27], [28], [29] 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 [32] 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 adopts a three-party threat model and employs a semi-symmetrical permutation scheme for protection instead of cryptographic primitives.

Confidential computing. Confidential computing has emerged as a critical paradigm for ensuring the privacy of model inference. Secure enclaves, such as Intel SGX and ARM TrustZone, enable secure processing by isolating sensitive computations from the rest of the system. Apple's private cloud compute [33] solution ensures that user data remains inaccessible even to the cloud provider during model execution. In these approaches, the calculation between parameters and data is executed in TEE in plain text. Our STIP is different from this, the computation happens on transformed data and parameters, so no additional hardware trust is required.

VIII. Conclusion

In this paper, we studied privacy concerns in Transformer inference. We proposed a three-party threat model and presented the design of STIP, a privacy-preserving transformer inference system based on our semi-symmetrical permutation scheme. Theoretical analysis and experiments in real-world production environment evaluated STIP's practical privacy protection, accuracy lossless, scalability and computational efficiency.

Acknowledgment

This research was supported by the National Key R&D Program of China 2021YFB2900103, 2021ZD0110400, China National Natural Science Foundation with No. 623B2093, 62441228, 62132018 , 62231015, Science and Technology Tackling Program of Anhui Province, No.202423k09020016, Innovation Program for Quantum Science and Technology 2021ZD0302900 and "Pioneer" and "Leading Goose" R&D Program of Zhejiang, No. 2023C01029, 2023C01143, USTC Kunpeng&Ascend Center of Excellence, RGC Theme-based Research Scheme T43-513/23-N, and Strategic Topics Grant STG1/E-403/24-N.

References

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[2] J. Gao, W. Ji, F. Chang, S. Han, B. Wei, Z. Liu, and Y. Wang, "A systematic survey of general sparse matrix-matrix multiplication," *ACM Computing Surveys*, vol. 55, no. 12, pp. 1–36, 2023.

[3] J. Wei, Y. Tay, R. Bommasani, C. Raffel, B. Zoph, S. Borgeaud, D. Yogatama, M. Bosma, D. Zhou, D. Metzler *et al.*, "Emergent abilities of large language models," *Transactions on Machine Learning Research*, 2022.

[4] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[5] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[6] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar *et al.*, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.

[7] OpenAI, "Chatgpt," https://openai.com/blog/chatgpt, 2022.

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

[9] Microsoft, "Microsoft 365 copilot," https://blogs.microsoft.com/blog/2023/03/16/introducing-microsoft-365-copilot-your-copilot-for-work/, 2023.

[10] S. Chen, M. Xu, J. Ren, Y. Cong, S. He, Y. Xie, A. Sinha, P. Luo, T. Xiang, and J.-M. Perez-Rua, "Gentron: Diffusion transformers for image and video generation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 6441–6451.

[11] F. Romero, Q. Li, N. J. Yadwadkar, and C. Kozyrakis, "Infaas: Automated model-less inference serving," in *2021 USENIX Annual Technical Conference*, 2021, pp. 397–411.

[12] 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.

[13] T. Nayan, Q. Guo, M. Al Duniawi, M. Botacin, S. Uluagac, and R. Sun, "Sok: All you need to know about on-device ml model extraction-the gap between research and practice," in *33rd USENIX Security Symposium (USENIX Security 24)*, 2024, pp. 5233–5250.

[14] W. Jiang, H. Li, G. Xu, T. Zhang, and R. Lu, "A comprehensive defense framework against model extraction attacks," *IEEE Transactions on Dependable and Secure Computing*, vol. 21, no. 2, pp. 685–700, 2023.

[15] Z. Zhang, Y. Chen, and D. Wagner, "Seat: Similarity encoder by adversarial training for detecting model extraction attack queries," in *Proceedings of the 14th ACM Workshop on artificial intelligence and security*, 2021, pp. 37–48.

[16] P. Jiang, K. Xin, C. Li, and Y. Zhou, "High-efficiency device-cloud collaborative transformer model," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 2203–2209.

[17] C. Thapa, P. C. M. Arachchige, S. Camtepe, and L. Sun, "Splitfed: When federated learning meets split learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 8, 2022, pp. 8485–8493.

[18] L. Zeng, X. Chen, Z. Zhou, L. Yang, and J. Zhang, "Coedge: Cooperative dnn inference with adaptive workload partitioning over heterogeneous edge devices," *IEEE/ACM Transactions on Networking*, vol. 29, no. 2, pp. 595–608, 2020.

[19] N. D. Pham, A. Abuadbba, Y. Gao, T. K. Phan, and N. Chilamkurti, "Binarizing split learning for data privacy enhancement and computation reduction," *IEEE Transactions on Information Forensics and Security*, 2023.

[20] D. Pasquini, G. Ateniese, and M. Bernaschi, "Unleashing the tiger: Inference attacks on split learning," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 2113–2129.

[21] S. Abuadbba, K. Kim, M. Kim, C. Thapa, S. A. Camtepe, Y. Gao, H. Kim, and S. 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*, 2020, pp. 305–318.

[22] X. Pan, M. Zhang, S. Ji, and M. Yang, "Privacy risks of general-purpose language models," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 1314–1331.

[23] Q. Pang, J. Zhu, H. Möllering, W. Zheng, and T. Schneider, "Bolt: Privacy-preserving, accurate and efficient inference for transformers," in

2024 IEEE Symposium on Security and Privacy (SP). IEEE, 2024, pp. 4753–4771.

[24] B. Yan, K. Li, M. Xu, Y. Dong, Y. Zhang, Z. Ren, and X. Cheng, "On protecting the data privacy of large language models (llms): A survey," *arXiv preprint arXiv:2403.05156*, 2024.

[25] Y. Dong, W.-j. Lu, Y. Zheng, H. Wu, D. Zhao, J. Tan, Z. Huang, C. Hong, T. Wei, and W. Cheng, "Puma: Secure inference of llama-7b in five minutes," *arXiv preprint arXiv:2307.12533*, 2023.

[26] W.-j. Lu, Z. Huang, Z. Gu, J. Li, J. Liu, K. Ren, C. Hong, T. Wei, and W. Chen, "Bumblebee: Secure two-party inference framework for large transformers," *Cryptology ePrint Archive*, 2023.

[27] X. Hou, J. Liu, J. Li, Y. Li, W.-j. Lu, C. Hong, and K. Ren, "Ciphergpt: Secure two-party gpt inference," *Cryptology ePrint Archive*, 2023.

[28] M. Hao, H. Li, H. Chen, P. Xing, G. Xu, and T. Zhang, "Iron: Private inference on transformers," *Advances in Neural Information Processing Systems*, vol. 35, pp. 15 718–15 731, 2022.

[29] T. Chen, H. Bao, S. Huang, L. Dong, B. Jiao, D. Jiang, H. Zhou, J. Li, and F. Wei, "The-x: Privacy-preserving transformer inference with homomorphic encryption," *arXiv preprint arXiv:2206.00216*, 2022.

[30] J. Zhang, X. Yang, L. He, K. Chen, W.-j. Lu, Y. Wang, X. Hou, J. Liu, K. Ren, and X. Yang, "Secure transformer inference made non-interactive," *Network and Distributed System Security (NDSS) Symposium*, 2025.

[31] LMArena, "Chatbot arena llm leaderboard," https://lmarena.ai/, 2024.

[32] K. Gupta, N. Jawalkar, A. Mukherjee, N. Chandran, D. Gupta, A. Panwar, and R. Sharma, "Sigma: Secure gpt inference with function secret sharing," in *2024 Privacy Enhancing Technologies Symposium*, 2024, pp. 61–79.

[33] Apple, "Private cloud compute: A new frontier for ai privacy in the cloud," 2024. [Online]. Available: https://security.apple.com/blog/private-cloud-compute/

[34] X. Cao, T. Başar, S. Diggavi, Y. C. Eldar, K. B. Letaief, H. V. Poor, and J. Zhang, "Communication-efficient distributed learning: An overview," *IEEE journal on selected areas in communications*, vol. 41, no. 4, pp. 851–873, 2023.

[35] J. He, K. Yang, G. Tang, Z. Huang, L. Lin, C. Wei, Y. Yan, and W. Wang, "Rhombus: Fast homomorphic matrix-vector multiplication for secure two-party inference," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, 2024, pp. 2490–2504.

[36] G. J. Székely, M. L. Rizzo, and N. K. Bakirov, "Measuring and testing dependence by correlation of distances," *arXiv preprint arXiv:0803.4101*, 2008.

[37] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[38] A. Q. Jiang, A. Sablayrolles, A. Roux, A. Mensch, B. Savary, C. Bamford, D. S. Chaplot, D. de las Casas, E. B. Hanna, F. Bressand, G. Lengyel, G. Bour, G. Lample, L. R. Lavaud, L. Saulnier, M.-A. Lachaux, P. Stock, S. Subramanian, S. Yang, S. Antoniak, T. L. Scao, T. Gervet, T. Lavril, T. Wang, T. Lacroix, and W. E. Sayed, "Mixtral of experts," 2024.

[39] M. Prabhakaran and M. Rosulek, "Cryptographic complexity of multi-party computation problems: Classifications and separations," in *Advances in Cryptology – CRYPTO 2008*, D. Wagner, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 262–279.

[40] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," *ICLR*, 2021.

[41] J. Ruan, Y. Chen, B. Zhang, Z. Xu, T. Bao, G. Du, S. Shi, H. Mao, Z. Li, X. Zeng, and R. Zhao, "Tptu: Large language model-based ai agents for task planning and tool usage," 2023.

[42] R. Y. Aminabadi, S. Rajbhandari, A. A. Awan, C. Li, D. Li, E. Zheng, O. Ruwase, S. Smith, M. Zhang, J. 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*. IEEE, 2022, pp. 1–15.

[43] O. Kuchaiev, J. Li, H. Nguyen, O. Hrinchuk, R. Leary, B. Ginsburg, S. Kriman, S. Beliaev, V. Lavrukhin, J. Cook *et al.*, "Nemo: a toolkit for building ai applications using neural modules," *arXiv:1909.09577*, 2019.

[44] H. Liu, C. Li, Q. Wu, and Y. J. Lee, "Visual instruction tuning," 2023.

[45] I. Beltagy, M. E. Peters, and A. Cohan, "Longformer: The long-document transformer," *arXiv preprint arXiv:2004.05150*, 2020.

[46] C. Zhang, J. Xia, B. Yang, H. Puyang, W. Wang, R. Chen, I. E. Akkus, P. Aditya, and F. Yan, "Citadel: Protecting data privacy and model confidentiality for collaborative learning," in *Proceedings of the ACM Symposium on Cloud Computing*, ser. SoCC '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 546–561. [Online]. Available: https://doi.org/10.1145/3472883.3486998

[47] S. Kamara, P. Mohassel, and M. Raykova, "Outsourcing multi-party computation," *Cryptology ePrint Archive*, 2011.

[48] F. Zheng, C. Chen, X. Zheng, and M. Zhu, "Towards secure and practical machine learning via secret sharing and random permutation," *Knowledge-Based Systems*, vol. 245, p. 108609, 2022.

[49] W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong *et al.*, "A survey of large language models," *arXiv preprint arXiv:2303.18223*, 2023.

[50] A. M. Rush, "The annotated transformer," in *Proceedings of workshop for NLP open source software (NLP-OSS)*, 2018, pp. 52–60.

[51] S. K. Ainsworth, J. Hayase, and S. Srinivasa, "Git re-basin: Merging models modulo permutation symmetries," *arXiv preprint arXiv:2209.04836*, 2022.

[52] J. Lee, Y. Lee, J. Kim, A. Kosiorek, S. Choi, and Y. W. Teh, "Set transformer: A framework for attention-based permutation-invariant neural networks," in *International conference on machine learning*. PMLR, 2019, pp. 3744–3753.

[53] U. Bhargava, A. Sharma, R. Chawla, and P. Thakral, "A new algorithm combining substitution & transposition cipher techniques for secure communication," in *2017 International Conference on Trends in Electronics and Informatics (ICEI)*. IEEE, 2017, pp. 619–624.

[54] T. Maekawa, A. Kawamura, Y. Kinoshita, and H. Kiya, "Privacy-preserving svm computing in the encrypted domain," in *2018 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*. IEEE, 2018, pp. 897–902.

[55] Q. He, W. Yang, B. Chen, Y. Geng, and L. Huang, "Transnet: Training privacy-preserving neural network over transformed layer," *Proceedings of the VLDB Endowment*, vol. 13, no. 12, pp. 1849–1862, 2020.

[56] H. Xu, L. Xiang, H. Ye, D. Yao, P. Chu, and B. Li, "Permutation equivariance of transformers and its applications," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 5987–5996.

[57] X. Di, J. Li, H. Qi, L. Cong, and H. Yang, "A semi-symmetric image encryption scheme based on the function projective synchronization of two hyperchaotic systems," *PloS one*, vol. 12, no. 9, p. e0184586, 2017.

[58] N. Fares and S. Askar, "A novel semi-symmetric encryption algorithm for internet applications," *Journal of University of Duhok*, vol. 19, no. 1, pp. 1–9, 2016.

[59] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush, "Huggingface's transformers: State-of-the-art natural language processing," 2020.

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

[61] T. Zhao, Q. Ran, L. Yuan, Y. Chi, and J. Ma, "Information verification cryptosystem using one-time keys based on double random phase encoding and public-key cryptography," *Optics and Lasers in Engineering*, vol. 83, pp. 48–58, 2016.

[62] T. Bianchi, V. Bioglio, and E. Magli, "Analysis of one-time random projections for privacy preserving compressed sensing," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 2, pp. 313–327, 2015.

[63] S. R. Oliveira and O. 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*, 2004, pp. 40–46.

[64] Y. Wang, Y.-X. Wang, and A. Singh, "A theoretical analysis of noisy sparse subspace clustering on dimensionality-reduced data," *IEEE Transactions on Information Theory*, vol. 65, no. 2, pp. 685–706, 2018.

[65] P. Jauernig, A.-R. Sadeghi, and E. Stapf, "Trusted execution environments: properties, applications, and challenges," *IEEE Security & Privacy*, vol. 18, no. 2, pp. 56–60, 2020.

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

[67] Q. Li, Z. Shen, Z. Qin, Y. Xie, X. Zhang, T. Du, and J. Yin, "Translinkguard: Safeguarding transformer models against model stealing in edge deployment," *arXiv preprint arXiv:2404.11121*, 2024.
[68] T. Lee, B. Edwards, I. Molloy, and D. Su, "Defending against neural network model stealing attacks using deceptive perturbations," in *2019 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2019, pp. 43–49.
[69] F. Yang, Z. Chen, and A. 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*, 2019, pp. 25–35.
[70] S. Zhou, T. Zhu, D. Ye, W. Zhou, and W. Zhao, "Inversion-guided defense: Detecting model stealing attacks by output inverting," *IEEE Transactions on Information Forensics and Security*, 2024.
[71] B. Zhang and R. Sennrich, "Root mean square layer normalization," *Advances in Neural Information Processing Systems*, vol. 32, 2019.
[72] N. Shazeer, "Glu variants improve transformer," *arXiv preprint arXiv:2002.05202*, 2020.
[73] S. Yao, J. Li, D. Liu, T. Wang, S. Liu, H. Shao, and T. 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*, 2020, pp. 476–488.
[74] S. Merity, C. Xiong, J. Bradbury, and R. Socher, "Pointer sentinel mixture models," 2016.

APPENDIX

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., $\text{ReLU}(x\pi)=\text{ReLU}(x)\pi$ and $\text{SoftMax}(x\pi)=\text{SoftMax}(x)\pi$. Next, we prove that:

$\text{LayerNorm}(x\pi;\gamma\pi,\beta\pi) = \text{LayerNorm}(x;\gamma,\beta)\pi$.

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:

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

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 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 $\{\pi_{i,1}, \pi_{i,2}, \pi_{i,3} \ | i \in [L]\}$ transformations and only kept the transformations of π and π_c.