Securing Transformer-based AI Execution via Unified TEEs and Crypto-protected Accelerators

Jiaqi Xue, Yifei Zhao, Mengxin Zheng, Fan Yao, Yan Solihin, Qian Lou
University of Central Florida

***Abstract*—Recent advances in Transformer models, e.g., large language models (LLMs), have brought tremendous breakthroughs in various artificial intelligence (AI) tasks, leading to their wide applications in many security-critical domains. Due to their unprecedented scale and prohibitively high development cost, these models have become highly valuable intellectual property for AI stakeholders and are increasingly deployed via machine learning as a service (MLaaS). However, MLaaS often runs on untrusted cloud infrastructure, exposing data and models to potential breaches. Mainstream protection mechanisms leverage trusted execution environments (TEEs) where confidentiality and integrity for secretive data are shielded using hardware-based encryption and integrity checking. Unfortunately, running model inference entirely within TEEs is subject to non-trivial slowdown, which is further exacerbated in LLMs due to the substantial computation and memory footprint involved. Recent studies reveal that the hybrid TEE-based scheme offloading partial model inference operations to the untrusted accelerators (e.g., GPU) is a promising solution. However, prior offloading schemes fail to ensure dual protection of data and model in Transformer inference, as they cannot securely offload critical operations, i.e., `Attention` and `SoftMax`, forcing these computations to remain confined within TEEs. To address these challenges, we propose** TwinShield**, a framework enabling secure Transformer inference in heterogeneous TEE and accelerator systems with dual protection for both model and data.** TwinShield **offloads $\sim$ 87% of computation to GPUs and delivers $4.0\times$ to $6.1\times$ speedups over previous approaches across various Transformer models.**

I. Introduction

Transformers [1] have demonstrated outstanding performance on a wide range of domains including computer vision [2] and natural language processing [3], [4], which are the building blocks for many emerging applications such as chatbots [5] and medical image analysis [6], [7]. As Transformers become increasingly popular, the *confidentiality* and *integrity* of the inference services become a critical concern, especially in confidentiality-sensitive sectors such as healthcare [6], [7], finance [8], and personal assistant applications [9], [10]. Given the substantial size and deployment complexity of these models, cloud-based Transformer-as-a-Service (TaaS) has emerged as a widely adopted solution for end users to access these state-of-the-art models in a cost-efficient way [11], [12], [13].

In these services, data provided by clients, such as personal health information (including sleep patterns, pulse, and heart rate) and banking records, are highly private. However, it is widely known that remote computation (as in the cloud) may not be trusted as adversaries can exploit either privileged system software [14] or hardware vulnerabilities [15] to compromise data privacy and integrity. This becomes particularly worrisome for Transformer-based systems where exposure of sensitive data can result in tremendous breaches of personal privacy (e.g., for clients). Moreover, an adversary may maliciously tamper with the model and its computation, leading to severe integrity compromise that introduces catastrophic system consequences. In summary, offering data confidentiality and inference integrity is imperative.

Trusted Execution Environments (TEEs), such as Intel SGX [16], [17], offer an environment for safeguarding the privacy (and sometimes integrity) for sensitive computation. In systems with TEEs, the CPU is treated as the root of trust. The processor shields individual secure domains (i.e., enclaves) from privileged system software attacks via hardware-enforced isolation. Furthermore, counter-mode encryption and integrity tree-based data verification are performed by the TEE-hardware to protect the breach and tampering of off-chip data belonging to enclaves [17]. Accordingly, prior studies have investigated the use of TEEs for secure machine learning inference [18], [19], [20], [21]. For instance, MLCapsule [18] proposes to store CNN models in enclave and perform model inference completely in TEEs, hence protecting computation integrity and the confidentiality of all data. Unfortunately, deployment of the entire ML model inside TEEs introduces extremely high overhead due to the limited resources available within TEEs. Subsequent works [19], [22], [20], [21], [23], [24] attempt to improve the performance of TEE-based model inference by *outsourcing* certain computation from TEEs to an untrusted external accelerator (e.g., GPUs, FPGAs and ASICs), and *verificating* the computation integrity inside the enclave. While the aforementioned secure ML outsourcing techniques enhance system performance of TEE-only methods, they struggle to outsource sufficient computations to untrusted accelerators from trusted TEEs. The challenges are summarized as follows:

(I) Confidentiality Challenge: Multiplicative Linear Operations, e.g., Transformer's Attention. We categorize linear operations into two types: additive and multiplicative. Additive operations, such as matrix multiplication involving one *variable* matrix and one *constant* matrix. Multiplicative operations involve matrix multiplication where neither operand is a constant matrix. Prior works show the feasibility of securing execution of the additive operation on untrusted accelerators via secret sharing and Freivalds' algorithm [25], which works well for the traditional convolutional neural networks (CNNs) since the convolution of fixed constant pre-trained kernels and variable inputs is additive operations. However, Transformers

include massive multiplicative linear operations where neither operands are constant matrices, rendering prior techniques inapplicable;

(II) Confidentiality Challenge: Non-linear Operations, e.g., `SoftMax`*.* We find that different from CNNs where operations in linear layers overwhelmingly dominate the computation for model inference ($> 98\%$) [19], non-linear operators in Transformers (i.e., `SoftMax`) contribute non-trivial computation overhead, especially for inputs with long tokens. Therefore, under an outsourcing scheme with linear operation-only offloading, the execution of such non-linear functions in TEEs will become the new bottleneck. As a result, it is necessary to further outsource `SoftMax` operations for further unleash the performance advantages of off-chip accelerators. Unfortunately, none of the prior mechanisms is able to outsource non-linear operations while maintaining proper model privacy and integrity at the same time.

(III) Integrity Challenge: Effective Verification for Multiplicative linear and Non-linear Operations. Prior works [19], [22], [20] rely on Freivalds' algorithm [25] to guarantee the integrity of the outsourced matrix computation, i.e., based on the matrix multiplication's associative law, $A \cdot B = C$ can be verified by $A \cdot (B \cdot r) == C \cdot r$, where A, B, C are matrices, r is a vector. However, it cannot be applied to the non-linear `SoftMax` in Transformers, which are element-wise operations, not matrix multiplications; hence, a new integrity mechanism is needed.

To address the above challenges, in this paper, we propose TwinShield, to enable a confidential and verifiable Transformer inference. The client uploads private data to the cloud server, which performs Transformer inference within the trusted TEEs and untrusted accelerators. TwinShield's protocol enables most Transformer computations to run on accelerators while ensuring data confidentiality and computation integrity. Our protocols and contributions are summarized as follows:

- For challenge (I), we design a confidentiality-guaranteed algorithm, *OutAttnMult*, to securely outsource multiplicative attention operations to an untrusted accelerator. Our algorithm transforms multiplicative linear operations into additive computation with a few pre-computed offline computations, enabling secure outsourcing of these computations.
- For Challenge (II), we propose a secure `SoftMax` outsourcing algorithm, *OutSoftMax*, which offloads its primary computational component (exponentiation) while retaining only a few additions and divisions within TEEs.
- For Challenge (III), we design *U-Verify* that guarantees the integrity of outsourced computation (both linear and non-linear). For the non-linear `SoftMax` function particularly, we propose a new *check product* protocol. *U-Verify* also improves efficiency in linear operations compared to prior methods.
- Through extensive experiments on various models, such as vision, language, and multi-modal Transformers, we show TwinShield achieves substantial throughput improvements ranging from $4.9\times$ to $7.7\times$ for private inferences, and from $3.9\times$ to $6.1\times$ for private verifiable inferences, without sacrificing accuracy.

II. Threat Model

We consider an outsourcing scheme between a client-side data owner $\mathcal{C}$ and a server $\mathcal{S}$, where $\mathcal{S}$ executes a Transformer model $f(x) : X \rightarrow Y$ on data provided by $\mathcal{C}$. The model $f(\cdot)$ can belong to the server (e.g., in SaaS/API [26], [27], [28]). We adopt a realistic threat model in which the server $\mathcal{S}$ is not fully trustworthy and may be malicious or vulnerable to tampering with the computation results $f(x)$. This departs from the traditional semi-honest setting, in which $\mathcal{S}$ is assumed to be honest but curious about inferring $\mathcal{C}$'s data privacy. An ideal protection scheme should satisfy the following security properties: **Data Privacy**: $\mathcal{S}$ cannot learn any information about input x. **Integrity & Verification** : $\mathcal{C}$ could detect an integrity attack when interacting with $\mathcal{S}$ for any input x and ensure the correctness of $y = f(x)$. **Function Privacy**: If $f(\cdot)$ belongs to $\mathcal{S}$, $\mathcal{C}$ cannot learn more about $f(\cdot)$ than what is revealed by $y = f(x)$. Similar to prior works [19], [22], we assume the availability of TEEs (e.g., Intel SGX) that offer hardware-based data privacy, integrity, and function privacy protection for execution inside an enclave. Our methods aims to ensure these security features for computations outside TEEs. Note that recently Intel SGX has been the subject of side-channel attacks [29], [30], [31], however, most of these issues are being studied with various mitigation techniques proposed [32], [33], [34]. These attacks are not in the scope of our work.

III. Background and Related Work

A. Transformers

A basic Transformer consists of an embedding layer and consecutive transformer layers. Every transformer layer is a composition of a multi-head self-attention (MSA) module, a feed-forward (FFN) module, two normalization modules and residual connections. The input data is split into patches, which are then transformed into a token sequence via the embedding layer. The input token sequence can be uniformly denoted as $X_e \in \mathbb{R}^{N \times D}$, where N is the number of tokens and D is the embedding dimension. We describe the main computation blocks in Transformers below.

Additive Linear Operations. The additive linear operations in Transformers are mainly the linear layers, where the output features are computed by multiplying the input features with weight matrices. For example, in the Attention module, given the input tokens $X_e \in \mathbb{R}^{N \times D}$, the output $Q, K, V \in \mathbb{R}^{N \times d_h}$ are computed by multiplying input X_e with three weight matrices $W_q, W_k, W_v \in \mathbb{R}^{D \times d_h}$, where d_h is the head dimension. Similarly, in the Feed Forward module, the input tokens $X_e \in \mathbb{R}^{N \times D}$ are multiplied by two weight matrices $W_1, W_2 \in \mathbb{R}^{D' \times D}$:

$$FeedForward(X_e) = Act(X_e \cdot W_1^T + b_1) \cdot W_2 + b_2 \quad (1)$$

These additive linear operations, such as $(X_e \cdot W_q)$ and $(X_e \cdot W_1^T)$, can be securely outsourced via existing techniques [19], [22]. The additions with the bias matrices b_1, b_2 incur only marginal computation overhead in practice.

Multiplicative Linear Operations. There are massive multiplicative linear operations in Transformers which cannot be outsourced via prior methods. The primary multiplicative linear operations are computing the attention map and attention output in the Attention module:

$$Attention(Q, K, V) = SoftMax(QK^T/\sqrt{d_h})V \quad (2)$$

The multiplicative operations (e.g., $Q \cdot K^T$) are fundamentally different from the additive linear operations (e.g., $Q = X_e \cdot W_q$). This is because in the multiplicative operations, neither operand is a constant matrix. As a result, the multiplicative operations cannot be securely outsourced via existing techniques. We refer to the matrix multiplication between Q and K, and between the attention map and V as attention matrix multiplication (AttnMult).

Non-linear Operations. Apart from the linear operations, Transformers consist of numerous non-linear operations such as the `SoftMax` function in the Attention module in Equation 2 and the Activation function `Act` in Equation 1. These non-linear operations lead to considerable computation overhead during inference. For example, the `SoftMax` is applied to $(QK^T/\sqrt{d_h}) \in \mathbb{R}^{N \times N}$ to compute the attention map. It has a complexity of $O(N^2)$, i.e., quadratic to the input size.

We highlight that the multiplicative linear operations such as $Q \cdot K^T$ and non-linear operations such as `SoftMax` are computed independently across multiple attention heads. For MSA with H heads, the multi-head attention is computed as:

$$MSA(Q, K, V) = Concat(head_1, ..., head_H)W_O \quad (3)$$

where $Concat(\cdot)$ is the concatenation operation,

$$head_i = Attention(XW_q^i, XW_k^i, XW_v^i) \quad (4)$$

and $W_O \in \mathbb{R}^{Hd_h \times D}$ is a weight matrix to map features in all heads to the output dimension. The MSA is the key mechanism in the Transformers and also the performance bottleneck. However, existing works cannot securely outsource the heavy computation in the multiplicative linear operations and non-linear operations within the MSA module.

Normalization. Normalization modules normalize the inputs of MSA and FFN. Given the input tokens $X_e \in \mathbb{R}^{N \times D}$, every value x_i in X_e is normalized to y_i by:

$$y_i = \gamma \cdot ((x_i - \mu)/\sqrt{\sigma^2 + \epsilon}) + \beta \quad (5)$$

where μ is the mean value, σ is the standard variance, γ is the scaling factor and ϵ, β are offsets. μ and σ are computed differently according to the specific normalization method. Due to their element-wise nature of these operations, it is practical to implement them within TEEs.

B. Trusted Execution Environments (TEEs)

TEEs like Intel SGX [16] provide a secure environment where data confidentiality and, in some cases, computation integrity are ensured by hardware. Intel SGX specifically safeguards the confidentiality and integrity by isolating data and code within an enclave, shielded from external elements including the operating system, hypervisor, and hardware devices on the system bus. This isolation involves a dedicated memory region, the Processor Reserved Memory (PRM), managed by SGX-enabled CPUs. Here, the Enclave Page Cache (EPC) stores enclave data and code in 4 KB pages, accessible only through specific CPU instructions. This setup prevents unauthorized access to the EPC, maintaining a secure environment for sensitive computations. SGX also supports remote attestation, allowing remote verification of an enclave's integrity through cryptographic proofs. These proofs involve hashing and signing the enclave's contents, verified by Intel's service. This feature has motivated research into running deep learning models within TEEs for security [18]. Further studies have investigated outsourcing additive linear operations to untrusted accelerators to enhance efficiency without compromising security [22], [19]. Our work expands on this by also outsourcing multiplicative linear operations and non-linear operations, enabling efficient, confidential and verifiable large-scale Transformer inference.

Legacy GPUs and GPU-based TEEs. While some emerging GPUs, such as the NVIDIA Hopper [35], have begun to support TEE environments, many other emerging GPUs and legacy GPUs (such as GTX series and A100) currently deployed in existing data centers remain in use and are likely to persist for years to come. As a result, CPU-based TEEs are still essential, and our proposed methods can continue to benefit these GPUs effectively. In addition, while GPU TEE can potentially enable native trusted GPU-based model inference, several key issues tamper its realistic adoption. Firstly, only a selected line of GPUs has the feature of TEE, which also needs to be paired with certain CPU TEEs to function properly. Such a configuration is not widely available for real-world deployment. Secondly, many existing large-scale production system are equipped with non-TEE GPUs that still have high performance, upgrading them with the TEE-enabled GPU leads to cost ineffectiveness and sustainability issues. Thirdly, with the growing heterogeneity of hardware accelerators (e.g., GPUs, TPUs an FPGAs), a secure computation scheme that relies on each hardware device to support TEE for workload outsourcing is impractical due to the potential compatibility issues among multiple vendors and the complexity of cross-device TEE protocol designs. Therefore, designing a secure outsourcing scheme that only utilizes the CPU TEE as the root of trust and can take advantage of the tremendous performance speedup from executions in the untrusted accelerators is imperative.

C. Secret Sharing for Data Confidentiality

Secret Sharing [36], [37], [38] is a cryptographic primitive that allows multiple parties to compute a function over their

inputs while keeping them private. All our algorithms are built on a two-party secret sharing over the field $\mathbb{F}_p$, where p is a prime number indicating field size. In a two-party secret sharing, a secret x is split into two shares by random sampling $\langle x\rangle_0, \langle x\rangle_1 \in \mathbb{F}_p$, such that $x = \langle x\rangle_0 + \langle x\rangle_1 \mod \mathbb{F}_p$. Secret sharing offers a strong security guarantee that, given a share $\langle x\rangle_0$ or $\langle x\rangle_1$, the value of the original x is hidden, i.e., either party can reconstruct the value of x with negligible possibility [36]. In the setting of TEE-based confidential inference, the value x can be split by a randomness $r \in \mathbb{F}_p$ chosen by the TEEs, such that the two shares are $\langle x\rangle_0 = r$ and $\langle x\rangle_1 = x - r$, respectively. Prior works [19], [20] employ secret sharing to provide privacy guarantees when outsourcing additive linear operation with constant weights w. Yet, existing outsourcing schemes cannot be extended to multiplicative operations where both operands are variables, such as Q and K, as it is impossible to precompute multiplication between r and either Q or K.

D. Computation Verification for Integrity

The verification algorithm enables a client to assert the correctness of computations performed by a server. Within the landscape of TEEs, where computations are outsourced to high-performance untrusted devices such as GPU, ensuring the integrity of these operations is paramount. Soter [21] introduces a "fingerprint" matrix method for integrity checks by the TEEs, which, however, may be vulnerable to targeted attacks. Additionally, recent research [39] suggests a sampling-based verification by the TEEs to compare against GPU outputs, facing limitations in detecting selective manipulations without extensive sampling. Freivalds' algorithm [25], referenced in [19], [20], [22], provides an efficient mechanism for verifying matrix multiplications of the form $AB = C$. The algorithm commences by generating a random vector r, followed by the TEEs computing the products $B \cdot r$ and $C \cdot r$. The next step involves multiplying A with $B \cdot r$, and comparing this outcome to $C \cdot r$. A discrepancy between these products indicates a failure of AB to equal C, whereas a match suggests a probable equality between AB and C. Employing this method, the TEEs are able to perform a verification of $\mathcal{O}(n^3)$ matrix multiplication complexity using a more efficient $\mathcal{O}(n^2)$ vector-matrix multiplication operation, thereby enhancing the verification efficiency within the TEEs. However, Freivalds' algorithm cannot be used to verify non-linear functions like `SoftMax`. In contrast, our proposed *U-Verify* method is capable of supporting such verification.

E. Related Work

In this subsection, we compare TwinShield and existing research. The first research direction, denoted by *TEE-only*, focuses on executing all computations within TEEs [40], [41]. An example is TensorSCONE [40], which conducts all inference processes inside a TEE enclave to ensure the confidentiality of both the model and data, along with inference integrity. While this approach guarantees security within the enclave, it is less efficient than performing computations in untrusted accelerators outside the TEEs. The second research trajectory, represented by *additive outsource*, aims to safeguard data confidentiality and inference integrity without necessarily protecting model confidentiality, assuming that the model provider and the cloud server are the same entity. Therefore, there's no need for model confidentiality. This approach, such as Slalom [19] and DarKnight [22], allows for the use of additive confidentiality-preserving and verification techniques to offload certain computations to untrusted hardware. Unlike cloud services that do not require model confidentiality, recent efforts like MLCapsule [18], Soter [21], ShadowNet [20] and others [42], [43], [44], [45] prioritize model privacy over user input confidentiality in on-device settings, indicating a shift in focus depending on the deployment environment. The fourth strand of research [11], [39] explores enhancing TEEs security through cryptographic methods, such as Fully Homomorphic Encryption (FHE) [46], [47], to mitigate risks like model theft and side-channel attacks. These enhancements are considered complementary to our approach. Tempo [48], which can provide protection for both model and input confidentiality on model training, lacks the established theoretical security foundations seen in Slalom [19] and ShadowNet [20]. Our work, TwinShield , aligns with the second research line but goes beyond their capabilities by facilitating the outsourcing of complex operations like multiplicative attention operations and the non-linear `SoftMax` function.

IV. Motivation

We categorize linear operations into two types: *additive* and *multiplicative*. Additive linear operations, involve one *variable* matrix and one *constant* matrix. In contrast, multiplicative linear operations involve operands that are both runtime variables. Prior works show the feasibility of securely outsourcing additive operations to untrusted accelerators via secret sharing and Freivalds' algorithm, which works well for the traditional convolutional neural networks (CNNs) since the convolution of constant pre-trained kernels and variable inputs belongs to the additive operations. We adapt this method to outsource the additive operations in the Transformer, such as $Q = X \cdot W_Q$. This method involves precomputing $R \cdot W_Q$, outsourcing the operation $(X - R) \cdot W_Q$, and an addition to obtain Q within TEEs. However, this technique does not apply to multiplicative linear operations involving two-variable matrices, for example, $QK^T = (XW_Q) \cdot (XW_K)^T$ in attention multiplication (AttnMult.), which necessitates execution within the TEEs. Additionally, the `SoftMax` function, essential for creating the attention map from QK^T, is inherently non-linear and has not been successfully outsourced by current methods, requiring it to be processed inside the TEEs. Although integrity for linear operations can be confirmed using Freivalds algorithm [25], verifying the integrity of non-linear functions such as `SoftMax` presents an ongoing challenge.

After relocating the additive linear layers to a GPU, the remaining in-TEE processing of *AttnMult.* and the `SoftMax` consumes over 60% of the total execution time. This underscores the necessity of externalizing both the multiplicative

AttnMult. and `SoftMax` computations to achieve greater efficiency. By outsourcing the multiplicative linear operations, we could potentially halve the execution time required by the TEEs. Moreover, offloading the `SoftMax` could further reduce latency, with potential savings of up to 83%. While this approach marginally increases the verification workload within the TEEs, the substantial gains in efficiency from outsourcing these operations justify the effort. This evidence motivates the pursuit of secure techniques for offloading attention matrix multiplication and `SoftMax` computations, with a focus on maintaining computational integrity, particularly for the inherently complex `SoftMax` operation.

V. TwinShield Design

Overview.

We first adapt prior work to securely outsource additive linear operations in Transformer. Then we propose ❶ *OutAttnMult* in Section V-A to securely outsource multiplicative attention operations. With all linear computation outsourced via *OutAttnMult*, the non-linear `SoftMax` function becomes the main bottleneck. To this end, we further propose ❷ *OutSoftMax* in Section V-B, to outsource the `SoftMax`. We note that the `SoftMax` involves a significant number of exponentiations, alongside few additions and divisions. Our strategic approach focuses on outsourcing the expensive exponential computations while retaining the addition and division operations within the TEEs. Specifically, this is achieved by outsourcing e^{x+r} to accelerators and recovering e^x by dividing the precomputed e^r. Also, we introduce ❸ *U-Verify* in Section V-C to ensure the integrity of outsourced computations, especially the verification of non-linear function integrity.

Our experiments reveal that, within the TEEs, normalization and activation functions are relatively lightweight, accounting for less than 5% of total execution time. Conversely, attention matrix multiplication and `SoftMax` are identified as primary bottlenecks, consuming approximately 55% and 35% of execution time, respectively. By outsourcing these bottlenecks, we can significantly enhance overall efficiency, enabling more efficient and secure Transformer inference.

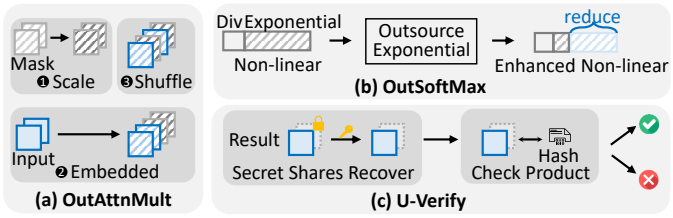

Fig. 1. (a) *OutAttnMult*'s operations within TEEs. (b) *OutSoftMax* outsources time-consuming exponential calculations in the `SoftMax` to enhance efficiency. (c) *U-Verify* is performed after recovering the outsourced computation to verify the integrity of the computation by checking the product.

The starting point is to securely outsource the additive linear operations. We adapt method from prior work [19] and integrate our proposed *U-Verify* for more efficient verification. Initially, the input X is masked with a random matrix R within the TEEs and sent to accelerators to compute $(X+R)\cdot W$. The TEEs then use the precomputed RW to recover the desired XW by subtracting RW from the accelerators' output. Verification of outsourced additive operations can be performed using either the traditional Freivalds' algorithm or our proposed *U-Verify*, as detailed in Section V-C. By default, we employ *U-Verify*, which, as shown in Figure 8, provides significant efficiency improvements over the Freivalds' algorithm.

A. Outsource Multiplicative Attention Operation: OutAttnMult

Unlike additive linear operations, multiplicative linear operations involve two variable operands. This variability prevents TEEs from precomputing the product of either operand with the predefined random mask, as they lack prior knowledge about these operands. Specifically, consider the attention multiplication $Q \cdot K^T$: TEEs (in Secure World) mask Q with R_Q and K^T with R_K^T, then outsource $(Q + R_Q) \cdot (K^T + R_K^T)$ to the accelerators (in Normal World), yielding $QK^T + R_QK^T + QR_K^T + R_QR_K^T$. To recover the target result QK^T, the TEEs must subtract the additional terms. Among these, only $R_QR_K^T$ can be precomputed as it does not depend on the variable matrices Q and K^T, unlike others which cannot be precomputed due to their dependency on Q or K^T.

We notice that the un-precomputable terms both QR_K^T and R_QK^T involve one predetermined mask and one unknown variable operand, allowing their outsourcing via the scheme for additive linear operation. For instance, TEEs can outsource $(Q + R_Q) \cdot R_K^T$ and subsequently obtain QR_K^T by subtracting the precomputed $R_QR_K^T$. However, this naïve approach presents a critical security risk by exposing R_K^T. This exposure allows the adversary in the normal world to potentially recover K^T using $K^T + R_K^T$ obtained from the initial outsourcing round.

To prevent this risk, we propose a strategy that enhances security by using a scaled version bR_K^T rather than R_K^T. More importantly, bR_K^T is not transmitted directly to the accelerator but is integrated into the matrix $K^T + R_K^T$ through a column-wise permutation. This approach serves two primary security functions: 1) it conceals the distinction between $K^T + R_K^T$ and bR_K^T, thwarting attackers from identifying them, and 2) it facilitates the simultaneous computation of $(Q + R_Q) \cdot bR_K^T$ along with $(Q+R_Q)\cdot(K^T+R_K^T)$, thereby obviating the need for an additional round of outsourcing. Subsequently, the TEEs can retrieve QR_K^T from $(Q + R_Q) \cdot R_K^T$ by applying scalar multiplication with $1/b$ and subtracting $R_QR_K^T$. Although we focus here on K^T, the processing of Q employs a similar principle.

The details of *OutAttnMult* are in Figure 2. Given input matrices $Q \in \mathbb{F}^{m\times n}$ and $K^T \in \mathbb{F}^{n\times p}$ in a finite filed $\mathbb{F}$, *OutAttnMult* is divided into offline phase and online phase.

Offline Preprocessing. Initially, TEEs (in Secure World) generate two random matrices $R_Q \in \mathbb{F}^{m\times n}$ and $R_K^T \in \mathbb{F}^{n\times p}$. It then precomputes aR_Q and bR_K^T by two scalar multiplications.

Embedded Additive Outsource. In this stage, TEEs first obfuscate Q and K^T to $Q + R_Q$ and $K^T + R_K^T$, respectively.

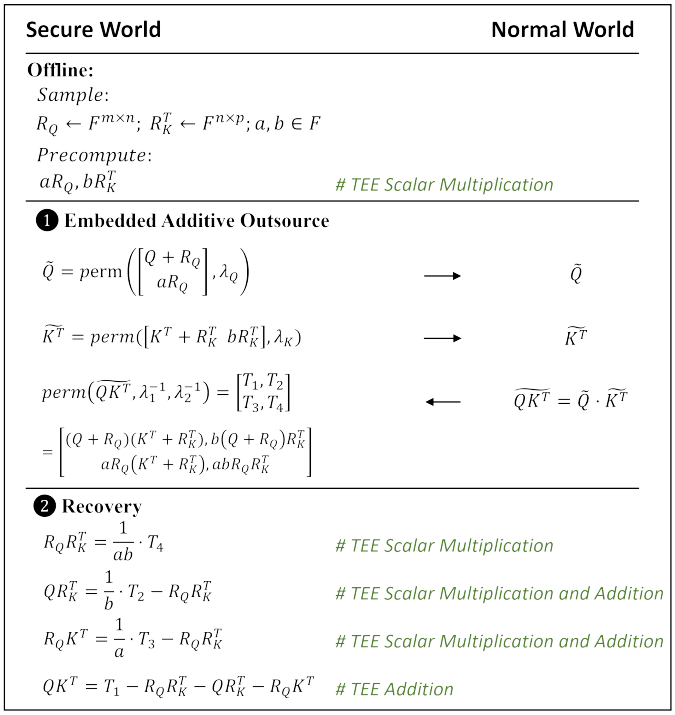

Fig. 2. Illustration of Outsource Multiplicative attention operation: *OutAttnMult*

These matrices are then embedded into $\widetilde{Q}$ and $\widetilde{K^T}$ through strategic permutations. Specifically, $\widetilde{Q}$ is crafted by vertically stacking aR_Q beneath $Q + R_Q$ and applying a row-wise permutation,

$$\widetilde{Q} = perm\left(\begin{bmatrix} Q+R_Q \\ aR_Q \end{bmatrix}, \lambda_1\right) \tag{6}$$

Similarly, $\widetilde{K^T}$ is constructed by horizontally concatenating $K^T + R_K^T$ with bR_K^T and applying a column-wise permutation,

$$\widetilde{K^T} = perm\left(\begin{bmatrix} K^T + R_K^T & bR_K^T \end{bmatrix}, \lambda_2\right) \tag{7}$$

$perm(\cdot, \lambda)$ here indicates matrix permutation with permutation indices λ, so that the adversary in normal world cannot distinguish R_Q or R_K^T from the blinded matrics.

These blinded matrices are then outsourced to the accelerator (in Normal World) for multiplication. After recovering the received results with the permutation indices, TEEs get:

$$perm(\widetilde{QK^T}, \lambda_1^{-1}, \lambda_2^{-1}) = \begin{bmatrix} (Q+R_Q)(K^T+R_K^T) & a(Q+R_Q)R_K^T \\ bR_Q(K^T+R_K^T) & abR_QR_K^T \end{bmatrix} \tag{8}$$

Figure 1 (a) intuitively shows this masked input processing.

Recover. As detailed in Figure 2, the TEEs start with applying a scalar multiplication to $abR_QR_K^T$ to obtain $R_QR_K^T$. The TEEs then retrieve QR_K^T and R_QK^T by performing scalar multiplications on $a(Q+R_Q)R_K^T$ and $bR_Q(K^T+R_K^T)$, respectively, and subsequently subtracting $R_QR_K^T$ from each. The final recovery of QK^T is achieved by strategically subtracting these terms.

Complexity Analysis. In vanilla secure matrix multiplication within TEEs, computing QK^T for matrices $Q \in \mathbb{F}^{m\times n}$ and $K^T \in \mathbb{F}^{n\times p}$ requires $\mathcal{O}(mnp)$ multiplications to be performed in resource-constrained TEEs. In contrast, *OutAttnMult* significantly reduces this burden by offloading the bulk of computation to the accelerators.

In the offline phase, TEEs perform two scalar multiplications with a complexity of $O(mn + np)$ for aR_Q and bR_K.

At the Embedded Additive Outsource stage, TEEs execute two permutations and two additions to prepare $\widetilde{Q}$ and $\widetilde{K^T}$. The GPU then handles the computationally intensive matrix multiplication $\widetilde{Q} \cdot \widetilde{K^T}$, with a complexity of $\mathcal{O}(mnp)$, given that $\widetilde{Q} \in \mathbb{F}^{2m\times n}$ and $\widetilde{K^T} \in \mathbb{F}^{n\times 2p}$.

Finally, in the recovery stage, the TEEs perform three scalar multiplications with $O(mn + np)$ and five additions to recover the desired QK^T. Overall, *OutAttnMult* shifts the computational load from $\mathcal{O}(mnp)$ multiplications within the TEEs to $\mathcal{O}(mnp)$ multiplications on the accelerator, alongside scalar multiplications and less costly permutation, and addition operations within the TEEs.

Security Analysis. In the outsourcing protocol in Figure 2, data within the TEEs (the Secure World) is protected, while data processed in accelerators (the Normal World) is exposed to potential attackers. Our goal is to prevent the attackers in the normal world from deducing the original Q or K^T. To achieve this, the TEEs construct $\widetilde{Q}$ and $\widetilde{K^T}$ via Equations 6 and 7. Taking $\widetilde{Q}$ as an example, the TEEs create a secret share by adding R_Q to Q and subsequently permutes $Q + R_Q$ together with aR_Q using private permutation indices λ_1. Since $Q+R_Q$ is equivalent to applying a one-time pad [49], its distribution is indistinguishable from aR_Q in the view of attackers [43]. The security level is quantified as $\log(d \cdot (2m)!)$, where $2m$ represents the total rows in $\widetilde{Q}$ and d denotes the finite field size. This security level estimates the probabilities for an attacker to accurately discern $Q + R_Q$ and aR_Q from $\widetilde{Q}$ and to correctly identify scalar a. In Transformers, Q typically has a large dimension (e.g., 128 for BERT), an 8-bit scalar would provide a security level of approximately 13,471 bits. Additionally, by expanding R_Q with random values and assigning varying scalars to different rows (columns for R_K^T), the TEEs can tailor the security level to meet specific requirements and matrix sizes, further enhancing protection. Detailed methodologies and additional insights are presented in Appendix A.

B. Outsource Non-linear `SoftMax`: OutSoftMax

Prior outsourcing methods for CNN-based models typically offload linear layers to accelerators, while keeping non-linear `ReLU` within TEEs due to their relatively simpler computations and the difficulties of non-linear outsourcing. However, in the context of Transformer models, the `SoftMax` within attention layers poses a substantial computational bottleneck, accounting for about 64% of the total processing time after linear operations have been outsourced from TEEs for sequences of length 512. The complexity of the `SoftMax` operation increases quadratically with the input length, which means

its computational burden becomes even more pronounced. Specifically, within the SoftMax process, the exponentiation operation alone is responsible for 92.9% of the SoftMax inference time when executed within the TEEs.

The rationale for outsourcing the SoftMax function stems from its reliance on extensive exponential calculations, which interestingly exhibit a property akin to linear operations. Specifically, linear layers are amenable to outsourcing due to the distributive property of matrix addition over multiplication, facilitating the computation of $(X + R) \cdot W$ as $XW + RW$. In contrast, non-linear layers typically do not share this trait. Yet, the exponential function crucial to the SoftMax displays a similar linear-like property: $e^{X+R} = e^X \cdot e^R$. By precalculating e^R during the offline phase, the TEEs can securely outsource the exponential computation e^{X+R}, capitalizing on this linear-like behavior. The accelerator then processes e^{X+R} and sends it back to the TEEs, where e^X is derived by dividing e^{X+R} by the precalculated e^R on an element-wise basis. This method effectively transforms an exponential operation into a multiplication, substantially easing the computational load as shown in Figure 1 (b). This is especially advantageous given that exponentiation is significantly more resource-intensive compared to basic arithmetic operations within the TEEs.

Our *OutSoftMax* algorithm is depicted in Figure 3 and comprises one offline stage along with two online stages: Outsource Masked e^X and Division in the TEEs. For an input vector $X \in \mathbb{F}^n$, the process is as described below.

Offline Preprocessing. During the offline stage, for each element x_i of the input X, the TEEs in the secure world sample a corresponding random value r_i from the field $\mathbb{F}$ and computes:

$$e^{r_i}, \quad \forall i \in \{1, \ldots, n\} \tag{9}$$

Outsource Masked e^X. TEEs mask the input vector by computing:

$$x_i' = x_i - r_i, \quad \forall i \in \{1, \ldots, n\} \tag{10}$$

Then send x_i' to the accelerator in normal world, which returns:

$$e^{x_i'}, \quad \forall i \in \{1, \ldots, n\} \tag{11}$$

Upon receving the results, the TEEs restore e^{x_i} by multiplying the accelerator's output with the precomputed exponentials:

$$e^{x_i} = e^{x_i'} \cdot e^{r_i}, \quad \forall i \in \{1, \ldots, n\} \tag{12}$$

Division in the TEEs. TEEs firstly compute the normalization scalar by:

$$s = \sum_{i=1}^{n} e^{x_i} \tag{13}$$

and subsequently, the SoftMax scores are computed by divisions:

$$y_i = \frac{e^{x_i}}{s}, \quad \forall i \in \{1, \ldots, n\} \tag{14}$$

which completes the secure SoftMax outsourcing.

By leveraging the outsourced computation for the most resource-demanding operation, i.e., exponentiation, the *OutSoftMax* efficiently computes SoftMax scores within the TEE's constraints.

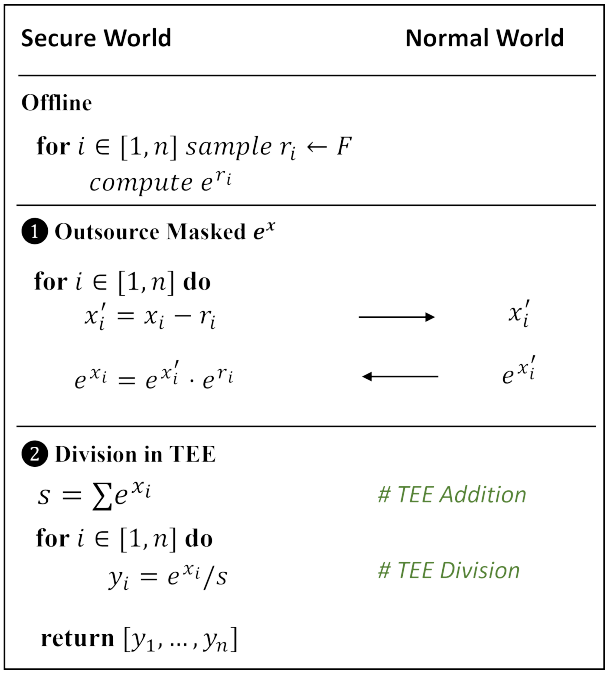

Fig. 3. Outsourcing non-linear SoftMax: *OutSoftMax*

Complexity Analysis. In the offline phase, Outsource Masked e^X, the TEEs prepare for the SoftMax operation by sampling and precomputing the exponentiations of n random values, one for each element of the input vector. During the first online phase, the TEEs mask each input element by performing n subtractions. The masked values are then outsourced to the accelerator for further processing, which undertakes n exponentiation operations. In the online phase, i.e., Division in the TEEs, the TEEs complete the SoftMax computation by executing n multiplications to combine the accelerator's output $e^{x_i'}$ with precomputed values e^{r_i}, followed by n additions to sum the exponential terms and n multiplications to calculate the SoftMax probabilities. So our *OutSoftMax* convert n exponentiation operations to convert $2n$ multiplication operations and n addition operations in the TEEs. By relocating the exponentiation tasks to the accelerators, the *OutSoftMax* algorithm relieves the TEEs of the most computationally demanding aspect of the SoftMax function. Considering that exponentiation can be an order of magnitude more time-consuming than simpler arithmetic operations in the TEEs setting, outsourcing these n operations significantly enhances the efficiency of secure inference processes. The strategic offloading of these tasks ensures that the performance bottleneck within the TEEs is mitigated.

Security Analysis. In our *OutSoftMax* protocol, only the transformed vector $X' = [x_1 - r_1, ..., x_n - r_n]$ is exposed

to the normal world. Due to the additive secret sharing [36], [37], $X' = X - R$ can be viewed as one of the secret shares of original X. Without the other share $R = [r_1, ..., r_n]$, the attacker cannot reconstruct the original X from X'.

C. U-Verify

Verification of *OutSoftMax*. Prior matrix multiplication outsourcing schemes use Freivalds' algorithm [25] to verify whether the the product of two input matrices $A \cdot B$ equals to the output matrix C. It achieves this by multiplying both B and C by a random vector r, then checking if A times the result of Br equals Cr. In this case, the TEEs can use $\mathcal{O}(mn+np)$ multiplications to verify an $\mathcal{O}(mnp)$ matrix multiplication, where (m, n) is the size of matrix A and (n, p) is the size of matrix B. However, Freivalds' algorithm is unsuitable for verifying element-wise operations like exponentiation since it relies on matrix-specific properties. In the `SoftMax` computation, exponentiation is applied individually to each element, lacking the associative and distributive properties that Freivalds' algorithm depends on, thus necessitating a different approach for verification. Our insight stems from recognizing the unique *linear-like feature* of the exponential function where $e^{a_1x_1+a_2x_2} = (e^{x_1})^{a_1} \cdot (e^{x_2})^{a_2}$.

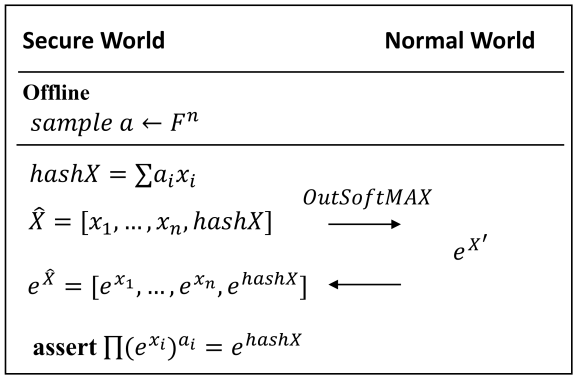

Fig. 4. *U-Verify* on *OutSoftMax*.

Figure 4 illustrates the procedure of our verification method for *OutSoftMax*. Consider an input vector $X = [x_1, ..., x_n]$. The TEEs begin with generating a secret random vector $a = [a_1, ..., a_n]$, and computing the hash of X, denoted as $hashX$, using the equation: $hashX = \sum_{i=1}^{n} a_i x_i$. This hash is then randomly inserted into X to form $\hat{X}$, and then, TEEs utilize *OutSoftMax* to send accelerators the secret share, $X' = \hat{X} - R = [x_1 - r_1, ..., hashX - r_{hash}, ..., x_n - r_n] = [x'_1, ..., hashX', ..., x'_n]$ to compute the exponential of each element, resulting in: $e^{X'} = [e^{x'_1}, ..., e^{hashX'}, ..., e^{x'_n}]$. Upon receiving $e^{X'}$ from accelerators, the TEEs first recover e^X according to Equation 12 and then verifies the correctness of the computation by checking the following equality:

$$\prod_{i=1}^{n} (e^{x_i})^{a_i} = e^{hashX} \tag{15}$$

As depicted in Figure 1 (c), since the hash is integrated prior to the outsourcing mask, integrity verification is performed after the results are recovered.

Complexity Analysis. In the `SoftMax` verification process, the TEEs first generate random positive coefficients $a = [a_1, ..., a_n]$ and computes the $hashX = \sum a_i x_i$, involving n multiplications and $n-1$ additions. This hash, alongside X, forms an augmented vector $\hat{X}$, which is then outsourced by *OutSoftMax* to the accelerator to compute $e^{X'}$. The TEEs subsequently verify the computation by comparing $\prod (e^{x_i})^{a_i}$ with e^{hashX}, ensuring the correctness of the `SoftMax` operation. This verification requires up to $n(a_{\max}-1)+n$ multiplications, depending on the largest a_i value. If the TEEs constrain the values of a below 3, the required multiplications are capped at $3n$, making this verification process significantly more efficient than performing n exponentiations directly within the TEEs for `SoftMax` computation.

Security Analysis. For a successful adversarial scenario, where the attackers seek to alter the e^{x_i} element in the correct output e^X, they would need to modify the $hashX$ according to the coefficient a_i to bypass the verification. However, without knowledge of both coefficients a_i and the location of $hashX$ within $\hat{X}$, the attack success rate is limited to $\frac{1}{n \cdot 2^d}$, where n is the size of e^X, and 2^d represents the space of possible a_i. Formally, the security level is expressed as $\log(n \cdot 2^d)$.

More importantly, since *OutSoftMax* is applied to the hashed vector $\hat{X}$, which is blinded with a random mask, the security of *U-Verify* is further strengthened by *OutSoftMax*. In other words, to launch an attack, the attackers must first brute-force the random masks used in *OutSoftMax* to recover $\hat{X}$ from X'. Thus, the overall security level is raised to $\log(n \cdot 2^d \cdot d^n)$, where $\frac{1}{d^n}$ represents the probability that attackers correctly identify all masks used in *OutSoftMax*.

Furthermore, traditional replay attacks are thwarted by the one-time use of scalars in *U-Verify* and masks in *OutSoftMax*. More details can be found in Appendix A.

Verification of *OutAttnMult*. Figure 5 outlines the procedure for verifying *OutAttnMult* operations. In the offline phase, the TEEs commence by sampling a secret random vector h_Q from the finite field $\mathbb{F}^n$ to compute the hash $hashQ$ as follows: $hashQ = h_Q \cdot Q$. Next, during the online phase, the TEEs construct an augmented matrix by appending $hashQ$ to matrix Q, and then utilizes the accelerator's computation of the product with K^T using *OutAttnMult*. Upon receiving the results from the accelerator:

$$\begin{bmatrix} Q \\ hashQ \end{bmatrix} \cdot K^T = \begin{bmatrix} QK^T \\ hashQ \cdot K^T \end{bmatrix} \tag{16}$$

The TEEs recover the result via *OutAttnMult*, and then verify the multiplication's correctness by ensuring that $h_Q \cdot QK^T$ equals $hashQ \cdot K^T$.

Security Analysis. A successful integrity attack requires tampering with the matrix multiplication result QK^T such that it still passes the integrity check $h_Q \cdot QK^T = hash_Q \cdot K^T$. To bypass this check, an attacker would need to modify $hash_Q$ so that the altered $hash_Q \cdot K^T$ remains consistent with $h_Q \cdot QK^T$.

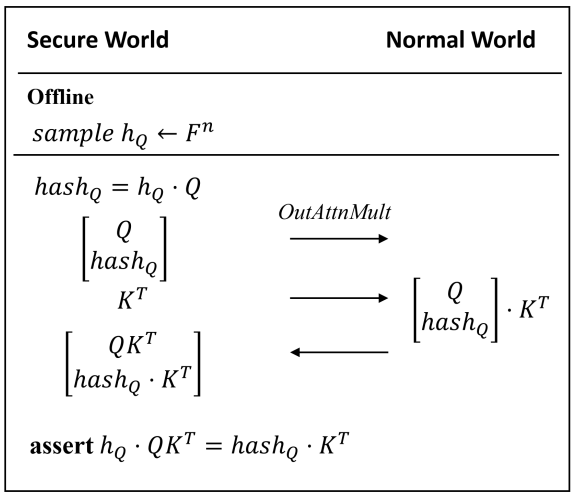

Fig. 5. *U-Verify* on *OutAttnMult*.

However, multiple barriers prevent such an attack. First, the permutation procedure in *OutAttnMult* obscures $hash_Q$ within a mixture of $hash_Q$ and Q, making it impossible for the attacker to identify $hash_Q$. Second, the additive masks applied during *OutAttnMult* render any malicious multiplication ineffective. Most critically, the random vector h_Q is available only within the TEEs, so without knowing h_Q, it is impossible to maintain the equality $h_Q \cdot QK^T = hash_Q \cdot K^T$ by modifying QK^T, $hash_Q$, or K^T. Simply multiplying all terms by the same scalar also fails because the mask introduced by *OutAttnMult* disrupts this multiplicative relationship. More discussions can be found in Appendix B.

Complexity Analysis. The TEEs compute the hash $hashA$ using $n \times m$ multiplications and few additions. The accelerator then performs the bulk of computation by multiplying the augmented matrix with B, requiring a total of $(m+1) \times n \times p$ multiplications for both the matrix product and the hash. Upon receiving the result, the TEEs verify the integrity with a single vector-matrix multiplication involving $m \times p$ multiplications.

VI. Experimental Methodology

In this section, we introduce the experimental methodology.

Models. In our evaluation, we selected four transformers: 1) ViT-16B [2] applies the transformer structure to computer vision, comprised of 12 layers and 16 attention heads, each with a hidden feature size of 768, tailored for image classification. 2) BERT-Base [3], a seminal model in NLP, features 12 layers and 12 attention heads, each with a hidden feature size of 768, aimed at comprehending and processing language. 3) LLaMA-7B [4] marks an advancement in language models with its 32 layers and 32 attention heads, each with an extensive hidden feature size of 4096, for intricate language tasks. 4) CLIP [50] fuses vision and language by employing 12 layers with 12 attention heads for both its text and visual encoders, each with a hidden feature size of 768, and is trained across a diverse set of images and text pairs.

Datasets. ViT-16B and CLIP are evaluated on the ImageNet [51] dataset, a benchmark for image classification, with accuracy serving as the metric for success. BERT-Base is tested against the SST-2 dataset [52], a standard for sentiment analysis in NLP, also using accuracy as the evaluative measure. For LLaMA-7B, we employ the Wiki-Text dataset [53] to gauge its language modeling capability, utilizing Perplexity as the metric, which quantifies how well a probability model predicts a sample. Accuracy measures the proportion of correct predictions over the total, reflecting classification performance, while Perplexity measures the model's certainty in its predictions, with lower values indicating better predictive performance.

System Setup and Implementation. We conducted the TwinShield implementation on a server powered by an Intel(R) Xeon(R) Gold 6342 CPU, operating at 2.8GHz, and equipped with 512GB of DRAM. This setup also included an NVIDIA A40 GPU with 48GB of VRAM. Our SGX implementation leveraged Eigen [54], a linear-algebra library also employed by TensorFlow for constructing DNN layers such as the attention module, `SoftMax`, LayerNorm, GeLU, and ReLU. The development environment included TensorFlow and Python 3, used for model quantization and inference. We sourced pre-trained models for ViT, BERT, and CLIP from Keras [55] and applied quantization techniques as described in recent studies [56], [57]. Additionally, we acquired a pre-trained, quantized 8-bit LLaMA model from HuggingFace [58].

Quantization. TwinShield adopts a quantization strategy for both inputs and model weights, drawing on the approaches of Slalom [19] and DarKnight [22]. Initially, it converts values from floating-point to fixed-point by selecting a fractional bit number, l, scaling values by 2^l, and rounding to integers. For negative values, a correction p is applied to adjust them into the field $\mathbb{Z}_p$, where prime $p = 2^{24} - 3$. Subsequent computations are outsourced to the GPUs by the TEEs, which later dequantizes the GPU's results to obtain the original values. Our experimental setup, with $l = 8$, resulted in a maximum accuracy drop of 1.9% as shown in Table VI. Accuracy for ViT and CLIP was measured on the ImageNet validation set, BERT-Base on the SST-2 dataset, and LLaMA's perplexity on the Wiki-Text dataset, experiencing at most a 1.9% accuracy decrease and a 0.21 increase in perplexity for LLaMA.

VII. Experimental Results

A. End-to-end performance

Comparison to baseline methods. In Table I, we provide a detailed evaluation of TwinShield across four transformer architectures and three distinct datasets, both with and without integrity verification. We first compare against a TEE-only baseline [18], where all inferences are executed entirely within the TEEs, demonstrating the least efficiency. For example, a single ViT inference takes 0.713 s in this setup. By outsourcing additive matrix multiplications between input X and weights W, an approach labeled as "Additive OutSrc.," we achieve a $1.5\times$ average speedup with integrity verification.

Building on this, TwinShield significantly extends the outsourcing capabilities by offloading all multiplicative linear operations and `SoftMax` computations in the attention module from TEEs to the GPU. This optimization results in a $3.6\times$ speedup over "Additive OutSrc." and an overall $5.4\times$ improvement compared to the TEE-only baseline, while maintaining both integrity and privacy. These enhancements are primarily due to TwinShield 's efficient outsourcing of computational bottlenecks, particularly multiplicative matrix multiplications and `SoftMax`. With only privacy protection (and no matrix multiplication required for verification), TwinShield achieves an even greater $6.67\times$ average speedup.

The performance gains are especially pronounced in larger models like LLaMA, where TwinShield achieves a $6.1\times$ speedup over the TEE-only baseline. This highlights the critical role of GPU-based outsourcing in mitigating the resource limitations of TEEs, particularly for expansive transformer models.

TABLE I
COMPARISON OF TwinShield AND PRIOR METHODS, I.E., TEE-ONLY AND ADDITIVE LINEAR OUTSOURCING METHODS, ON END-TO-END TIME (S).

Method		ViT-ImageNet	BERT-SST2	CLIP-ImageNet	LLaMA-WiKi
TEE-only		0.713	1.294	1.972	113.4
Additive OutSrc.	w/o verf.	0.511	0.858	1.255	72.53
	w/ verf.	0.523	0.886	1.298	75.28
TwinShield	w/o verf.	0.151	0.174	0.205	14.77
	w/ verf.	0.188	0.216	0.363	18.60

Results on long-token inputs. Table I illustrates that TwinShield achieves a $6.0\times$ speedup on BERT with an input token number of 128. The performance improvement can be further pronounced when the inputs have more tokens (long-token inputs). The Figure 6 (a) and (b) depict the speedup with and without integrity verification, respectively. The results showcase that TwinShield 's advantage over TEE-only and "Additive OutSrc." methods grows with increasing token numbers. For instance, with integrity verification, the speedup with 64 input tokens is $2.8\times$ and surges to $10.7\times$ with 256 tokens. This is attributed to TwinShield 's optimization strategy, which simplifies the TEEs' workload from $\mathcal{O}(n^3)$ to $\mathcal{O}(n^2)$ matrix multiplications and transforms $\mathcal{O}(n)$ exponential operations to multiplications. Without the integrity verification, the speedup is even more pronounced, reaching $15.2\times$ for 256 tokens, as it eliminates the need for the TEEs to perform $\mathcal{O}(n^2)$ matrix multiplications for verification. Conversely, the "Additive OutSrc". approach does not achieve such significant gains because it still relies on TEEs for the remaining matrix multiplication and `SoftMax` computations, which become bottlenecks and limit performance improvements across varying token numbers.

Results on standard CPU. To assess our outsourcing schemes' performance without the specific constraints of Intel SGX, we evaluated the benchmarks presented in Table I on the same CPU, but outside SGX enclave mode. Table II demonstrates benchmarks for BERT and ViT on a single core with either direct computation or various secure outsourcing schemes. On the BERT, by outsourcing additive matrix multiplication to the GPU (Additive OutSrc.) with integrity verification, the latency drops from 0.492 s to 0.334 s. This is further reduced to 0.131 s with TwinShield , which outsources both linear and non-linear operations, achieving a $3.8\times$ speedup. For ViT, the TwinShield can achieve $3.3\times$ and $4.1\times$ speedup with and without integrity verification.

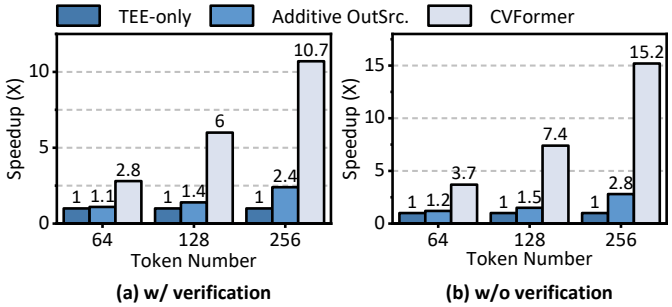

Fig. 6. TwinShield obtains higher performance speedup on long-token inputs.

TABLE II
COMPARISON OF TwinShield AND PRIOR METHODS USING BERT AND VIT ON AN UNTRUSTED CPU.

Method		BERT (s)	ViT (s)
CPU-only		0.492	0.331
Additive OutSrc.	w/o verf.	0.318	0.235
	w/ verf.	0.334	0.247
TwinShield	w/o verf.	0.095	0.081
	w/ verf.	0.131	0.102

B. Ablation Study and Benchmark

Ablation study on the effectiveness of proposed techniques. Table III evaluates the performance of our proposed methods on the BERT model with a 128-token input, comparing scenarios both with and without integrity verification. With integrity verification enabled, outsourcing additive matrix multiplications reduces the inference latency from 1.294 s to 0.886 s. Further optimization is achieved by utilizing *OutAttnMult* to outsource multiplicative attention matrix multiplications, which significantly decreases the end-to-end latency to 0.570 s. This improvement is attributed to simplifying the TEEs' workload from matrix-matrix to vector-matrix multiplications for validating GPU computations.

In addition, incorporating *OutSoftMax* to outsource the `SoftMax` operation further reduces the latency to 0.531 s. This reduction reflects the shift in the TEE's role from performing computationally intensive exponential operations to simpler multiplication tasks. By combining both outsourcing strategies, the inference latency is ultimately reduced to just

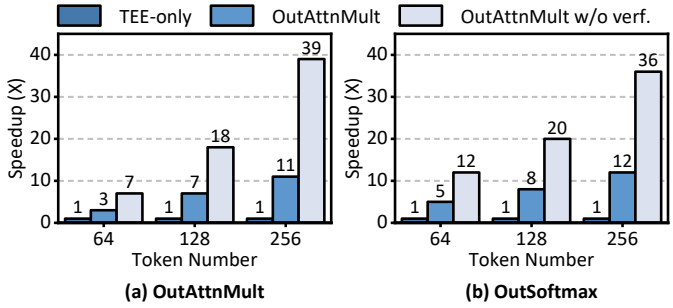

Fig. 7. Performance comparison of *OutAttnMult* (a) and *OutSoftMax* (b) over TEE-only execution.

0.216 s, achieving a substantial 83.3% reduction for a single BERT inference.

TABLE III
ABLATION STUDY OF PROPOSED TECHNIQUES. TEE-ONLY NATIVELY PROVIDES INTEGRITY GUARANTEE.

Technique	Time (s)	
	w/o verf.	w/ verf.
TEE-only	-	1.294
Additive OutSrc.	0.858	0.886
Additive OutSrc. + *OutAttnMult*	0.561	0.570
Additive OutSrc. + *OutSoftMax*	0.525	0.531
Additive OutSrc. + *OutAttnMult* + *OutSoftMax*	0.174	0.216

Performance Benchmark on *OutAttnMult* and *OutSoftMax*. To investigate the effectiveness of our proposed methods, *OutAttnMult* and *OutSoftMax*, across different matrix sizes and token counts, we conducted a performance benchmark. The results are illustrated in Figure 7. In Figure 7 (a), we observe that *OutAttnMult* secures progressively higher speedups with the growth in square matrix dimensions, from a $3\times$ enhancement with a dimension of 256 to an $11\times$ improvement with a dimension of 1024. This performance increase is attributed to *OutAttnMult*'s efficiency in reducing the complexity of matrix multiplications from $\mathcal{O}(n^3)$ to $\mathcal{O}(n^2)$ within the TEEs. The limited memory capacity of the TEEs, which exacerbates computational overhead, further accentuates the performance disparity between *OutAttnMult* and conventional TEE-only computation.

Figure 7 (b) highlights the performance improvements introduced by *OutSoftMax*. With matrix sizes expanding from 256 to 1024 dimensions, the observed speedups range from $5\times$ to $12\times$. These enhancements stem from the relative efficiency of multiplication over exponentiation, with *OutSoftMax* effectively transforming exponential operations within the TEEs into multiplications. The observed trend of increased speedups with larger matrix dimensions can be linked to the TEE's constrained memory, necessitating more frequent paging for larger matrices and thereby reaping greater benefits from our optimization strategies.

C. Analysis of Proposed Techniques

Breakdown analysis of *OutAttnMult*. In Table IV, we present a latency breakdown for each stage of the *OutAttnMult* process, as applied to a single attention matrix multiplication of the BERT model. The results reveal that the offline phase within the TEEs is small, clocking in at 0.169 ms. This is because our *OutAttnMult* also outsources $R_Q R_K^T$ to the accelerator, compared to the scheme used in Slalom [19] which requires precomputing matrix multiplications. In the online phase, the Embedded Additive Outsource requires slightly more time, at 0.152 ms, due to the necessity for matrix permutation and a combination of two subtractions and an addition. The Recovery stage which needs scalar multiplication and addition requires 0.498 ms. The Integrity Check stage, however, exhibits the highest latency within the TEEs, i.e., 0.763 ms, necessitated by two vector-matrix multiplications to confirm the GPU's computation integrity. For the GPU, the matrix multiplication tasks in Embedded Additive Outsource stages are executed rapidly, leveraging the GPU's superior performance for matrix operations and contributing only 0.209 ms.

Summing up, the total online latency for the TEEs is 1.413 ms, while the GPU contributes an additional 0.209 ms, leading to an aggregate end-to-end latency of 1.622 ms for the complete *OutAttnMult*.

TABLE IV
LATENCY BREAKDOWN OF *OutAttnMult*.

Stage	TEE (ms)	GPU (ms)
Offline	0.169	-
Embedded Additive Outsource	0.152	0.209
Recovery	0.498	-
Integrity Check	0.763	-
Online Total	1.413	0.209

Breakdown analysis of *OutSoftMax*. To investigate the latency various different stages in *OutSoftMax*, we also conduct breakdown analysis in Table V with token number of 256. The offline phase within the TEEs is the most time-intensive, at 5.815 ms, largely owing to the sampling of a random vector and computing the exponentials for each of its elements. During the Outsource Masked e^x stage, the TEEs handles element-wise multiplications of e^{x_i} and e^{r_i}, contributing to the online latency. Similarly, the Division in the TEEs phase requires element-wise divisions, further adding to the TEE's workload. The Integrity Check, crucial for ensuring the correctness of the GPU's computations, involves additional element-wise multiplications by the TEEs to validate the results against e^{hashX}. Overall, the online portion of the *OutSoftMax* process necessitates 1.190 ms from the TEEs and only 0.138 ms from the GPUs. This breakdown analysis illustrates that speedup is attributed to converting the operation within TEEs from element-wise exponential to much cheaper element-wise multiplication.

TABLE V
LATENCY BREAKDOWN OF *OutSoftMax*.

Stage	TEE (ms)	GPU (ms)
Offline	5.815	-
Outsource Masked e^x	0.327	0.138
Division in TEE	0.381	-
Integrity Check	0.482	-
Online Total	1.190	0.138

Ablation on the value of random coefficient a in *U-Verify* for *OutSoftMax*. Our analysis of *OutSoftMax* reveals the impact of the maximum value of random coefficients ($a_{\max}$) on computational efficiency. In trials using a 256×256 matrix, we observed that increasing $a_{\max}$ from 2 to 7 leads to a proportional decrease in speedup—from a maximum of $7.4\times$ down to $5.3\times$. This trend demonstrates a linear relationship between $a_{\max}$ and the speedup, which is consistent with our complexity analysis, confirming that the number of required multiplications is directly tied to $a_{\max}$. By carefully choosing an optimal $a_{\max}$, *OutSoftMax* can effectively minimize the TEE's latency by simplifying $\mathcal{O}(n)$ exponential operations to $\mathcal{O}(n)$ multiplications.

Comparison between Freivald's algorithm and *U-Verify*. As Section V-C shows, it's important to highlight that Freivalds' algorithm cannot be used for verifying non-linear operations like `softmax`, in contrast to our *U-Verify* approach. In assessing the verification efficiency on linear attention multiplication, we compared our *U-Verify* with the Freivalds' algorithm [25]. Figure 8 indicates that *U-Verify* delivers an approximate 33% reduction in latency when compared to Freivalds' method. Freivalds' algorithm typically requires three vector-matrix multiplications to verify the product of $A^{n\times n}$ and $B^{n\times n}$ equals $C^{n\times n}$, calculating Cs, Bs, and subsequently $A(Bs)$. In contrast, *U-Verify* streamlines this process to just two vector-matrix multiplications. It achieves this by embedding a hash row, $hashA$, within A and outsourcing $hashA \cdot B$ to the GPU. Consequently, the TEEs merely need to verify that $h_A \cdot C$ corresponds to $hashA \cdot B$, leading to the observed efficiency gains.

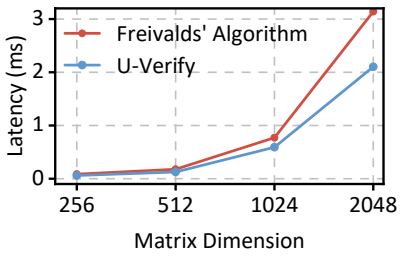

Fig. 8. Comparison between Freivalds' algorithm and *U-Verify* on linear attention multiplication. Note that Freivalds' algorithm is not applicable for non-linear operations such as `softmax`, unlike our *U-Verify* method.

Accelerator Studies. In addition to evaluating legacy GPUs, we also explored other accelerator platforms to assess the adaptability of our approach.

FPGAs. To evaluate the versatility of TwinShield with alternative accelerators, we conducted experiments using the Xilinx Alveo U280 FPGA. This FPGA features 8 GB of HBM2 memory, 32 GB of DDR memory with a bandwidth of 460 GBps, and 1,304K logic cells. It connects to the host machine via a PCIe Gen3x16 I/O interface. We utilized the Vitis BLAS Library for *OutAttnMult* operations and the Vitis AI Library for outsourcing the *OutSoftMax*. The results showed that outsourcing computations to the FPGA significantly reduced latency compared to performing all operations within a TEE. Specifically, TwinShield achieved latency reductions ranging from $1.93\times$ to $3.25\times$, demonstrating the effectiveness of our outsourcing strategy across different accelerator platforms.

TPUs. We also tested the adaptability of TwinShield on Google's TPU VM v3-8, which is equipped with 128 GB of total memory distributed across 8 chips and offers an interconnect bandwidth of 100 GBps. For implementation, we used TensorFlow 2.1. The results revealed that leveraging a TPU for outsourcing operations drastically reduced latency compared to relying exclusively on a TEE for computations. Notably, TwinShield achieved latency reductions between $7.43\times$ and $10.91\times$ when utilizing the TPU, highlighting the significant advantages of our outsourcing strategy across diverse computational platforms.

VIII. CONCLUSION

In this paper, we propose TwinShield , an innovative framework to safeguard the privacy and integrity of Transformer inference services in the cloud setting. We design secure and efficient modules, *OutAttnMult* and *OutSoftMax*, to outsource bottleneck computations in Transformers including the non-additive attention multiplication and non-linear `softmax` functions. We further propose a novel verification scheme *U-Verify* to ensure the integrity of the outsourced computation. Extensive experiments show that TwinShield offers from $4.0\times$ to $6.1\times$ performance improvement over prior works for private verifiable inference without sacrificing accuracy.

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] 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.

[3] 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.

[4] 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.

[5] L. Floridi and M. Chiriatti, "Gpt-3: Its nature, scope, limits, and consequences," *Minds and Machines*, vol. 30, pp. 681–694, 2020.

[6] M. A. Myszczynska, P. N. Ojamies, A. M. Lacoste, D. Neil, A. Saffari, R. Mead, G. M. Hautbergue, J. D. Holbrook, and L. Ferraiuolo, "Applications of machine learning to diagnosis and treatment of neurodegenerative diseases," *Nature Reviews Neurology*, vol. 16, no. 8, pp. 440–456, 2020.

[7] J. G. Richens, C. M. Lee, and S. Johri, "Improving the accuracy of medical diagnosis with causal machine learning," *Nature communications*, vol. 11, no. 1, p. 3923, 2020.

[8] J. B. Heaton, N. G. Polson, and J. H. Witte, "Deep learning for finance: deep portfolios," *Applied Stochastic Models in Business and Industry*, vol. 33, no. 1, pp. 3–12, 2017.

[9] G. McLean and K. Osei-Frimpong, "Hey alexa… examine the variables influencing the use of artificial intelligent in-home voice assistants," *Computers in Human Behavior*, vol. 99, pp. 28–37, 2019.

[10] J. Foerster, I. A. Assael, N. De Freitas, and S. Whiteson, "Learning to communicate with deep multi-agent reinforcement learning," *Advances in neural information processing systems*, vol. 29, 2016.

[11] D. Natarajan, A. Loveless, W. Dai, and R. Dreslinski, "Chex-mix: Combining homomorphic encryption with trusted execution environments for oblivious inference in the cloud," in *2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2023, pp. 73–91.

[12] Y. Zhang, J. Xue, M. Zheng, M. Xie, M. Zhang, L. Jiang, and Q. Lou, "Cipherprune: Efficient and scalable private transformer inference," in *The Thirteenth International Conference on Learning Representations*.

[13] M. Zheng, Q. Lou, and L. Jiang, "Primer: Fast private transformer inference on encrypted data," *DAC 2023*, 2023.

[14] T. Pahima, "Breakingformation: Orca security research team discovers aws cloudformation vulnerability," *Complete Cloud Security in Minutes-Orca Security*, 2022.

[15] L. Tung, "Google cloud: Here are the six'best'vulnerabilities security researchers found last year," *ZDNET, Mar*, 2021.

[16] "Intel® software guard extensions (intel® sgx) developer guide," https://www.intel.com/content/www/us/en/developer/tools/software-guard-extensions/overview.html, 2023.

[17] "Intel® software guard extensions (intel® sgx) developer guide," https://www.intel.com/content/www/us/en/developer/tools/software-guard-extensions/attestation-services.html, 2023.

[18] L. Hanzlik, Y. Zhang, K. Grosse, A. Salem, M. Augustin, M. Backes, and M. Fritz, "Mlcapsule: Guarded offline deployment of machine learning as a service," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2021, pp. 3300–3309.

[19] F. Tramer and D. Boneh, "Slalom: Fast, verifiable and private execution of neural networks in trusted hardware," in *International Conference on Learning Representations*, 2018.

[20] Z. Sun, R. Sun, C. Liu, A. R. Chowdhury, L. Lu, and S. Jha, "Shadownet: A secure and efficient on-device model inference system for convolutional neural networks," in *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2023, pp. 1596–1612.

[21] T. Shen, J. Qi, J. Jiang, X. Wang, S. Wen, X. Chen, S. Zhao, S. Wang, L. Chen, X. Luo *et al.*, "{SOTER}: Guarding black-box inference for general neural networks at the edge," in *2022 USENIX Annual Technical Conference (USENIX ATC 22)*, 2022, pp. 723–738.

[22] H. Hashemi, Y. Wang, and M. Annavaram, "Darknight: An accelerated framework for privacy and integrity preserving deep learning using trusted hardware," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 212–224.

[23] X. Deng, S. Fan, Z. Hu, Z. Tian, Z. Yang, J. Yu, D. Cao, D. Meng, R. Hou, M. Li *et al.*, "Trinity: A general purpose fhe accelerator," *Accepted by MICRO'24*, 2024.

[24] A. W. B. Yudha, J. Xue, Q. Lou, H. Zhou, and Y. Solihin, "Boostcom: Towards efficient universal fully homomorphic encryption by boosting the word-wise comparisons," in *(PACT'24) The International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2024.

[25] R. Freivalds, "Probabilistic machines can use less running time." in *IFIP congress*, vol. 839, 1977, p. 842.

[26] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[27] "Medical gpt," https://www.medicalgpt.info/, 2024.

[28] "Finance gpt," https://financegpt.uk/, 2024.

[29] J. Van Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx, "Foreshadow: Extracting the keys to the intel {SGX} kingdom with transient {Out-of-Order} execution," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018, pp. 991–1008.

[30] S. Van Schaik, A. Milburn, S. Österlund, P. Frigo, G. Maisuradze, K. Razavi, H. Bos, and C. Giuffrida, "Ridl: Rogue in-flight data load," in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 88–105.

[31] C. Canella, D. Genkin, L. Giner, D. Gruss, M. Lipp, M. Minkin, D. Moghimi, F. Piessens, M. Schwarz, B. Sunar *et al.*, "Fallout: Leaking data on meltdown-resistant cpus," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 769–784.

[32] F. Brasser, S. Capkun, A. Dmitrienko, T. Frassetto, K. Kostiainen, and A.-R. Sadeghi, "Dr. sgx: Automated and adjustable side-channel protection for sgx using data location randomization," in *Proceedings of the 35th Annual Computer Security Applications Conference*, 2019, pp. 788–800.

[33] X. Lou, T. Zhang, J. Jiang, and Y. Zhang, "A survey of microarchitectural side-channel vulnerabilities, attacks, and defenses in cryptography," *ACM Computing Surveys (CSUR)*, vol. 54, no. 6, pp. 1–37, 2021.

[34] Y. Xu, W. Cui, and M. Peinado, "Controlled-channel attacks: Deterministic side channels for untrusted operating systems," in *2015 IEEE Symposium on Security and Privacy*. IEEE, 2015, pp. 640–656.

[35] J. Choquette, "Nvidia hopper h100 gpu: Scaling performance," *IEEE Micro*, vol. 43, no. 3, pp. 9–17, 2023.

[36] R. Cramer, I. B. Damgård *et al.*, *Secure multiparty computation*. Cambridge University Press, 2015.

[37] D. Demmler, T. Schneider, and M. Zohner, "Aby-a framework for efficient mixed-protocol secure two-party computation." in *NDSS*, 2015.

[38] Q. Lou and L. Jiang, "She: A fast and accurate deep neural network for encrypted data," in *Advances in Neural Information Processing Systems (NeurIPS) 2019*, 2019, pp. 10 035–10 043.

[39] Y. Wei, X. Wang, S. Bian, W. Zhao, and Y. Jin, "The-v: Verifiable privacy-preserving neural network via trusted homomorphic execution," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 2023, pp. 1–9.

[40] R. Kunkel, D. L. Quoc, F. Gregor, S. Arnautov, P. Bhatotia, and C. Fetzer, "Tensorscone: A secure tensorflow framework using intel sgx," *arXiv preprint arXiv:1902.04413*, 2019.

[41] T. Lee, Z. Lin, S. Pushp, C. Li, Y. Liu, Y. Lee, F. Xu, C. Xu, L. Zhang, and J. Song, "Occlumency: Privacy-preserving remote deep-learning inference using sgx," in *The 25th Annual International Conference on Mobile Computing and Networking*, 2019, pp. 1–17.

[42] T. Zhou, Y. Luo, S. Ren, and X. Xu, "Nnsplitter: an active defense solution for dnn model via automated weight obfuscation," in *Proceedings of the 40th International Conference on Machine Learning*, ser. ICML'23. JMLR.org, 2023.

[43] Z. Zhang, C. Gong, Y. Cai, Y. Yuan, B. Liu, D. Li, Y. Guo, and X. Chen, "No privacy left outside: On the (in-) security of tee-shielded dnn partition for on-device ml," in *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 2023, pp. 52–52.

[44] Z. Liu, Y. Luo, S. Duan, T. Zhou, and X. Xu, "Mirrornet: A tee-friendly framework for secure on-device dnn inference," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 2023, pp. 1–9.

[45] Z. Zhang, L. K. Ng, B. Liu, Y. Cai, D. Li, Y. Guo, and X. Chen, "Teeslice: slicing dnn models for secure and efficient deployment," in *Proceedings of the 2nd ACM International Workshop on AI and Software Testing/Analysis*, 2022, pp. 1–8.

[46] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE transactions on information theory*, vol. 31, no. 4, pp. 469–472, 1985.

[47] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *International conference on the theory and applications of cryptographic techniques*. Springer, 1999, pp. 223–238.

[48] R. Xu and Z. Fang, "Tempo: Confidentiality preservation in cloud-based neural network training," *arXiv preprint arXiv:2401.11531*, 2024.

[49] M. Bellare and P. Rogaway, "Introduction to modern cryptography," *Lecture Notes*, 2001.

[50] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark *et al.*, "Learning transferable visual models from natural language supervision," in *International conference on machine learning*. PMLR, 2021, pp. 8748–8763.

[51] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[52] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts, "Recursive deep models for semantic compositionality over a

sentiment treebank," in *Proceedings of the 2013 conference on empirical methods in natural language processing*, 2013, pp. 1631–1642.
[53] S. Merity, C. Xiong, J. Bradbury, and R. Socher, "Pointer sentinel mixture models," *arXiv preprint arXiv:1609.07843*, 2016.
[54] G. Guennebaud, B. Jacob *et al.*, "Eigen v3," http://eigen.tuxfamily.org, 2010.
[55] F. Chollet *et al.*, "Keras," https://keras.io, 2015.
[56] Z. Liu, Y. Wang, K. Han, W. Zhang, S. Ma, and W. Gao, "Post-training quantization for vision transformer," *Advances in Neural Information Processing Systems*, vol. 34, pp. 28 092–28 103, 2021.
[57] S. Kim, A. Gholami, Z. Yao, M. W. Mahoney, and K. Keutzer, "I-bert: Integer-only bert quantization," in *International conference on machine learning*. PMLR, 2021, pp. 5506–5518.
[58] "Hugging face." https://huggingface.co/, 2024.
[59] J. Katz and Y. Lindell, *Introduction to modern cryptography: principles and protocols*. Chapman and hall/CRC, 2007.

APPENDIX

In this section, we will outline the method to construct the feasible set, $F(\hat{X})$ for a given transformed matrix $\hat{X}$ (the X could be Q or K^T).

We refer to the r_i in the random vectors $R = [r_1, ..., r_{t-n}]$ as mask vectors. Additionally, let $\subset_R$ represent a uniform random sampling. The idea is to back-trace and compute the set of possible of pre-images. Now the feasible set is constructed as follows:

1) Select the set of $t - n$ indices uniformly at random:
$$\Omega \subset_R [m], |\Omega| = t - n \tag{17}$$
Ω represents a possible set of indices that correspond to the mask vectors.
2) The corresponding set of mask vectors is:
$$\Phi_\Omega = \{\hat{x_i} | i \in \Omega\} \tag{18}$$
3) Let $\overline{\Phi}_\Omega = \{\hat{x_i} | i \in [n] \backslash \Omega\}$ be the set of transformed original vectors. Additionally, let $\overline{X} = [\overline{x}_1, ..., \overline{x}_n]$ where $\overline{x}_i \in \overline{\Phi}_\Omega$ and $\overline{x}_i \neq \overline{x}_j$, $i, j \in [n]$, $i \neq j$.
4) Sample a random permutation σ to recover the shuffled matrix. Thus $\overline{X}_\alpha = [\overline{x}_{\sigma(1)}, ..., \overline{x}_{\sigma(n)}]$ represents a possible transformed matrix.
5) Compute
$$\begin{gathered} \forall i \in [n] \\ \mathcal{F}^i_{\Omega,\sigma}(\hat{X}) = \{x | x = d \cdot (\overline{x}_{\sigma(i)} - \hat{x}'), d \in \mathbb{R}, \hat{x}' \subset_R \Phi_\Omega\} \end{gathered} \tag{19}$$
$\mathcal{F}^i_{\Omega,\sigma}(\hat{X})$ denotes the set of possible values for the vector x_i for the give Ω and σ.
6) Clearly, we have
$$\mathcal{F}(\hat{X}) = \bigcup_{\Omega} \bigcup_{\sigma} \mathcal{F}_{\Omega,\sigma}(\hat{X}) \tag{20}$$

Clearly, larger the value of obfuscation ratio r, greater is the size of Ω and consequently, $\mathcal{F}(\hat{X})$. Additionlly, it is evident that $\mathcal{F}(\hat{X_a}) \supset \mathcal{F}(\hat{X_b})$ where $t_a = |\hat{X_a}| > |\hat{X_b}| = t_b$ (equivalently, $t_a > t_b$).

For the transformed $\widetilde{Q}$ and $\widetilde{K^T}$, let $F(\widetilde{Q})$ and $F(\widetilde{K^T})$ represent the set of original matrixs that could have been transformed to $\widetilde{Q}$ and $\widetilde{K^T}$, i.e., the set of possible pre-images for $\widetilde{Q}$ and $\widetilde{K^T}$. We call them *feasible set* for $\widetilde{Q}$ and $\widetilde{K^T}$. The construction of *feasible set* can be found in appendix A.

Theorem 1. For a $\widetilde{QK^T}$ computation and a given view of the GPU $\text{View}_{\text{GPU}} = (\widetilde{Q}, \widetilde{K^T}, \widetilde{QK^T}, (QK^T)')$, where $(QK^T)'$ is used to operate subsequent computation in GPU, has been transformed by TEE. We have:
$$\begin{gathered} \forall (Q_i, Q_j) \in \mathcal{F}(\widetilde{Q}) \times \mathcal{F}(\widetilde{Q}) \\ \Pr[Q = Q_i | \text{View}_{\text{GPU}}] = \Pr[Q = Q_j | \text{View}_{\text{GPU}}] \end{gathered} \tag{21}$$
and
$$\begin{gathered} \forall (K^T_i, K^T_j) \in \mathcal{F}(\widetilde{K^T}) \times \mathcal{F}(\widetilde{K^T}) \\ \Pr[K^T = K^T_i | \text{View}_{\text{GPU}}] = \Pr[K^T = K^T_j | \text{View}_{\text{GPU}}] \end{gathered} \tag{22}$$

The above theorem states that, on observing transformed input matrices $\widetilde{Q}$ and $\widetilde{K^T}$, output matrix $\widetilde{QK^T}$ and next input matrix $(QK^T)'$, an attacker cannot distinguish between two matrices that belong to its feasible set. Thus, the feasible sets act as cloaking regions for the original matrices.

Proof. First, we present two helper lemmas as follows.

Lemma 1. *Attacker cannot reconstruct QK^T from $\widetilde{QK^T}$ and $(QK^T)'$.* Recall that the attacker (GPU) computes $\widetilde{QK^T} = \widetilde{Q} \cdot \widetilde{K^T}$ and subsequently transmits it to TEE. Within the TEE, $\widetilde{QK^T}$ undergoes a recovery process to QK^T through a specific linear transformation, denoted as $f(\cdot)$. Following this, QK^T is subjected to a masking process via another linear transformation, represented as $g(\cdot)$, resulting in the matrix $(QK^T)'$, which is then relayed back to the GPU for subsequent computations. Given the absence of knowledge regarding the intricacies and parameters of the linear transformations $f(\cdot)$ and $g(\cdot)$ on the part of the attacker, it is postulated that the reconstruction of the intermediate matrix QK^T from the accessible matrices $\widetilde{QK^T}$ and $(QK^T)'$ is not computationally feasible for the adversarial entity.

Lemma 2. *Random masked vectors $r_i \in R_Q$ or $r_j \in R_K$ are indistinguishable from transformed vectors $q_i \in Q$ or $k_j \in K$ which are the same shape.* Note that $\widetilde{Q}$ and $\widetilde{K^T}$ are embedded in a field $\mathbb{F}$. Thus clearly, masking the inputs is equivalent to applying a one-time pad [59]. And the $\widetilde{Q}$ and $\widetilde{K^T}$ are permuted before sending to GPU.

Theorem 2. Given the intricacies of the transformation process applied to the n-dimension matrix X, we can quantify the probability of correctly deducing the original matrix from its transformed version by the attacker. This quantification is simplified under certain assumptions, notably regarding the scalar coefficients used in the transformation. The theorem assumes these scalar coefficients are integers constrained within the range $(-L, L)$. This simplification is a significant factor in calculating the overall probability and is chosen for its practicality in mathematical modeling and computation.

$$\Pr[\text{Correct}] = \frac{\binom{n}{n}}{\binom{2n}{n}} \times \left(\frac{1}{2n-n}\right)^n \times \left(\frac{1}{2L}\right)^n \times \frac{1}{n!} \tag{23}$$

This equation encapsulates the aggregate probability, taking into account various factors including the selection of the correct vectors, identification of corresponding masks, accurate

prediction of scalar coefficients, and the reordering of the shuffled matrix.

Proof. The proof of Theorem 2 integrates the factors influencing the overall probability of an attacker correctly guessing the original matrix X. The probability is derived from the combination of several factors:

1) Selection of Original Vectors: The correct identification of the original n vectors from a total of $2n$ vectors is represented by the combinatorial ratio $\frac{\binom{n}{n}}{\binom{2n}{n}}$, indicating the statistical likelihood of choosing the exact subset of original vectors.
2) Mask Identification: In the obfuscation process, correctly identifying the appropriate masks from the remaining n vectors has a probability of $\left(\frac{1}{2n-n}\right)^n$, assuming each choice is independent and uniform.
3) Scalar Coefficients: The scalar coefficients, which are integers within the range $(-L, L)$, have a uniform probability distribution. Thus, the probability of correctly guessing all coefficients is $\left(\frac{1}{2L}\right)^n$.
4) Order Recovery: The likelihood of correctly re-establishing the original sequence in the shuffled matrix is represented by the permutation probability $\frac{1}{n!}$.

Given the transformed vector $X' = [x_1 - r_1, ..., x_i - r_i]$, it is infeasible for an attacker to recover the original vector $X = [x_1, x_2, ..., x_i]$. The level of security is guaranteed by the additive secret sharing [36], [37] which assumes that the attacker lacks knowledge of the random vector $[r_1, r_2, ..., r_i]$, which are essential for the reconstruction of the original vector.

A. Verification of OutSoftMax

Given the vector share of the GPU $[x_1 - r_1, ..., hashX - r_{hash}, ..., x_n - r_n]$, GPU is expected to return $[e^{x_1-r_1}, ..., e^{hashX-r_{hash}}, ..., e^{x_n-r_n}]$. Here we consider three specific types of attack as the supplement:

(a) the attacker tampers the results to

$$[e^{x_1-r_1+\Delta_1}, ..., e^{hashX-r_{hash}}, ..., e^{x_n-r_n}] \tag{24}$$

(b) the attacker tampers the results to

$$[e^{x_1-r_1} + \Delta_1, ..., e^{hashX-r_{hash}}, ..., e^{x_n-r_n}] \tag{25}$$

(c) the attacker swap the positions of different elements.

According to the verification process in Equation 15, the TEE will check if:

$$\left(e^{x_1+\Delta_1}\right)^{a_1} \prod_{i=2}^{n} (e^{x_i})^{a_i} = e^{hashX} \tag{26}$$

For the attack (a), the attacker needs also change:

$$e^{hashX-r_{hash}} \rightarrow e^{hashX-r_{hash}} e^{\Delta_1 a^1} \tag{27}$$

So it is crucial to discern the index of $hashX$ as well as the coefficients a_i that the TEE uses for verification of x_i in Equations 15.

Lemma 3. *Given the vector* $[x_1 - r_1, ..., hashX - r_{hash}, ..., x_n - r_n]$, $hashX - r_{hash}$ *is indistinguishable from any transformed* $x_i - r_i$. Since $x_i - r_i$ and $hashX - r_{hash}$ are embedded within a field $\mathbb{F}$ and the $hashX - r_{hash}$ is inserted by the TEE in an obfuscated location.

Theorem 3. Building upon Lemma 3, we can determine the probability of an attacker successfully tampering with k bits using attack (a) of an n-dimensional vector without detection by *U-Verify*:

$$\text{Pr[Correct]} = \frac{1}{n} \times \left(\frac{1}{2L}\right)^k \tag{28}$$

The factor $\frac{1}{n}$ reflects the probability of accurately identifying $hashX$, echoing the premise of Lemma 3. The following term, $\left(\frac{1}{2L}\right)^k$, represents the probability of correctly guessing all k coefficients a_i, where these coefficients are integers sampled from the interval $(-L, L)$.

For attack (b), the tampered item $e^{x_1-r_1} + \Delta_1$ will first be multiplied by e^{r_1} during the verification in TEE, and then the TEE will check if:

$$(e^{x_1} + \Delta_1 e^{r_1})^{a_1} \prod_{i=2}^{n} (e^{x_i})^{a_i} = e^{hashX} \tag{29}$$

so the attacker needs also know the r_1, a_1 and e^{x_1} to successful attack. The success probability is:

$$\text{Pr[Correct]} = \frac{1}{n} \times \left(\frac{1}{2L}\right)^k \times \left(\frac{1}{2L}\right)^k \times \left(\frac{1}{2L}\right)^k \tag{30}$$

For attacker (c), because the secret vector $\boldsymbol{a}$ will used in the integrity check as Equation 15, any mismatch of a_i and x_i will leads to the failure verification in Equation 15.

B. Verification of OutAttnMult

In the *U-Verify* process, the TEE first generates $hashQ$ by $hash_Q = h_Q \cdot Q$, and then outsources the concatenated matrix by *OutAttnMult*. The security are guaranteed by the lower probability of identify the $hashQ$ from the $\widetilde{Q}$ quantized in Equation 16. Upon the assumption that the attacker cannot identify the $hashQ$, the integrity are guaranteed using the variants of an algorithm by Freivalds [25].

Theorem 4. (Freivalds) *Let* Q, K^T *and* Z *be* $n \times n$ *matrices on over a field* $\mathbb{F}$ *and let* s *be a uniformly random vector in* $\mathbb{S} \subseteq \mathbb{F}$. *Then* $Pr[sZ = (sQ)K^T | Z \neq QK^T] = Pr[s(Z - QK^T) = 0 | (Z - QK^T) \neq 0] \leq \frac{1}{|\mathbb{S}|}$.

TABLE VI
ACCURACY OF MODELS AND QUANTIZED MODELS.

	Layers	Parameters	Metrics	Original	Quantized
ViT-16B	12	87M	ACC ↑	74.6%	72.7%
CLIP	24	151M	ACC ↑	73.6%	73.3%
BERT	12	110M	ACC ↑	92.4%	91%
LLaMA	32	7B	Perplexity ↓	5.68	5.89

In Table VI, we report the performance of evaluated models with and without the simple quantization scheme described in Section VI. Quantization results in at most a 1.9% drop in accuracy and a 0.21 perplexity increase.