Mosformer: Maliciously Secure Three-Party Inference Framework for Large Transformers

Ke Cheng*
Xidian University
Xi'an, China
chengke@xidian.edu.cn

Yuheng Xia*
Xidian University
Xi'an, China
yuhengxia@stu.xidian.edu.cn

Anxiao Song
Xidian University
Xi'an, China
songanxiao@stu.xidian.edu.cn

Jiaxuan Fu
Xidian University
Xi'an, China
jiaxuanfu@qq.com

Wenjie Qu
National University of Singapore
Singapore, Singapore
wenjiequ@u.nus.edu

Yulong Shen†
Xidian University
Xi'an, China
ylshen@mail.xidian.edu.cn

Jiaheng Zhang†
National University of Singapore
Singapore, Singapore
jhzhang@nus.edu.sg

Abstract

Transformer-based models like BERT and GPT have achieved state-of-the-art performance across a wide range of AI tasks but raise serious privacy concerns when deployed as cloud inference services. To address this, secure multi-party computation (MPC) is commonly employed, encrypting both user inputs and model parameters to enable inference without revealing any private information. However, existing MPC-based secure transformer inference protocols are predominantly designed under the semi-honest security model. Extending these protocols to support malicious security remains a significant challenge, primarily due to the substantial overhead introduced by securely evaluating complex non-linear functions required for adversarial resilience. We introduce Mosformer, the first maliciously secure three-party (3PC) inference framework that efficiently supports large transformers such as BERT and GPT. We first design constant-round comparison and lookup table protocols with malicious security, leveraging verifiable distributed point functions (VDPFs). Building on these, we develop a suite of 3PC protocols for efficient and secure evaluation of complex non-linear functions in transformers. Together with optimized modulus conversion, our approach substantially reduces the overhead of secure transformer inference while preserving model accuracy. Experimental results on the vanilla transformer block show that Mosformer achieves up to a 5.3× speedup and a 4.3× reduction in communication over prior maliciously secure protocols. Despite offering stronger security guarantees, Mosformer achieves comparable or even superior online performance to state-of-the-art semi-honest 2PC and 3PC frameworks, including BOLT (Oakland 2024), BumbleBee (NDSS 2025), SHAFT (NDSS 2025), and Ditto (ICML 2024), on full-scale models such as BERT and GPT-2.

CCS Concepts

• **Security and privacy → Privacy-preserving protocols**;

Keywords

Secure Transformer Inference, Malicious Security, Function Secret Sharing

ACM Reference Format:

Ke Cheng, Yuheng Xia, Anxiao Song, Jiaxuan Fu, Wenjie Qu, Yulong Shen, and Jiaheng Zhang. 2025. Mosformer: Maliciously Secure Three-Party Inference Framework for Large Transformers. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security (CCS '25), October 13–17, 2025, Taipei, Taiwan, China.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3719027.3765028

1 Introduction

Large transformer-based models like BERT [12], GPT [9], and DeepSeek [31] achieve state-of-the-art results across tasks from language understanding to code generation, driving their deployment as cloud inference services (e.g., ChatGPT). However, transmitting plaintext user inputs to remote servers poses serious privacy risks. For example, a ChatGPT bug in redis-py leaked parts of other users' chat histories and billing data [39]. While various privacy-preserving and trust-enhancing techniques have been explored to address different aspects of security in model deployment, including zero-knowledge proofs [33, 41, 54, 55] and watermarking methods [42, 43], there remains a growing need for inference systems that are not only accurate and efficient but also preserve both user data privacy and model confidentiality. To mitigate these risks,

*Partial work done when visiting Jiaheng's group at NUS.
†Corresponding authors.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

CCS '25, Taipei, Taiwan, China.

© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 979-8-4007-1525-9/2025/10
https://doi.org/10.1145/3719027.3765028

recent studies have proposed inference protocols that leverage homomorphic encryption [16] and secure multi-party computation (MPC) [53], either individually or in combination. These works protect user inputs and model parameters, enabling inference without revealing any additional information beyond the final output.

Many recent works have developed MPC-based inference protocols for large transformers across various settings. Secure two-party (2PC) protocols such as Iron [20], MPCformer [27], SIGMA [19], BOLT [40], Nimbus [29], BumbleBee [34], and SHAFT [22] enable private inference on BERT- and GPT-style models with hundreds of millions to billions of parameters. However, 2PC protocols often incur high communication costs and require frequent interaction, limiting their scalability for large models or long sequences. To address these limitations, recent works have explored three-party (3PC) settings, including PUMA [13], Ditto [51], and Privformer [1]. In particular, PUMA and Ditto achieve around 20 seconds per-token inference latency for GPT2-base under the semi-honest model. Privformer advances this line of work by adopting a stronger threat model—malicious security with an honest majority—offering robustness against actively adversarial parties. Despite this stronger security guarantee, deploying large transformers under malicious security remains challenging due to the significant computational and communication overhead introduced by adversarial protections. For instance, Privformer is approximately 5× slower than Ditto when evaluating the vanilla transformer [46]. Furthermore, it lacks efficient support for complex operations such as Softmax and GELU, limiting its scalability to larger models like BERT and GPT.

This challenge largely stems from the reliance of transformer architectures on complex non-linear functions such as GELU, Softmax, and LayerNorm. Securely evaluating these functions requires operations like comparisons, table lookups, divisions, and reciprocal square roots, which are not naturally compatible with efficient MPC protocols. Among them, comparison stands out as one of the most fundamental and frequently invoked building blocks, underpinning critical components such as activation functions and normalization layers. Although efficient protocols exist under the semi-honest model, their maliciously secure counterparts incur prohibitive overhead, making them unsuitable for large-scale deployment. For instance, Privformer uses the malicious comparison protocol from Falcon [49], which accounts for a large portion of total execution time, ranging from half in LAN to over 80% in WAN settings. (see Figure 3 in Appendix A in the full version [7]). A common alternative is to replace expensive activations with simpler or MPC-friendly functions, but this often degrades accuracy and cannot be directly applied to existing models. For example, MPCFormer [27] approximates GELU using quadratics and ReLU, improving efficiency but causing over 5% accuracy loss. Recovery typically requires distillation or fine-tuning, which rely on plaintext data and are often unrealistic in privacy-sensitive settings.

1.1 Our Contributions

To tackle these challenges, we introduce Mosformer, a maliciously secure three-party (3PC) inference framework for large transformers, which reduces communication overhead and rounds, thereby achieving low inference latency. Our design follows the same setting of [1, 37, 49], where all inputs are secret-shared across three non-colluding servers that execute the MPC protocols against one corrupted party (see Section 3.3 for details). Compared to state-of-the-art semi-honest frameworks, our protocol delivers comparable or even superior performance in both online runtime and communication, despite offering stronger malicious security guarantees. We summarize our contributions as follows.

- We present a communication-efficient 3PC comparison protocol, Π_{DReLU}, that achieves malicious security by leveraging *Verifiable Distributed Point Functions* (VDPF). The protocol supports constant-round online evaluation with minimal communication, while offloading expensive key generation to the offline phase. Specifically, it requires only constant 18 rounds and $24\ell - 8$ bits of communication per comparison for bit length $\ell = 32$, yielding a 1.6× reduction in rounds and 3.9× reduction in communication compared to Falcon [49], one of the most advanced maliciously secure 3PC frameworks. The protocol may be of independent interest.
- Additionally, we develop a maliciously secure 3PC lookup table protocol (i.e., Π_{LUT}) as a foundation for evaluating complex non-linear functions in transformers. Combining this with our comparison protocol, we construct maliciously secure and efficient protocols for inverse, reciprocal square root, GELU, Softmax, and LayerNorm. We also introduce secure modulus conversion protocols that dynamically adjust bit-lengths based on layer characteristics, reducing overhead while preserving high accuracy. Combining these techniques, we introduce Mosformer, the first maliciously secure 3PC inference framework that efficiently supports large transformers such as BERT and GPT.
- To validate the efficiency of our schemes, we implement all proposed protocols in Mosformer and release the code at https://github.com/XidianNSS/Mosformer. We evaluate Mosformer on several transformer models, including the vanilla transformer [46], BERT [12], and GPT-2 [9], measuring both model utility and inference efficiency. Results demonstrate that Mosformer significantly improves efficiency with minimal utility loss (within 2%). Compared to existing maliciously secure 3PC frameworks, Mosformer achieves 3.4-5.3× faster inference and uses 1.8-4.3× less communication than Privformer [1] and Falcon [49]. Remarkably, even when compared against state-of-the-art semi-honest solutions (both 2PC and 3PC settings), Mosformer still outperforms frameworks such as BOLT [40], Bumblebee [34], SHAFT [22], PUMA [13], and Ditto [51], reducing online inference time by 1.2-9×.

1.2 Technical Overview

1.2.1 Constant-Round Comparison Protocol with Malicious Security. Existing maliciously secure 3PC comparison protocols [25, 28, 37] typically reduce comparison to most significant bit (MSB) extraction, incurring $O(\ell)$ or $O(\log \ell)$ rounds and dominating overall runtime. Our goal is to design a constant-round, communication-efficient 3PC comparison protocol with malicious security. The core idea is to reduce signed comparisons to simpler unsigned ones, which are further mapped to secure equality-matching problems. We leverage recent advances in Distributed Point Functions (DPFs) to realize equality matching with minimal online communication. To ensure robustness against malicious behavior, we combine this with redundant evaluation across three parties, yielding an efficient maliciously secure comparison protocol.

We build upon a key observation from Cryptflow2 [44], which expresses MSB extraction as:

$$\mathrm{MSB}(x) = \mathrm{MSB}(\hat{x}) \oplus \mathrm{MSB}(2^\ell - r) \oplus 1\{\hat{y}_0 > 2^{\ell-1} - \hat{y}_1 - 1\}$$

where $x \in [0, 2^\ell - 1]$, $\hat{x} = x + r \bmod 2^\ell$, $\hat{y}_0 = \hat{x} \bmod 2^{\ell-1}$, and $\hat{y}_1 = (2^\ell - r) \bmod 2^{\ell-1}$. The first two MSB terms can be computed locally or preprocessed offline, leaving only the final comparison term for online evaluation. Since our protocol operates over the ring $\mathbb{Z}_{2^\ell}$, where values in $[0, 2^{\ell-1})$ represent non-negative integers, the remaining term $1\{\hat{y}_0 > 2^{\ell-1} - \hat{y}_1 - 1\}$ reduces to an unsigned comparison. While Falcon [49] supports maliciously secure unsigned comparisons, it incurs $\log \ell$ rounds due to repeated secure multiplications. To reduce interaction, we replace the multiplications with equality-matching operations using verifiable DPFs (VDPFs), enabling single-round online comparison. VDPFs ensure key correctness in the offline phase but provide no guarantees against online adversarial behavior. We thus introduce redundant three-party execution to enforce correctness during online evaluation under malicious security.

1.2.2 Secure, fast, and accurate 3PC protocols for the non-linear functions in transformers. Securely evaluating complex non-linear layers in transformers necessitates accurate computation of elementary functions, such as inverse, exponentiation, and reciprocal square root. Existing secure implementations typically rely on either function approximations or lookup tables (LUTs). Function approximations, including high-degree polynomials, Fourier, or iterative methods, incur substantial communication overhead to ensure accuracy under malicious security. Meanwhile, LUT-based approaches often require excessively large tables, making them inefficient, particularly in a malicious setting. For example, securely evaluating a 32-bit inverse function demands a prohibitively large LUT with 2^{32} entries.

To mitigate this inefficiency, we propose a domain reduction technique to significantly reduce LUT size without sacrificing accuracy. Specifically, we observe that the inverse function within Softmax layers always operates on positive inputs. For any such input x, there exists a scale base $b \in \mathbb{Z}_{2^\ell}$ and an index $k \in \mathbb{Z}_{2^\ell}$ satisfying $b^k \leq x < b^{k+1}$. By scaling the input by $b^{-(k+1)}$, the domain is confined to the interval $[1/b, 1)$. Thus, for a target floating-point precision f, we achieve a compact LUT of approximately $(1 - 1/b) \cdot 2^f$ entries. With a typical precision of $f = 12$, the LUT size remains under 2^{12}. This approach generalizes easily to other non-linear functions such as reciprocal square root under similar constraints.

To securely evaluate functions within the reduced domain, we construct a maliciously secure LUT protocol based on Verifiable Distributed Point Functions (VDPFs). By combining domain reduction with our VDPF-based lookup and comparison protocols, we obtain constant-round maliciously secure 3PC protocols for accurately evaluating a wide range of non-linear functions within transformers. In contrast to prior methods [1, 6, 27, 35], which rely on coarse approximations or architectural modifications, our approach preserves exact function semantics. Consequently, our solution eliminates the need for model retraining or fine-tuning, enhancing practicality and deployment flexibility.

1.2.3 Accelerating Maliciously Secure 3PC Transformer Inference via Modulus Conversion. We propose secure modulus conversion protocols that dynamically adjust the input bit-length based on layer-specific characteristics, significantly reducing computational and communication overhead while maintaining inference accuracy. Existing MPC-based transformer inference frameworks (e.g., SHAFT [22], PUMA [13], MPCFormer [27]) typically adopt 64-bit arithmetic to avoid overflow in linear-layer multiplications, thereby preventing errors or information leakage [37, 48]. In contrast, we observe that (1) inputs to non-linear layers in transformers are naturally bounded, and (2) our secure non-linear evaluation confines computation to small, well-defined domains. These observations allow for the direct evaluation of non-linear layers over smaller modulus rings (e.g., 16 or 32 bits). Experimental results demonstrate that, with negligible accuracy loss, our framework with modulus conversion achieves a 30% reduction in runtime and a 25% reduction in communication overhead compared to the baseline without modulus conversion.

2 Related Work

2.1 Secure Transformer Inference

Early work on secure inference primarily focused on convolutional neural networks (CNNs)[17, 32, 38, 45], but the emergence of transformers has introduced new challenges due to their scale and architectural complexity. As illustrated in Table 1, recent efforts address these challenges by developing transformer-specific secure frameworks, which can be categorized into three system models.

2PC Framework. Secure transformer inference under two-party computation (2PC) has advanced rapidly. Early works address nonlinear layers using rough surrogates, often requiring fine-tuning or retraining. For example, THE-X [6] replaces Softmax/GELU with ReLU-based surrogates, incurring accuracy drops over 5% and necessitating knowledge distillation and fine-tuning. Later 2PC frameworks, such as Iron [20], CipherGPT [21], and BOLT [40], improve accuracy and efficiency via function- or lookup-based approximations. BOLT integrates homomorphic encryption for linear layers and uses high-order piecewise polynomials to achieve near-floating-point accuracy without model modification. Nimbus [29], BumbleBee [34], and NEXUS [56] reduce communication by compressing ciphertexts or using low-degree approximations. Despite these advances, existing 2PC frameworks operate under the semi-honest model, leaving maliciously secure 2PC transformer inference an open and important challenge.

(2+1)-PC Framework. To improve the efficiency of 2PC protocols, several frameworks adopt the (2+1)-party model [3], where a trusted third party (TTP) provides auxiliary material during preprocessing. SHAFT [22] further improves Softmax and GELU accuracy via differential equations and Fourier approximations, reducing error by 51–76%. MPCFormer [27] replaces non-linear functions with MPC-friendly surrogates, but suffers from significant accuracy degradation (often >5%). SecFormer [35] improves GELU using Fourier approximation but retains MPCFormer's Softmax strategy, leading to similar accuracy losses. These issues are mitigated

Table 1: Comparison of Secure Transformer Inference Frameworks

	Framework	Malicious Security?	Cryptographic Tools				Nonlinear Layer Methods			Supported Models			Need Fine-tuning or Retraining?
			HE	OT	ASS	FSS	Rough Rep.	Func Appr.	LUT Appr.	BERT	GPT	Restricted	
2PC	THE-X [6]	No	●	○	○	○	●	○	○	●	○	○	Yes
	Iron [20]	No	○	●	●	○	○	○	●	●	○	○	No
	BOLT [40]	No	●	●	●	○	○	●	●	●	○	○	Yes
	CipherGPT [21]	No	●	●	●	○	○	●	●	○	●	○	No
	Nimbus [29]	No	●	○	●	○	○	●	○	●	○	○	No
	BumbleBee [34]	No	●	●	●	○	○	●	○	●	●	○	No
	NEXUS [56]	No	●	○	○	○	○	●	○	●	●	○	No
(2+1)-PC	SHAFT [22]	No	○	○	●	○	○	●	○	●	●	○	No
	MPCFormer [27]	No	○	○	●	○	●	○	○	●	○	○	Yes
	SecFormer [35]	No	○	○	●	○	○	●	○	●	○	○	Yes
	SIGMA [19]	No	○	○	●	●	○	○	●	●	●	○	No
3PC	PUMA [13]	No	○	○	●	○	○	●	○	●	●	○	No
	Ditto [51]	No	○	○	●	○	●	●	○	●	●	○	Yes
	Privformer [1]	Yes	○	○	●	○	●	●	○	○	○	●	Yes
	Mosformer (Ours)	Yes	○	○	●	●	○	○	●	●	●	○	No

HE: Homomorphic encryption, **OT**: Oblivious transfer, **ASS**: Additive secret sharing, **FSS**: Function secret sharing. For nonlinear layer methods, **Rough Rep.** means rough replacement, **Func Appr.** means function-based approximation, including polynomial approximation, Fourier approximation, and iterative approximation, **LUT Appr.** means Lookup table-based approximation. For supported models, **Restricted** indicates that the framework only supports the vanilla transformer block as proposed by Vaswani et al. [46], and does not extend to derived architectures such as BERT or GPT.

through fine-tuning and knowledge distillation, yet such retraining on plaintext data is impractical in many real-world scenarios. SIGMA [19] proposes a hybrid protocol combining Function Secret Sharing (FSS) and Additive Secret Sharing (ASS) to securely evaluate nonlinear layers in transformer models. By leveraging lookup tables during the online phase, it avoids costly interactive protocols and enables accurate, low-latency inference on large models such as GPT-2 and LLaMA-2, without requiring model fine-tuning. However, SIGMA depends on a TTP to precompute and distribute FSS keys. In contrast, Mosformer removes this trust assumption by introducing a symmetric three-party protocol that supports maliciously secure key generation without relying on any TTP.

3PC Framework. Another line of work explores three-party computation (3PC) with an honest-majority assumption to improve performance. Frameworks like PUMA [13] and Ditto [51] use 2-out-of-3 replicated sharing and lightweight ring arithmetic with preprocessed triples, achieving better efficiency than 2PC under the semi-honest model. Privformer [1] extends this setting to malicious security, which better reflects real-world deployment. Built on Falcon, it tailors 3PC protocols for ReLU and inverse square root, but approximates Softmax using ReLU-based attention [8], requiring fine-tuning to recover accuracy. This approach is limited to specific models like the vanilla transformer and is incompatible with general architectures such as BERT or GPT. In contrast, Mosformer is the only maliciously secure 3PC framework that supports efficient inference on large transformer models, including BERT and GPT, without model modifications or retraining.

2.2 Generic Maliciously-secure 3PC Frameworks with Honest Majority

Maliciously secure 3PC frameworks have attracted increasing attention in recent years. Representative systems include ABY3 [37], SWIFT [25], Pika [47], CryptFlow [26], Falcon [49], and binary circuit-based 3PC [28]. These frameworks assume an honest majority and leverage secret sharing, garbled circuits, or hybrid protocols to support privacy-preserving deep learning. ABY3 is an early hybrid design offering semi-honest performance with theoretical malicious extensions. Falcon and SWIFT achieve efficient malicious inference for CNNs like VGG16 and LeNet. Pika introduces function secret sharing to improve the efficiency of nonlinear operations such as Softmax and Sigmoid, paving the way for supporting more complex models. CryptFlow compiles TensorFlow models into secure protocols but relies on trusted hardware (SGX) and lacks support for transformer-specific operators.

Despite their progress, existing frameworks offer limited support for transformer inference. They are optimized for CNNs and traditional DNNs, but struggle with the distinct computation patterns of transformers. First, secure transformer inference requires numerous secure comparisons, which are inefficient in existing 3PC frameworks. ABY3 [37], SWIFT [25], and binary circuit-based 3PC [28] use bit-level circuits with 3ℓ AND gates per comparison, while Falcon [49] adopts random masking; all incur $O(\ell)$ or $O(\log \ell)$ rounds and $O(\ell)$ communication. In contrast, our protocol achieves $O(1)$ rounds by leveraging preprocessed VDPFs. Second, key transformer components such as GELU, Softmax, and LayerNorm remain inefficient or unsupported. To date, no mainstream maliciously secure 3PC framework has demonstrated end-to-end inference for large-scale transformers.

3 Preliminaries

3.1 Transformer

Large language models such as GPT and BERT are built upon the transformer architecture proposed by Vaswani et al. [46]. The transformer architecture typically consists of an input embedding layer followed by stacked encoder and decoder modules, both sharing a similar structure composed of linear and non-linear layers. The specific structure is described in Appendix B in the full version [7].

3.2 Cryptographic Tools

3.2.1 Notations. Let $P = (P_0, P_1, P_2)$ denote the parties. For a protocol Π, we write $[[x]]/\langle x\rangle \leftarrow \Pi$ to denote the execution of the protocol. Let $1\{\mathcal{P}\}$ denote the indicator function that is 1 when the predicate $\mathcal{P}$ is true and 0 when $\mathcal{P}$ is false. We denote m-dimensional vector as $\vec{x} \in \mathbb{Z}^m$, and refer to its i-th element (indexed from 0) as $\vec{x}[i]$. If $x \in \mathbb{Z}_{2^\ell}$ is a scalar, then $x[i]$ denotes the i-th bit in its binary representation, where the most significant bit corresponds to $i = \ell - 1$. The right rotate of $\vec{x}$ by k bits is denoted as $\vec{x} \ggg k$.

3.2.2 Additive Secret Sharing. Our work involves the following two additive secret sharing.

$\binom{n}{n}$-**sharing** $[[x]]^n$. A secret value $x \in \mathbb{Z}_{2^\ell}$ is said to be $[[\cdot]]$-shared among n parties, if party P_b for $b \in \mathbb{Z}_n$ holds $[[x]]^n_b$ such that $x = \sum_{i=0}^{n-1}[[x]]^n_i$. For brevity, we omit the corner n when there is no ambiguity. We use $n = 2$ and 3 in this paper.

$\binom{3}{2}$-**sharing** $\langle x\rangle$ (a.k.a. Replicated Secret Sharing, RSS). A secret value $x \in \mathbb{Z}_{2^\ell}$ is said to be $\langle\cdot\rangle$-shared among three parties, if party P_b for $b \in \mathbb{Z}_3$ holds $\langle x\rangle_b = ([[x]]_b, [[x]]_{b+1})$.

We extend the above definitions to vector inputs. Specifically, we use $[[\vec{x}]]$ and $\langle\vec{x}\rangle$ to denote $\binom{n}{n}$-sharing and $\binom{3}{2}$-sharing of a vector $\vec{x}$, respectively. Unless otherwise specified, all computations are performed in $\mathbb{Z}_{2^\ell}$. For brevity, we omit the explicit notation mod 2^ℓ. Furthermore, the aforementioned additive secret sharing schemes naturally extend to Boolean-valued data, which we refer to as Boolean sharing. We can achieve the conversion from Boolean sharing to additive sharing via the B2A protocol [24].

3.2.3 Basic Protocols with Semi-honest/Malicous Security. $\binom{3}{2}$-sharing supports the following semi-honest computation protocols:

- Reshare: Given $\binom{3}{3}$-shared $[[x]]$, to obtain $\langle x\rangle$, P_b for $b \in \mathbb{Z}_3$ locally computes $[[x']]_b = [[x]]_b + [[r]]_b$, where $[[r]]_0 + [[r]]_1 + [[r]]_2 = 0$ and the auxiliary parameter $[[r]]_b$ can be generated offline as [49]. Then, P_b sends $[[x']]_b$ to P_{b+1} and sets $\langle x\rangle_b = ([[x']]_b, [[x']]_{b+1})$.
- Add: Given $\binom{3}{2}$-shared $\langle x\rangle$ and $\langle y\rangle$, to obtain $\langle x+y\rangle$, P_b for $b \in \mathbb{Z}_3$ locally computes $\langle x+y\rangle_b = ([[x]]_b + [[y]]_b, [[x]]_{b+1} + [[y]]_{b+1})$. For a public constant c, $\langle x\rangle + c$ can be computed as $([[x]]_0 + c, [[x]]_1, [[x]]_2)$ where P_b compute its share locally.
- Mul: Given $\binom{3}{2}$-shared $\langle x\rangle$ and $\langle y\rangle$, to obtain $\langle xy\rangle$, P_b for $b \in \mathbb{Z}_3$ locally computes $[[z]]_b = [[x]]_b[[y]]_b + [[x]]_{b+1}[[y]]_b + [[x]]_b[[y]]_{b+1}$ so that $z = [[z]]_0 + [[z]]_1 + [[z]]_2 = xy$ and z is $\binom{3}{3}$-shared. Then, parties invoke Reshare with inputting $[[z]]$ to get $\langle z\rangle = \langle xy\rangle$. Additionally, for a public constant c, $c \cdot \langle x\rangle$ can be computed as $(c \cdot [[x]]_0, c \cdot [[x]]_1, c \cdot [[x]]_2)$ where P_b can compute its share locally.

$\binom{3}{2}$-**sharing** (i.e., RSS) satisfies a *consistency property*, ensuring that addition and multiplication with a constant preserve share consistency. Let $(a_0, b_0), (a_1, b_1), (a_2, b_2)$ be the RSS shares held by parties P_0, P_1, P_2, respectively. The shares are consistent if and only if $a_i = b_{i+2}$ for all $i \in \mathbb{Z}_3$. Mosformer builds on the following ideal functionalities to achieve malicious security: $\mathcal{F}_{\text{trunc}}$, $\mathcal{F}_{\text{mul}}^{\langle\cdot\rangle}$, $\mathcal{F}_{\text{rand}}$, $\mathcal{F}_{\text{share}}$, $\mathcal{F}_{\text{recon}}$, $\mathcal{F}_{\text{open}}$, and $\mathcal{F}_{\text{MacCheck}}$. Each can be securely realized under malicious security using standard protocols [2, 15, 30, 37].

- $\mathcal{F}_{\text{trunc}}$ [37]: Given a $\binom{3}{2}$-shared $\langle x\rangle$ and a number f, share $\langle x/2^f\rangle$ between three parties.
- $\mathcal{F}_{\text{mul}}^{\langle\cdot\rangle}$ [30]: Given $\binom{3}{2}$-shared $\langle x\rangle$, $\langle y\rangle$, share $\langle x \cdot y\rangle$ between three parties.
- $\mathcal{F}_{\text{rand}}$ [15]: Sample a random value $r \xleftarrow{\$} \mathbb{Z}$ and share $\langle r\rangle$ between three parties.
- $\mathcal{F}_{\text{share}}$ [15]: Given a secret x held by P_b, share $\langle x\rangle$ between three parties.
- $\mathcal{F}_{\text{recon}}$ [15]: Given a $\binom{3}{2}$-shared $\langle x\rangle$ and a party index b, reveal x to P_b.
- $\mathcal{F}_{\text{open}}$ [15]: Given a $\binom{3}{2}$-shared $\langle x\rangle$, reveal x to all three parties.
- $\mathcal{F}_{\text{MacCheck}}$ [2]: Given $\binom{3}{2}$-shared $\langle x\rangle$, MAC key $\langle\alpha\rangle$ and the MAC tag $\langle\sigma_x\rangle$, the protocol aborts if $\alpha \cdot x \neq \sigma_x$.

3.2.4 Function Secret Sharing. A Function Secret Sharing (FSS) scheme [4, 5] for a function family $\mathcal{F}$ splits a function $f \in \mathcal{F}$ into two additive shares $f_0(x)$ and $f_1(x)$, such that each share individually reveals nothing about f, and for all $x \in \mathbb{G}^{\text{in}}$, it holds that $f_0(x) + f_1(x) = f(x)$.

Definition 1 (FSS Syntax [4, 5]). *A (two-party) FSS scheme consists of a pair of probabilistic polynomial-time (PPT) algorithms { Gen, Eval }:*

- *Gen*$(1^\lambda, f) \to (k_0, k_1)$*: Given a security parameter λ and a function $f \in \mathcal{F}$, outputs two functional keys k_0, k_1, which encode f_0, f_1 respectively, without revealing f.*
- *Eval*$(b, k_b, x) \to f_b(x)$*: Given party index $b \in \{0, 1\}$, key k_b, and input x, returns an additive share $f_b(x)$, such that $f_0(x) + f_1(x) = f(x)$.*

The pair (k_0, k_1) is referred to as the *FSS keys*, and the number of bits required to store each key is called the *key size*.

3.2.5 Verifiable Distributed Point Function (VDPF). Distributed Point Function (DPF) is a canonical construction in FSS for point functions, which evaluate to a non-zero value at exactly one point in the domain. Formally, a point function $f_{\alpha,\beta} : \{0, 1\}^n \to \mathbb{F}$ is defined as:

$$f^\bullet_{\alpha,\beta}(x) = \begin{cases} \beta, & x = \alpha \\ 0, & otherwise \end{cases} \tag{1}$$

where $\alpha \in \{0, 1\}^n$ denotes the target index, and $\beta \in \mathbb{F}$ is the non-zero value at that index.

In the presence of a malicious DPF key generator, incorrect keys may be produced. To address this, verifiable DPF [11] introduces a verifiability property, defined as follows:

Definition 2. *(Verifiable DPF [11]). A (two-party) verifiable distributed point function (VDPF) scheme consists of probabilistic polynomial-time (PPT) algorithms {Gen(·), BVEval(·), Verify(·)}:*

- *VDPF.Gen*$(1^\lambda, f^\bullet_{\alpha,\beta}) \to (k_0, k_1)$*: Given security parameter λ and function $f^\bullet_{\alpha,\beta}$, outputs a pair of functional keys (k_0, k_1).*
- *VDPF.BVEval*$(b, k_b, \{x_i\}) \to ([[y_b^{(x_i)}]], \pi_b)$ *for $i \in \{1, \ldots, L\}$: Given party index b, key k_b, and a batch of inputs $\{x_i\}$, returns evaluations $[[y_b^{(x_i)}]] = f^\bullet_{\alpha,\beta}(x_i)$ and a proof π_b of correctness.*
- *VDPF.Verify*$(\pi_0, \pi_1) \to (Accept/Reject)$*: Given a pair of proofs π_0 and π_1, outputs either Accept or Reject. The output should only be Accept if $y_0 + y_1$ defines the truth table of some point function, which occurs if it is non-zero in at most one location.*

3.3 System Overview

We consider the inference setting, where a model owner provides a pre-trained model $\mathcal{M}$, and a client supplies input data x. The computation is formalized as $y = \mathcal{M}_\theta(x)$, where θ denotes the model parameters. Following prior work [1, 13, 51], we adopt a secure outsourced three-party computation model, in which the inference is delegated to an MPC system comprising three parties $P = (P_0, P_1, P_2)$. The client secret-shares the input $\langle x \rangle$ using replicated secret sharing (RSS) and distributes the shares to the parties. Likewise, the model owner secret-shares the parameters $\langle \theta \rangle$ and sends them to P. The parties then collaboratively execute an interactive protocol over the shared inputs to compute the result $\langle y \rangle$, which is returned to the client for reconstruction. Throughout the protocol, no individual party learns any information about the client's input or the model weights.

Threat Model. Consistent with the threat models adopted in [1, 2], we assume a security model with abort under an honest-majority setting against malicious adversaries, where at most one out of three parties may be corrupted. A semi-honest adversary follows the protocol but attempts to infer private information from intermediate values. In contrast, a malicious adversary may arbitrarily deviate from the protocol, including tampering, interrupting, or forging messages to compromise correctness or privacy.

4 Maliciously Secure Comparison Protocol

Secure comparison protocols, a fundamental building block for implementing various non-linear operations, are typically reduced to most significant bit (MSB) extraction in existing maliciously secure 3PC frameworks [25, 28, 37]. However, such approaches incur high communication and round complexity, affecting overall performance. Our protocol reduces signed comparison to unsigned, and further to secure equality tests, efficiently implemented via VDPFs with minimal online communication. Malicious robustness is ensured through redundant three-party evaluation, yielding an efficient comparison protocol. We first present a protocol for unsigned comparison, upon which we build maliciously secure 3PC protocols for DReLU and Selection with constant-round complexity.

4.1 Comparison of Unsigned Integers

For two values $x, y \in \mathbb{U}_{2^n}$[1], if $x < y$, there exists an index $k \in [0, n-1]$ that satisfies $x[n-1] = y[n-1], \cdots, x[k+1] = y[k+1], x[k] = 0$ and $y[k] = 1$, where $x[i]$ represents the i-th bit of x. Our protocol builds on the bitwise comparison logic used in Falcon [49] for unsigned integers. Specifically, we define

$$\vec{u}[i] = x[i] - y[i], \quad \vec{w}[i] = x[i] \oplus y[i], \quad \vec{c}[i] = \vec{u}[i] + 1 + \sum_{j=i+1}^{n} \vec{w}[j].$$

If $x \geq y$, all elements of $\vec{c}$ are positive; in contrast, if $x < y$, exactly one element of $\vec{c}$ equals zero. Thus, comparing x and y can be reduced to checking whether any element of $\vec{c}$ is zero.

Falcon realizes the above functionality using ASS-based interactive protocols that require $8 + 4\log_2 \ell$ rounds and $O(\ell \log \ell)$ bits of communication. In our protocol that realizes the same functionality, we employ the verifiable distributed point function (VDPF) [11] to evaluate all elements in $\vec{c}$ in parallel, thereby avoiding costly

Table 2: A toy example of Π_{UCMP} ($n = 4$)

	$x=7, y=5$				$x=3, y=5$			
i	3	2	1	0	3	2	1	0
$x[i]$	0	1	1	1	0	0	1	1
$y[i]$	0	1	0	1	0	1	0	1
$\vec{u}[i]$	0	0	1	0	0	-1	1	0
$\vec{w}[i]$	0	0	1	0	0	1	1	0
$\vec{c}[i]$	1	1	2	2	1	0	3	3
$f^{\bullet}_{0,1}(\vec{c}[i])$	0	0	0	0	0	1	0	0
$x < y \Leftrightarrow \sum_{i=0}^{n-1} f^{\bullet}_{0,1}(\vec{c}[i])$	0				1			

Algorithm 1 Unsigned Integer Comparison (Π_{UCMP})

Input: P_b and P_{b+1} input $\binom{2}{2}$-shared bitwise representation $[[\vec{x}]]$ of a private value $x \in \mathbb{U}_{2^n}$, where $[[\vec{x}[i]]] \in \mathbb{Z}_{2n}$, for $i \in [0, n-1]$, along with a public integer $y \in \mathbb{U}_{2^n}$.

Output: P_b and P_{b+1} output $\binom{2}{2}$-shared $[[res]]$, where $res = 1\{x < y\}$.

[Setup] Upon initialization, the party P_{b+2} does:

- **for** $i = \{0, 1, 2, \cdots, n-1\}$ **do**
  - Randomly sample $\vec{\alpha}[i] \xleftarrow{\$} \mathbb{Z}_{2n}$ and $[[\vec{\alpha}[i]]]_0 \xleftarrow{\$} \mathbb{Z}_{2^n}$
  - $[[\vec{\alpha}[i]]]_1 = \vec{\alpha}[i] - [[\vec{\alpha}[i]]]_0$
  - $(\vec{k}^{\bullet}_0[i], \vec{k}^{\bullet}_1[i]) \leftarrow \mathsf{VDPF.Gen}(1^\lambda, \vec{\alpha}[i], 1)$
- **end for**
- P_{b+2} sends $([[\vec{\alpha}]]_0, \vec{k}^{\bullet}_0)$ to P_b and $([[\vec{\alpha}]]_1, \vec{k}^{\bullet}_1)$ to P_{b+1}.
- P_b and P_{b+1} jointly run the DPF key verification protocol in [11] to check the well-formness of the VDPF keys. If the check fails, abort.
- **for** $i = \{0, 1, 2, \cdots, n-1\}$ **do**
  - P_b and P_{b+1} expand its DPF keys on domain $\mathbb{Z}_{2n}$[2] to produce $\binom{2}{2}$-shared vectors $[[V[i]]]$ by $\mathsf{VDPF.BVEval}$, where V is a two-dimensional array and $[[V[i]]]$ is produced by $\vec{k}^{\bullet}_0[i]$ and $\vec{k}^{\bullet}_1[i]$, which represents the share of a one-hot vector at point $\vec{\alpha}[i]$.
  - P_b and P_{b+1} jointly compute:
  - $[[t]] = \sum_{j=0}^{2n} [[V[i][j]]]$
  - $[[s]] = [[\vec{\alpha}[i]]] - \sum_{j=0}^{2n} (j \cdot [[V[i][j]]])$
  - P_b and P_{b+1} open $[[t]], [[s]]$ then check if $t = 1$ and $s = 0$. If the check fails, abort.
- **end for**

[Evaluation] Upon receiving the sharing value $[[\vec{x}]]$ and a public n-bit integer y, the party P_b and P_{b+1} do:

- **for** $i = \{n-1, n-2, \cdots, 0\}$ **do**
  - $[[\vec{u}[i]]] = [[\vec{x}[i]]] - y[i]$
  - $[[\vec{w}[i]]] = [[\vec{x}[i]]] \oplus y[i]$
  - $[[\vec{c}[i]]] = [[\vec{u}[i]]] + 1 + \sum_{k=i+1}^{n} [[\vec{w}[k]]]$
- **end for**
- **for** $i = \{0, 1, 2, \cdots, n-1\}$ **do**
  - $[[\vec{\delta}[i]]] = [[\vec{c}[i]]] + [[\vec{\alpha}[i]]]$
- **end for**
- P_b and P_{b+1} open $\vec{\delta}$ over ring $\mathbb{Z}_{2n}$.
- **return** $[[res]] = \sum_{i=0}^{n-1} [[V[i][\vec{\delta}[i]]]]$.

[1] $\mathbb{U}_{2^n}$ denotes the set of n-bit unsigned integers.

[2] Since the maximum value of elements in $\vec{c}$ is bounded by $n+1$, it is sufficient to execute the VDPF over the smaller domain $\mathbb{Z}_{2n}$ rather than $\mathbb{Z}_{2^n}$, thereby significantly reducing the full-domain evaluation overhead of VDPF keys in the setup phase.

interactions and achieving a protocol with a constant round. Table 2 illustrates a toy example of the above evaluation process.

We formally describe our secure unsigned comparison protocol (Π_{UCMP}) in Algorithm 1. Given a $(2,2)$-shared bitwise representation $[[\vec{x}]]$ of a private input $x \in \mathbb{U}_{2^n}$, shared between parties P_b and P_{b+1}, and a public input $y \in \mathbb{U}_{2^n}$, the goal is to securely compute whether $x < y$. At the end of the protocol, parties P_b and P_{b+1} receive a $(2,2)$-shared output $[[res]]$, where $res = \mathbf{1}\{x < y\}$. Party P_{b+2} acts as a helper during the setup phase to facilitate the generation of DPF keys.

The setup phase of protocol Π_{UCMP} is designed to ensure security against malicious behavior during DPF key generation. A corrupted key provider P_{b+2} may generate incorrect keys, such as for a point function $f^{\bullet}_{\alpha,\beta}$ with $\beta \neq 1$, violating the assumption $\beta = 1$ and compromising correctness. Alternatively, the adversary may send valid keys but share an incorrect index $\alpha^* \neq \alpha$ with P_b and P_{b+1}, leading to incorrect evaluation $\delta^* = c + \alpha^*$. To prevent such errors, we adopt the VDPF scheme [11], which enforces correct key generation. During full-domain evaluation, a $\binom{2}{2}$-shared vector $[[\vec{v}]]$ is constructed, and correctness of β is ensured by checking $\sum_i [[\vec{v}[i]]] = 1$. To verify the shared index α, the parties compute $[[s]] = [[\alpha]]_b - \sum_{i \in \mathbb{Z}_{2n}} i \cdot [[\vec{v}[i]]]$ and reveal s, accepting only if $s = 0$. It should be noted that the online evaluation phase remains vulnerable to malicious attacks, such as additive errors when opening $\vec{\delta}$. In the next subsection, we show how to build a maliciously secure 3PC comparison protocol based on Π_{UCMP}.

Analysis. To avoid expensive online computation, P_b and P_{b+1} perform a full-domain evaluation of the VDPF during the setup phase, precomputing outputs for all possible inputs. This allows direct retrieval in the online phase without invoking VDPF.Eval, significantly reducing latency and computation. The key size of Π_{UCMP} is $n \cdot ((\lambda + 2) \cdot \log_2 2n + n + 4\lambda)$ bits, where λ is the VDPF security parameter. Online communication is limited to $n \log_2 2n$ bits in a single round (see Appendix D.1 in [7] for details).

4.2 DReLU

For an ℓ-bit value $x \in \mathbb{Z}_{2^\ell}$ in 2's complement notation, DReLU (i.e., derivative of ReLU) is defined as:

$$\mathsf{DReLU}(x) = 1\{x < 2^{\ell-1}\} = 1 - \mathsf{MSB}(x) \tag{2}$$

The most significant bit of x, i.e., $\mathsf{MSB}(x)$, can be computed following the approach of [14, 19, 44], using the following formula:

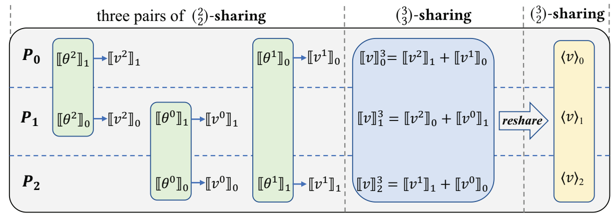
$$\mathsf{MSB}(x) = \mathsf{MSB}(x_0) \oplus \mathsf{MSB}(x_1) \oplus 1\{y_0 + y_1 \geq 2^{\ell-1}\} \tag{3}$$

where $x = x_0 + x_1$, $y_0 = x_0 \mod 2^{\ell-1}$ and $y_1 = x_1 \mod 2^{\ell-1}$.

Then, we use a random number $r \in \mathbb{Z}_{2^\ell}$ to blind x, yielding $\hat{x} = x + r \mod 2^\ell$. $\mathsf{MSB}(x)$ is subsequently computed as:

$$\mathsf{MSB}(x) = \mathsf{MSB}(\hat{x}) \oplus \mathsf{MSB}(2^\ell - r) \oplus 1\{\hat{y}_0 > 2^{\ell-1} - \hat{y}_1 - 1\} \tag{4}$$

where $\hat{y}_0 = \hat{x} \mod 2^{\ell-1}$, $\hat{y}_1 = (2^\ell - r) \mod 2^{\ell-1}$. The core idea of our 3PC protocol stems from the following observation: the first MSB term can be computed locally, while the second MSB term can be precomputed during the setup phase. As a result, only the final comparison needs to be performed in the online phase, which merely involves invoking Π_{UCMP} to compute $\mathbf{1}\{\hat{y}_0 > 2^{\ell-1} - \hat{y}_1 - 1\}$, where $\hat{y}_0$ is a public input and the bitwise representation of $2^{\ell-1} - \hat{y}_1 - 1$ is precomputed and secret-shared.



represents that P_b invoke Π_{UCMP} and B2A with P_{b+i+1} to obtain $[[\theta^{b-i+2}]]_{1-i}$, represents that calculating $[[v^{b-i+2}]]_{1-i} = [[1]]_{b+i+1} \cdot (1 - [[\theta^{b-i+2}]]_{1-i})$, where $b \in \{0,1,2\}$ and $i \in \{0,1\}$.

Figure 1: High-Level Design of Secure DReLU Protocol

We describe our maliciously secure DReLU protocol (Π_{DReLU}) in Algorithm 2. Given a $(3,2)$-shared input $\langle x \rangle$ where $x \in \mathbb{Z}_{2^\ell}$, the goal is to securely evaluate DReLU, which returns 1 if $x > 0$ and 0 otherwise. At the end of the protocol, the parties obtain a $(3,2)$-shared output $\langle v \rangle$, where $v = \mathsf{DReLU}(x)$. At a high level (see Figure 1), each pair among P_b, P_{b+1}, P_{b+2} jointly runs the Π_{UCMP} protocol to generate three $(2,2)$-shared comparison results. These are converted into a single $(3,2)$-shared output using the RSS consistency property, ensuring correctness against malicious behavior.

Specifically, in the setup phase, each pair among P_b, P_{b+1}, P_{b+2} jointly generates and distributes the $(2,2)$-shared terms $\mathsf{MSB}(2^\ell - r)$ and $2^{\ell-1} - \hat{y}_1 - 1$, followed by verification by all three parties (Lines 1–9). The parties then invoke the setup phase of Π_{UCMP} to generate and verify required parameters (Line 10), and generate the $(3,2)$-shared value $\langle 1 \rangle$, where both P_b and P_{b+1} hold $[[1]]_b$ (Line 11). In the evaluation phase, the parties compute the blinded input $\hat{x}$ from the $(3,2)$-shared $\langle x \rangle$ (Lines 12–17). Using Eq. (5) and Eq. (7), they invoke Π_{UCMP} with auxiliary parameters to compute the $(3,3)$-shared value $[[v]]$, where $v = 1 - \mathsf{MSB}(x)$ (Lines 18–25). Finally, they invoke Reshare to convert $[[v]]$ into a $(3,2)$-shared $\langle v \rangle$, and invoke $\mathcal{F}_{\mathsf{MacCheck}}$ to verify output correctness (Lines 26–29).

Security Against Malicious Adversaries. Π_{DReLU} consists of three interactive parts: 1) generation and distribution of VDPF keys and auxiliary parameters; 2) reconstruction of secret-shared blinded input; and 3) secure evaluation and resharing of the result for Eq. (7). We describe the defense mechanisms in each part to resist malicious adversaries.

1) The potentially malicious party P_b may distribute incorrect VDPF keys and auxiliary parameters. While VDPF key verification is covered in Section 4.1, we focus here on validating the auxiliary parameters, including the secret shares of r^b, c^b, and the bitwise representation $\vec{y}^b$. These parameters are considered correct if and only if they satisfy: $c = \mathsf{MSB}(2^\ell - r)$, and $y = 2^\ell - 1 - \big((2^{\ell-1} - r) \mod 2^{\ell-1}\big)$. Upon receiving $[[r^b]]$, the semi-honest parties P_{b+1} and P_{b+2} compute $[[c^*]] = [[\mathsf{MSB}(2^\ell - r^b)]]$, $[[y^*]] = 2^\ell - 1 - \big((2^{\ell-1} - [[r^b]]) \mod 2^{\ell-1}\big)$. To verify the correctness of $[[c^b]]$ and $[[\vec{y}^b]]$, they check whether $c^b = c^*$, $\sum_{i=0}^{\ell-2} (\vec{y}^b[i] \cdot 2^i) = y^*$. Accordingly, P_{b+1} and P_{b+2} locally compute $[[\delta_c]] = [[c^*]] - [[c^b]]$, $[[\delta_y]] = [[y^*]] - \sum_{i=0}^{\ell-2} ([[\vec{y}^b[i]]] \cdot 2^i)$, and jointly reveal δ_c, δ_y. If either value is non-zero, the protocol aborts.

2) The malicious party P_b may introduce errors during the reconstruction of the blinded input $\hat{x}$. Since P_b shares $[[r^b]]$ with P_{b+1} and P_{b+2}, the parties can define a consistent $\binom{3}{2}$-sharing $\langle r^b \rangle$ from $[[r^b]]$, where $\langle r^b \rangle = ([[r^b]]_0, [[r^b]]_1)$, $\langle r^{b+1} \rangle = (0, [[r^{b+1}]]_0)$, and

Algorithm 2 DReLU (Π_{DReLU})

Input: $\binom{3}{2}$-shared $\langle x\rangle$, where $x \in \mathbb{Z}_{2^\ell}$.
Output: $\binom{3}{2}$-shared $\langle v\rangle$, where $v = \mathsf{DReLU}(x)$.
[Setup] Upon initialization, the party P_b ($b \in \mathbb{Z}_3$) does:

1: Randomly sample $r^b \xleftarrow{\$} \mathbb{Z}_{2^\ell}$ and $[[r^b]]_0 \xleftarrow{\$} \mathbb{Z}_{2^\ell}$.
2: Compute $[[r^b]]_1 = r^b - [[r^b]]_0$.
3: Compute $c^b = \mathsf{MSB}(2^\ell - r^b)$, $y^b = 2^\ell - 1 - ((2^\ell - r^b) \mod 2^{\ell-1})$.
4: Sample $[[c^b]]_0, [[c^b]]_1$ such that $[[c^b]]_0 \oplus [[c^b]]_1 = c^b$ over $\mathbb{Z}_2$.
5: Sample $[[\vec{y}^b[i]]]_0, [[\vec{y}^b[i]]]_1$ such that $[[\vec{y}^b[i]]]_0 + [[\vec{y}^b[i]]]_1 = \vec{y}^b[i]$ for $i \in [0, \ell-2]$ over $\mathbb{Z}_{2(\ell-1)}$.
6: P_b sends $([[r^b]]_0, [[c^b]]_0, [[\vec{y}^b]]_0)$ to P_{b+2} and sends $([[r^b]]_1, [[c^b]]_1, [[\vec{y}^b]]_1)$ to P_{b+1}.
7: P_{b+1} and P_{b+2} locally compute $[[r']] = 2^\ell - [[r^b]]$ and $[[y^*]] = 2^\ell - 1 - ((2^{\ell-1} - [[r]]) \mod 2^{\ell-1})$.
8: P_{b+1} and P_{b+2} jointly compute $[[c^*]] = [[\mathsf{MSB}(r')]]$ using semi-honest secure MSB protocol in [32] and compute $[[\delta_c]] = [[c^*]] - [[c]]$ and $[[\delta_y]] = [[y^*]] - \sum_{i=0}^{\ell-2}([[\vec{y}[i]]] \cdot 2^i)$ locally.
9: P_{b+1} and P_{b+2} jointly reveal δ_c and δ_y. If $\delta_c \neq 0$ or $\delta_y \neq 0$, abort.
10: The parties jointly invoke the setup of Π_{UCMP} to generate and check the required parameters. If the check fails, abort.
11: The parties invoke $\langle\alpha\rangle \leftarrow \mathcal{F}_{rand}(\mathbb{Z}_{2^\ell})$ to share a MAC key $\alpha \in \mathbb{Z}_{2^\ell}$ and jointly generate a shared value $\langle 1\rangle = ([[1]]_b, [[1]]_{b+1})$.

[Evaluation] Upon receiving $\langle x\rangle$, the party P_b ($b \in \mathbb{Z}_3$) does:

12: Set $\langle r^b\rangle = ([[r^b]]_0, [[r^b]]_1)$, $\langle r^{b+1}\rangle = (0, [[r^{b+1}]]_0)$, and $\langle r^{b+2}\rangle = ([[r^{b+2}]]_1, 0)$.
13: **for** $i \in \{0,1,2\}$ **do**
14: $\quad\langle\hat{x}_i\rangle = \langle x\rangle + \langle r^i\rangle$
15: $\quad$Party P_{i+1} obtains $\hat{x}_i$ by invoking $\mathcal{F}_{\mathsf{recon}}(\langle\hat{x}_i\rangle, i+1)$
16: $\quad$Party P_{i+2} obtains $\hat{x}_i$ by invoking $\mathcal{F}_{\mathsf{recon}}(\langle\hat{x}_i\rangle, i+2)$
17: **end for**
18: **for** $i \in \{0,1\}$ **do**
19: $\quad[[res^{b-i+2}]]_{1-i} \leftarrow \Pi_{\mathsf{UCMP}}([[\vec{y}^{b-i+2}]]_{1-i},\ \hat{x}_{b-i+2} \mod 2^{\ell-1})$
20: $\quad\theta_{1-i}^{b-i+2} = i \cdot \mathsf{MSB}(\hat{x}_{b-i+2}) \oplus [[c^{b-i+2}]]_{1-i} \oplus [[res^{b-i+2}]]_{1-i}$
21: **end for**

P_b and P_{b+1} ($b \in \mathbb{Z}_3$) do:

22: Obtain $[[\theta^{b+2}]]$ by B2A with P_b input θ_1^{b+2} and P_{b+1} input θ_0^{b+2}.
23: $[[v_0]] = [[1]]_{b+1} \cdot (1 - [[\theta^{b+2}]])$, $[[mv_0]] = [[\alpha]]_{b+1} \cdot (1 - [[\theta^{b+2}]])$.

P_b and P_{b+2} ($b \in \mathbb{Z}_3$) do:

24: Obtain $[[\theta^{b+1}]]$ by B2A with P_b input θ_1^{b+1} and P_{b+2} input θ_0^{b+1}.
25: $[[v_1]] = [[1]]_b \cdot (1 - [[\theta^{b+1}]])$, $[[mv_1]] = [[\alpha]]_b \cdot (1 - [[\theta^{b+1}]])$.

P_0, P_1, and P_2 collaboratively compute:

26: $[[v]] = [[v_0]] + [[v_1]]$, $[[mv]] = [[mv_0]] + [[mv_1]]$
27: $\langle v\rangle \leftarrow \mathsf{Reshare}([[v]])$, $\langle mv\rangle \leftarrow \mathsf{Reshare}([[mv]])$
28: Invoke $\mathcal{F}_{\mathsf{MacCheck}}$ on $\langle v\rangle$ and $\langle mv\rangle$. Abort if the check fails.
29: **return** $\langle v\rangle$.

$\langle r^{b+2}\rangle = ([[r^{b+2}]]_1, 0)$. Note that $\langle x\rangle$ is already a consistent $\binom{3}{2}$-sharing, and since $\langle\hat{x}\rangle = \langle r\rangle + \langle x\rangle$, the resulting share $\langle\hat{x}\rangle$ remains consistent. Given that $\langle\hat{x}\rangle$ is consistent, the parties can reconstruct $\hat{x}$ correctly to P_{b+1} and P_{b+2} using the functionality $\mathcal{F}_{\mathsf{recon}}$.

3) The malicious party may introduce errors either during the computation of the $\binom{2}{2}$-shared DReLU result or in the subsequent resharing of the $\binom{3}{3}$-shared output. Since all such errors can be attributed to additive errors introduced before the resharing phase, and at least two of the three parties are semi-honest, it is guaranteed that at least one pair of correct $\binom{2}{2}$-shared values exists. As a result, the additive errors introduced prior to resharing can be efficiently detected using the functionality $\mathcal{F}_{\mathsf{MacCheck}}$.

Analysis. Table 3 presents a complexity comparison of secure DReLU protocols with state-of-the-art frameworks, including Falcon [49] and the protocol of Li et al. [28]. This advantage is further validated by empirical results in Table 4.

Table 3: Complexity Comparisons of Secure DReLU Protocols with SOTA, including Falcon (PETS'21) and the work of Li et al. (USENIX Security'23)

Protocol	Rounds	Comm. Cost (bits)
Falcon [49]	$9 + 4\log\ell$	$(\frac{29}{2}\log p + 1)\ell - 5\log p$
Li et al. [28]	$\frac{6\log_\lambda(4m)}{m}\ell + \log\ell$	$\frac{6\log_\lambda(4m)\log p}{m}\ell + 3\ell$
Ours	18	$16\ell + 2(\ell-1)\log\ell$

ℓ denotes the bit length, m is the number of AND gates processed in parallel in a single batch, λ is the degree of the polynomial used in the zero-knowledge proof, and p is a prime number.

4.3 Extension to Selection

The secure DReLU protocol naturally extends to the Select functionality with malicious security, where $\mathsf{Select}(x,y) = 1\{x \geq 0\} \cdot y$. Given $\binom{3}{2}$-shared values $\langle x\rangle$ and $\langle y\rangle$, the protocol Π_{Select} outputs $\langle y\rangle$ if $x \geq 0$, and $\langle 0\rangle$ otherwise. This supports ReLU and Max, since $\mathsf{ReLU}(x) = \mathsf{Select}(x,x)$ and $\mathsf{Max}(x,y) = \mathsf{Select}(x-y, x-y) + y$. A naïve realization of Π_{Select} multiplies $\mathsf{DReLU}(x)$ with y, producing a $\binom{3}{3}$-sharing that requires resharing and verification. To avoid additional verification overhead, we replace the multiplication by $[[1]]$ in Lines 23 and 25 of Algorithm 2 with $[[y]]$, and compute the MAC of y using a semi-honest multiplication. This is verified via $\mathcal{F}_{\mathsf{MacCheck}}$, eliminating additional verification for resharing. The full protocol is detailed in Algorithm 4 in Appendix C.1 in [7].

5 Maliciously Secure Non-linear Protocols

We introduce a maliciously secure 3PC protocol for lookup tables and integrate it with our comparison protocol to realize constant-round 3PC protocols for inverse and reciprocal square root. These serve as building blocks for implementing non-linear functions in transformers, including GELU, Softmax, and LayerNorm, under the malicious 3PC model. Protocol dependencies are shown in Figure 4 (see Appendix C in [7]). The full security and complexity analysis of the protocol is presented in Appendix D in [7].

5.1 Lookup Table

A lookup table (LUT) maps input values to precomputed outputs, enabling fast retrieval for non-linear functions. Given a $\binom{3}{2}$-shared input $\langle x\rangle$ and a public table $\vec{T}$, the goal of Π_{LUT} is to securely return $\langle\vec{T}[x]\rangle$ without leaking x. A standard approach is to encode x as a one-hot vector $\vec{u} \in \{0,1\}^n$ by comparing it with all indices. The desired output is then obtained via an inner product $\langle\vec{u}, \vec{T}\rangle$.

We design a communication-efficient, maliciously secure Π_{LUT} protocol using verifiable distributed point functions (VDPFs) and

Algorithm 3 Inverse (Π_{Inv})

Input: $\binom{3}{2}$-shared $\langle x\rangle$, scaling base b, fixed-point precision f, table $\vec{T}_{\mathsf{exp}}$ and table $\vec{T}_{\mathsf{inv}}$, where the size of $\vec{T}_{\mathsf{exp}}$ is $n = \lceil \log_b 2^{2f} - 1 \rceil$.
Output: $\binom{3}{2}$-shared $\langle z\rangle$ where $z = 1/x$.

1: **for** $i = 1$ to n **do** in parallel
2: $\quad \langle \vec{d}[i]\rangle \leftarrow \Pi_{\mathsf{DReLU}}(\langle x\rangle - b^i)$
3: **end for**
4: $\langle k\rangle = \sum_{i=0}^{n-1} \langle d[i]\rangle$
5: $\langle t\rangle \leftarrow \Pi_{\mathsf{LUT}}(\langle k+1\rangle, \vec{T}_{\mathsf{exp}})$
6: $\langle q\rangle \leftarrow \mathcal{F}_{\mathsf{mul}}^{\langle\cdot\rangle}(\langle x\rangle, \langle t\rangle)$
7: $\langle w\rangle \leftarrow \Pi_{\mathsf{LUT}}(\langle q - 1/b\rangle, \vec{T}_{\mathsf{inv}})$
8: **return** $\langle z\rangle \leftarrow \mathcal{F}_{\mathsf{mul}}^{\langle\cdot\rangle}(\langle t\rangle, \langle w\rangle)$.

MAC-based consistency checks. Each party samples a random index r and generates VDPF keys at position r, forming a secret-shared one-hot vector via full-domain DPF expansion. To realign this vector to the input x, parties compute masked offsets $x - r$ and apply index rotation. A MAC check ensures correctness: parties jointly sample a MAC key, compute MAC tags on the output, and perform a final consistency verification. Similar to Π_{DReLU}, our protocol guarantees malicious security with minimal rounds and low communication overhead by leveraging compact DPF-based components. The full protocol is provided in Algorithm 5 (see Appendix C.2 in [7]).

5.2 Inverse and Reciprocal Square Root

A straightforward approach to securely compute the inverse and reciprocal square root is to apply the secure LUT protocol Π_{LUT}. However, under malicious security, using a full-domain LUT of size 2^ℓ over $\mathbb{Z}_{2^\ell}$ incurs prohibitive overhead. To address this, we introduce a general *domain reduction* strategy that significantly reduces the LUT size while preserving accuracy. Given a positive input x, we select a scaling base $b \in \mathbb{Z}_{2^\ell}$ and determine the interval index k such that $b^k \le x < b^{k+1}$. By scaling x with $b^{-(k+1)}$, its value is normalized to the interval $[1/b, 1)$, allowing us to focus the LUT on a compact subdomain. With fractional bits of precision f, we can construct a lookup table of size $(1 - 1/b)\cdot 2^f$, which enables efficient evaluation within this restricted domain.

To determine the scaling factor $b^{-(k+1)}$, we apply Π_{DReLU} to compare $\langle x\rangle$ with thresholds $b^1, b^2, \ldots, b^n$, yielding a binary vector $\langle \vec{d}\rangle$ with $\vec{d}[i] = 1$ if $x \ge b^i$, and 0 otherwise, where the choice of n will be discussed later. When $x \in [b^k, b^{k+1})$, the first k elements of $\vec{d}$ are 1 and the rest are 0, allowing the interval index k (bounded within $[0, n]$) to be computed as $k = \sum_i \vec{d}[i]$. The corresponding scaling factor $b^{-(k+1)}$ is then efficiently retrieved from a compact auxiliary lookup table.

To ensure correct domain reduction over the ring $\mathbb{Z}_{2^\ell}$, we require that the inverse of the scaling factor, $(b^{k+1}/2^f)^{-1}\cdot 2^f$, is valid in $\mathbb{Z}_{2^\ell}$, i.e., it must be at least 1. This constraint leads to two implications: (i) the interval index must satisfy $k \le \log_b 2^{2f} - 1$; (ii) the input value x must be constrained to the interval $[0, 2^{2f}]$ to prevent incorrect computations. Fortunately, as shown in Table 10 (see Appendix E in [7]), the input distributions of transformer layers typically fall within this range, making the design practical for real-world inference. Under this constraint, at most $\lceil \log_b 2^{2f} - 1\rceil$ invocations of Π_{DReLU} are required, i.e., $n = \lceil \log_b 2^{2f} - 1\rceil$. Moreover, since each invocation corresponds to x shifted by a constant, all evaluations can be performed in parallel using shared keys, which significantly enhances computational efficiency.

For the inverse, $\frac{1}{x}$ is transformed into $b^{-(k+1)} \cdot \frac{1}{x\cdot b^{-(k+1)}}$. The detailed steps of the secure inverse protocol (Π_{Inv}) is shown in Algorithms 3. In this protocol, the following lookup tables are used:

- $\vec{T}_{\mathsf{exp}}[i] = \lfloor (b^i/2^f)^{-1}\cdot 2^f \rfloor$ for $i \in [0, \lceil \log_b 2^{2f} - 1\rceil]$
- $\vec{T}_{\mathsf{inv}}[i] = \lfloor (i/2^f + 1/b)^{-1}\cdot 2^f \rfloor$ for $i \in [0, (1 - 1/b)\cdot 2^f]$

Here, f denotes the number of fractional bits in the fixed-point representation, which determines computation precision. The reciprocal square root $\frac{1}{\sqrt{x}}$ is similarly reformulated as $\sqrt{b^{-(k+1)}}\cdot \frac{1}{\sqrt{x\cdot b^{-(k+1)}}}$. Due to space constraints, the secure reciprocal square root protocol (Π_{Rsqrt}) is provided in Algorithm 6, Appendix C.3 in [7]. We also refer the reader to Appendix F in the full version of this work [7] for a detailed analysis of how the scale base affects the precision of inverse and rsqrt protocols, which guides our parameter selection in secure inference.

5.3 Other Non-linear Functions

Combining our secure comparison and lookup table protocols enables the construction of maliciously secure and efficient 3PC protocols for GELU, Softmax, and LayerNorm. The formulated algorithms are deferred to Appendix C in [7].

GELU. The GELU function is defined as $\mathrm{GELU}(x) = 0.5x(1 + \tanh(\sqrt{2/\pi}(x + 0.044715x^3)))$. In semi-honest settings, secure GELU is often implemented using segmented functions or approximate polynomial fitting [34, 40], which require multiple invocations of secure multiplication and comparison protocols. However, these approaches become prohibitively expensive under malicious security. Inspired by SIGMA [19], we adopt a range-restricted lookup table construction to evaluate GELU efficiently in the malicious 3PC setting, implemented using our Π_{LUT} protocol.

Softmax. For a vector $\vec{x}$ of dimension k, let $x_{\max} = \max(x_0, x_1, \ldots, x_{k-1})$. The softmax function returns a vector $\vec{y}$ such that $y[i] = e^{x[i]-x_{\max}}/\sum_{j=0}^{k-1} e^{x[j]-x_{\max}}, \quad$ for $i \in [0, k-1]$. This computation consists of three main components: (i) maximum selection, (ii) exponentiation, and (iii) division. The maximum $x_{\max}$ is computed using $\lceil \log_2 k\rceil$ invocations of the Π_{Select} protocol. For exponentiation, we adopt the approximation method proposed in BOLT [40], implemented using our Π_{select} and Π_{LUT} protocols. Finally, the division is performed by invoking Π_{Inv} and $\mathcal{F}_{\mathsf{mul}}^{\langle\cdot\rangle}$.

LayerNorm. For a vector $\vec{x}$ of dimension k, LayerNorm outputs a vector $\vec{y}$ such that $y[i] = \gamma\cdot(x[i] - m)/\sqrt{v} + \beta$, where $m = \frac{1}{k}\sum_{j=0}^{k-1} x[j]$ is the mean and $v = \frac{1}{k}\sum_{j=0}^{k-1}(x[j] - m)^2$ is the variance. The parameters γ and β denote the weight and bias, respectively. Since the computation involves multiplication, and reciprocal square root, secure LayerNorm can be efficiently implemented by invoking Π_{Rsqrt} and $\mathcal{F}_{\mathsf{mul}}^{\langle\cdot\rangle}$.

6 Secure Transformer Inference with Modulus Conversion

Transformer-based models consist of sequential linear and non-linear layers, each connected with appropriately sized dimensions. Mosformer develops a suite of secure protocols for these layers, enabling the construction of arbitrary transformer architectures such as BERT and GPT. For linear layers, Mosformer employs Falcon's protocols [49], which achieve state-of-the-art performance under the three-party malicious security model. For non-linear layers, it utilizes the secure protocols introduced in Section 5.

In secure transformer inference, real-valued data is typically scaled and quantized into fixed-point integers over a ring $\mathbb{Z}_{2^\ell}$ to support efficient and secure computation. A fixed-point format with f bits of precision over $\mathbb{Z}_{2^\ell}$ can represent values in the range $[-2^{\ell-1-f}, 2^{\ell-1-f})$. However, the numerical requirements of different operations vary significantly. For example, multiplication consumes precision bits and reduces the effective representable range, whereas comparison primarily relies on the most significant bits and are largely insensitive to the underlying precision.

Maintaining a uniform ring throughout secure inference, as adopted in frameworks like SHAFT [22], PUMA [13], and MPC-Former [27], ensures correctness but incurs significant inefficiencies. In particular, using a fixed 64-bit representation introduces unnecessary computational and communication overhead. To mitigate this, Mosformer adopts an operation-aware modulus conversion mechanism that dynamically adjusts ring size and precision based on operation characteristics. As illustrated in Figure 5 (see Appendix E in [7]), Mosformer assigns different ring and precision settings to different types of operations: (1) Linear operations: $\mathbb{Z}_{2^{64}}$, 16-bit precision. (2) Softmax and reciprocal square root: $\mathbb{Z}_{2^{32}}$, 12-bit precision. (3) ReLU and GELU activations: $\mathbb{Z}_{2^{16}}$, 6-bit precision. The configurations, guided by transformer value ranges and protocol properties, use operation-aware precision to reduce computation and communication costs while preserving accuracy. Appendix E details the chosen ring and precision settings.

To perform the aforementioned modulus conversion between different rings, we introduce two core building blocks: downcast and upcast, which enable secure modulus conversion under the malicious secure three-party setting. Downcast converts values from a larger to a smaller fixed-point ring (e.g., from $\mathbb{Z}_{2^{64}}$ to $\mathbb{Z}_{2^{16}}$), while upcast performs the reverse. These protocols ensure secure and efficient transitions between computation layers with heterogeneous precision settings. Due to space limitations, full protocol details are presented in Appendix C.6 in [7].

7 Evaluations

7.1 Implementation and Experimental Setup

Testbed Environment. Mosformer is implemented in C++ with OpenMP for multi-threaded matrix multiplication. The source code is available at https://github.com/XidianNSS/Mosformer. All experiments are conducted on three servers running Ubuntu 24.04, each equipped with an AMD Ryzen 9 9950X 16-core processor and 128 GB of RAM. To emulate various network conditions, the Linux `tc` tool is used to control the bandwidth between servers. The default network configurations follow those used in Bumblebee [34], including a local-area network (LAN) setting with 1 Gbps bandwidth and 0.5 ms latency, and a wide-area network (WAN) setting with 400 Mbps bandwidth and 4 ms latency. In our experiments, we used 16 threads for matrix multiplication to accelerate the most compute-intensive operations, while the rest of the computation was executed in a single-threaded manner.

Table 4: Online Performance of Π_{DReLU}, Π_{Inv}, and Π_{Rsqrt} Compared to SOTA Malicious 3PC Schemes: Falcon (PETS'21), Li et al. (USENIX Security'23), and Privformer (EuroS&P'23)

DReLU					
#Inputs	Scheme*	LAN (s)	WAN (s)	Comm. (MB)	#Rounds
10^4	Falcon [49]	0.052	0.157	4.416	29
	Li et al. [28]	0.165	1.125	7.593	460
	Ours	0.021	0.081	1.144	18
10^5	Falcon [49]	0.482	1.116	44.155	29
	Li et al. [28]	0.501	2.528	75.934	460
	Ours	0.162	0.422	11.444	18
10^6	Falcon [49]	5.078	11.322	441.551	29
	Li et al. [28]	3.820	16.492	759.337	460
	Ours	1.992	4.949	114.441	18
Inverse					
#Inputs	Scheme*	LAN	WAN	Comm.	#Rounds
10^2	Falcon [49]	0.022	0.291	0.226	171
	Ours	0.006	0.084	0.051	42
10^4	Falcon [49]	0.28	0.855	22.631	171
	Ours	0.12	0.303	5.112	42
Reciprocal Square Root					
#Inputs	Scheme*	LAN	WAN	Comm.	#Rounds
10^2	Privformer [1]	0.034	0.488	0.254	281
	Ours	0.006	0.085	0.056	42
10^4	Privformer [1]	0.339	1.149	25.434	281
	Ours	0.124	0.315	5.646	42

*We evaluate all schemes, including our own, under a single-threaded setting.

Concrete Parameters. All secure computations are performed over secret-shared values under a mixed-modulus setting. For linear operations such as multiplication, values are represented as integers in the ring $\mathbb{Z}_{2^{64}}$ with 16-bit fixed-point precision. For softmax and reciprocal square root, we use a secret-sharing ring of $\mathbb{Z}_{2^{32}}$ with fixed-point precision $f = 12$. For ReLU and GELU within feed-forward layers, computations are executed over $\mathbb{Z}_{2^{16}}$ with $f = 6$. The security parameter for VDPF is set to $\lambda = 128$. The scale base b used in Π_{Inv} and Π_{Rsqrt} is set to 256, as detailed in Appendix F [7].

Models & Datasets. We evaluate our scheme on 3 transformer models, including the vanilla transformer [46], BERT-base [12], and GPT2-base [9]. All models are obtained from publicly available sources, with detailed model parameters provided in Table 11 of Appendix G in [7]. We evaluate BERT-base on the QNLI, RTE, and STS-B tasks from the GLUE benchmark [50]. QNLI and RTE are classification tasks evaluated using accuracy, whereas STS-B is a regression task evaluated using Pearson and Spearman correlation to assess sentence similarity. For GPT2-base, we evaluate language modeling performance on the WikiText-103 dataset [36], using perplexity as the evaluation metric.

7.2 Microbenchmarks

7.2.1 Baselines. We evaluate our DReLU, LUT, inverse, reciprocal square root, and GELU protocols over $\mathbb{Z}_{2^{32}}$ with 12-bit precision.

Falcon [49], Li et al. [28], and Pika [47] are state-of-the-art maliciously secure 3PC frameworks. Falcon implements a DReLU protocol via random masking and an inverse protocol that leaks the input range. Li et al. propose a bit-level comparison protocol using optimized Boolean circuits, which outperforms MP-SPDZ [23], Fantastic Four [10], and SWIFT [25]. We therefore exclude these prior frameworks from our comparison. Privformer [1] builds a maliciously secure reciprocal square root on Falcon's inverse protocol. Pika [47] presents a LUT protocol that achieves malicious security. While its design adopts a different input-output sharing format ((2,2)-sharing), it remains closely related to our protocol, which is based on (3,2)-sharing. To the best of our knowledge, no maliciously secure 3PC protocol for GELU is known; we defer the evaluation of our Π_{GELU} to Appendix H.1 in the full version [7].

7.2.2 Benchmarking Our Sub-Protocols. Table 4 presents a comprehensive comparison of the online performance of our sub-protocols Π_{DReLU}, Π_{Inv}, and Π_{Rsqrt} against state-of-the-art malicious 3PC schemes, including Falcon [49], Li et al. [28], and Privformer [1]. We observe that our sub-protocols consistently outperform all state-of-the-art baselines. For DReLU, our protocol achieves up to 7.9× LAN and 13.9× WAN speedup over Li et al., and 2.6 − 3.5× over Falcon across all input scales. It also reduces communication cost by up to 6.6× and rounds by 25× compared to Li et al.'s protocol. For inverse, we observe 2.3–3.7× improvements in runtime, a 4.4× reduction in communication, and a 4.1× reduction in the number of rounds. Our Rsqrt protocol achieves up to 5.7× speedup and 4.5× lower communication over Privformer. These results highlight the efficiency and scalability of our maliciously secure protocols.

Table 5 compares the performance of our LUT protocol (Π_{LUT}) with that of Pika [47] across varying input sizes, evaluating runtime under LAN and WAN settings and communication costs in both offline and online phases. While our LUT protocol is comparable to Pika [47] in online performance, it achieves substantial improvements in the offline phase, yielding up to 207.8× speedup in LAN settings and 85.7× in WAN settings for 10^5 inputs. This performance gain is primarily attributed to the difference in key validation domains: our protocol validates keys over $\mathbb{Z}_{2^\ell}$, whereas Pika performs validation over the larger ring $\mathbb{Z}_{2^{\ell+t}}$, where t is its security parameter. As a result, the overall complexity of our protocol is $O(2^\ell)$, in contrast to Pika's $O(2^{\ell+t})$. In terms of communication costs, our protocol incurs lower overhead in the offline phase and moderately higher overhead in the online phase, which represents a reasonable trade-off given the overall performance benefits. It is worth noting that the two schemes operate under different system settings. Our protocol assumes a symmetric 3PC model, while Pika is based on an asymmetric $(2+1)$-PC model. Accordingly, they adopt different input-output secret sharing formats: our protocol uses $(3,2)$-sharing, whereas Pika employs $(2,2)$-sharing. This fundamental difference prevents Pika's LUT protocol from being directly applied in our setting.

7.2.3 Maliciously Secure CNN Inference Using Π_{DReLU}. To demonstrate the generality and superiority of our comparison protocol, we evaluate secure CNN inference based on Π_{DReLU} on AlexNet and ResNet50. In our evaluation, we set the batch size b to 128 for AlexNet and 1 for ResNet50. The input tensor dimensions (c, h, w) represent the number of channels, height, and width, respectively, with values of $(3, 33, 33)$ for AlexNet and $(3, 224, 224)$ for ResNet50. As shown in Table 6, our approach achieves up to 5.5× speedup, 6.2× reduction in communication, and significantly fewer rounds compared to state-of-the-art maliciously secure baselines, including Falcon (PETS'21) and the work of Li et al. (USENIX Security'23). We further observe that the performance gap between Falcon and our work is smaller than that between Falcon and Li et al., indirectly reflecting Falcon's strong baseline performance. These results highlight the efficiency and scalability of our Π_{DReLU}-based design under malicious security.

Table 5: Comparison of Π_{LUT} with that of Pika (PETS'22) in terms of runtime (s) and communication costs (MB)

#Inputs	Scheme*	LAN (s)		WAN (s)		Comm. (MB)	
		Offline	Online	Offline	Online	Offline	Online
10^3	Pika [47]	2.776	0.007	4.217	0.032	0.311	0.023
	Ours	0.013	0.007	0.034	0.037	0.210	0.053
10^4	Pika [47]	28.143	0.032	34.298	0.058	3.108	0.229
	Ours	0.143	0.038	0.368	0.071	2.098	0.534
10^5	Pika [47]	284.865	0.288	299.725	0.363	31.081	2.289
	Ours	1.371	0.313	3.497	0.405	20.981	5.341

*We evaluate both Pika's protocol and ours using a single thread.

Table 6: Comparison of Secure CNN Inference based on Π_{DReLU} with Falcon (PETS'21) and the work of Li et al. (USENIX Security'23) in terms of online runtime (s) and online communication costs (GB)

Model	Batch Size	Input Size (c, h, w)*	Scheme†	Time	Comm.	#Rounds
AlexNet	128	(3, 33, 33)	Falcon [49]	92.37	3.23	1032
			Ours	36.72	1.25	232
ResNet50	1	(3, 224, 224)	Li et al. [28]	63.33	27.19	–
			Ours	11.44	4.40	1235

*(c, h, w) represents the number of channels, height, and width of an input tensor.
†Li et al. [28] perform secure CNN inference using 32 threads, whereas Falcon [49] and our scheme are evaluated with 16 threads.

7.3 Evaluation on the Vanilla Transformer

7.3.1 Baselines. Privformer is the only existing work supporting secure transformer inference under the same model as Mosformer, i.e., a three-party malicious setting with honest majority. However, it is limited to the vanilla transformer architecture proposed by Vaswani et al. [46], which includes multi-head attention (MHA), masked MHA, feed-forward networks (FFN), and LayerNorm, due to limited support for secure operations. As source code is unavailable, we re-implemented Privformer from its paper for a fair comparison.

For a broader comparison, we also evaluate Falcon [49], a representative three-party maliciously secure framework for CNN inference. As shown in Section 7.2, Falcon outperforms the work of Li et al. (USENIX Security'23). Since Falcon does not support the Softmax operation, we extend it by integrating CrypTen's exponential function protocol [24] with Falcon's existing max and division operations, resulting in a maliciously secure Softmax implementation. We refer to this extended version as Falcon+ in our evaluation.

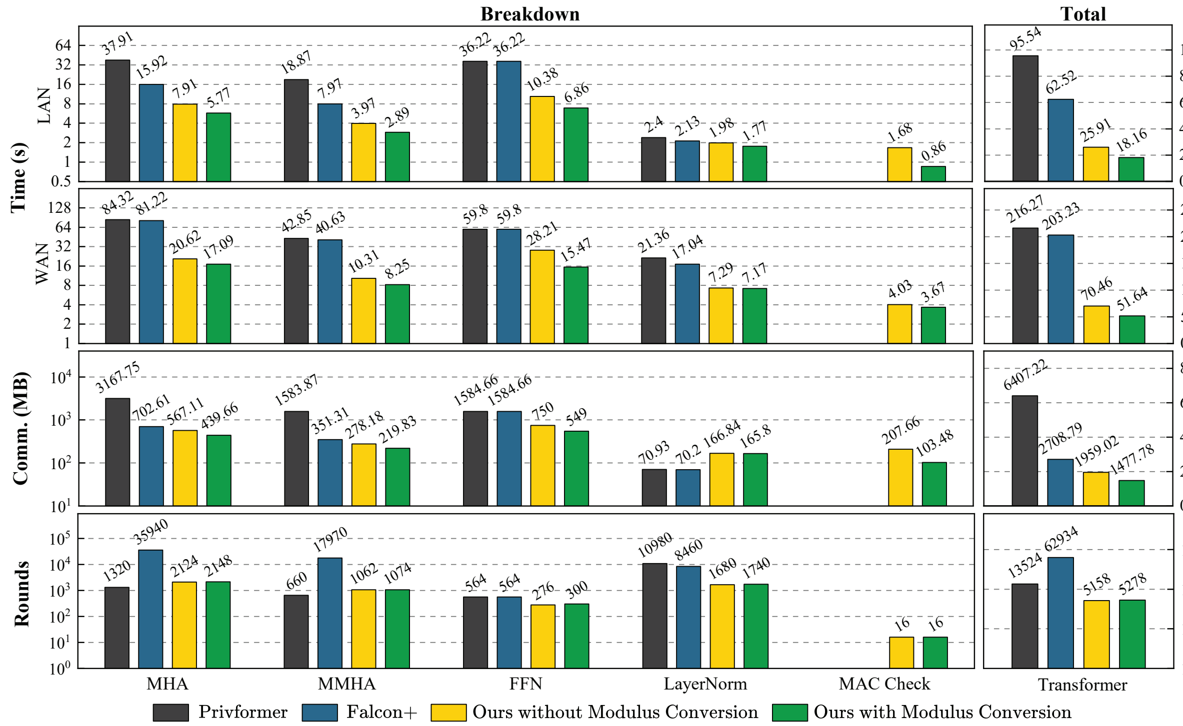


Figure 2: Online Performance Comparison and Breakdown of Secure Transformer Inference (All schemes use 16 threads; the evaluated vanilla transformer architecture consists of multi-head attention (MHA), masked MHA (MMHA), feed-forward networks (FFN), and LayerNorm.)

7.3.2 Overall Performance. We evaluate Mosformer with and without modulus conversion (M.C.) under both LAN and WAN settings, as shown in Figure 2. Compared to Privformer, Mosformer with M.C. achieves a 5.3× speedup in LAN and 4.2× in WAN. Compared to Falcon, the speedup reaches 3.4× in LAN and 3.9× in WAN. The corresponding communication cost is reduced by 4.3× and 1.8×, respectively, while the number of communication rounds is reduced by 2.6× and 11.9×, respectively.

7.3.3 Modulus Conversion. Modulus conversion reduces computation time and communication costs across all Mosformer components. Although it slightly increases communication rounds by 2.3%, it achieves substantial gains, cutting runtime by 30% in LAN and 27% in WAN and lowering communication cost by 25%. These improvements mainly result from reduced overhead in non-linear operations such as Softmax, ReLU, and reciprocal square root, which outweigh the added cost of modulus conversion.

7.3.4 Component-Level Performance. We provide a detailed analysis of Mosformer's implementation of attention, FFN, and LayerNorm in Appendix H.2, along with the deferred MAC check optimization (see the full version [7] for more details). Built upon our efficient non-linear protocols, Mosformer achieves $2-6.6\times$ speedup and significantly reduces communication and rounds compared to Privformer and Falcon+, demonstrating practical efficiency under malicious security.

7.4 Evaluation on BERT and GPT

7.4.1 Accuracy. Table 7 presents an accuracy comparison of inference in the plaintext and ciphertext domains across several secure frameworks, evaluated on the GLUE benchmark for BERT-base and the WikiText-103 dataset for GPT2-base. We report the percentage of accuracy drop (Acc. drop (%)) to quantify the utility degradation introduced by secure inference. For BERT-base, most frameworks exhibit limited accuracy degradation across the QNLI, RTE, and STS-B tasks. Our scheme achieves comparable or better performance than prior works. Specifically, on the STS-B task, Mosformer without modulus conversion (w/o M.C.) incurs only a 0.9% accuracy drop, outperforming BOLT (1.3%). Enabling modulus conversion (w/ M.C.) slightly increases the drop to 1.0%. For QNLI and RTE, Mosformer (w/o M.C.) incurs drops of 0.4% and 0.6%, respectively, matching or surpassing prior frameworks such as BOLT, BumbleBee, and SHAFT. Even with modulus conversion, the degradation remains modest (up to 1.7%), reflecting a reasonable trade-off between accuracy and computational efficiency. For GPT2-base, we evaluate utility via perplexity on WikiText-103, where lower is better. Mosformer shows strong results: perplexity rises only 0.3% without modulus conversion and 2.4% with it, clearly outperforming Ditto (5.3%).

These results confirm that Mosformer effectively preserves model utility under secure inference. The observed accuracy degradation without modulus conversion remains limited, with all tasks exhibiting losses below 1%. This is primarily due to the use of fixed-point

Table 7: Accuracy Comparison of Secure Inference Frameworks on the GLUE Benchmark for BERT and WikiText-103 for GPT2

Framework	Method	BERT-base			GPT2-base
		QNLI	RTE	STS-B	WikiText-103*
BOLT	plaintext	–	69.7	89.6	–
	ciphertext	–	69.3	88.4	–
	Acc. drop (%)	–	**0.6**	**1.3**	–
BumbelBee	plaintext	90.3	70.0	–	–
	ciphertext	90.2	70.0	–	–
	Acc. drop (%)	**0.1**	**0**	–	–
SHAFT	plaintext	90.7	–	–	–
	ciphertext	90.4	–	–	–
	Acc. drop (%)	**0.4**	–	–	–
PUMA	plaintext	91.6	68.6	–	12.3
	ciphertext	91.4	68.4	–	12.3
	Acc. drop (%)	**0.2**	**0.3**	–	0
Ditto	plaintext	91.6	68.6	–	12.3
	ciphertext	91.8	67.8	–	12.9
	Acc. drop (%)	**0.2**	**1.2**	–	**5.3**
Ours	plaintext	90.6	69.7	86.1	37.5
	w/ M.C.†	89.1	68.6	87.0	38.4
	w/o M.C.†	90.2	69.3	85.3	37.6
	Acc. drop (%) w/ M.C	**1.7**	**1.6**	**1.0**	**2.4**
	Acc. drop (%) w/o M.C	**0.4**	**0.6**	**0.9**	**0.3**

*A lower score on this metric indicates better performance.
† w/o M.C. and w/ M.C. denote Mosformer without and with modulus conversion, respectively.

arithmetic in secure computation, which replaces floating-point operations in plaintext inference and inherently introduces some precision loss. Furthermore, Mosformer leverages table lookup methods to securely evaluate nonlinear functions such as Inverse, GELU, and Softmax, which may introduce additional minor computational errors. When operation-aware modulus conversion is applied, the additional accuracy loss is primarily attributed to the further reduction in bit precision. For example, the secure computation of Softmax was originally performed over $\mathbb{Z}_{2^{64}}$ with 16-bit fixed-point precision. After applying modulus conversion, the computation is conducted over $\mathbb{Z}_{2^{32}}$ with reduced 12-bit precision, resulting in more noticeable approximation errors due to the coarser numerical granularity. Nevertheless, the maximum accuracy degradation remains within 2.4%, demonstrating that Mosformer strikes a practical balance between accuracy and computational efficiency for secure inference of large transformers.

Interestingly, we observe that Mosformer with M.C. outperforms plaintext inference on the STS-B task. Since STS-B is evaluated using Pearson and Spearman correlations, which focus on ranking consistency, minor precision errors introduced by secure computation may inadvertently improve the ranking alignment. Similar effects have been reported in Ditto [51] on the QNLI task. In addition, different lookup table configurations impact the accuracy and runtime of secure transformer inference. We provide corresponding ablation studies in Appendix H.3 in [7]. Moreover, different hyperparameter selections related to modulus conversion may lead to varying degrees of utility loss. To validate the effectiveness of our design choices, we conduct ablation studies on the hyperparameter selection for different operations under modulus conversion. The results confirm the soundness of our selected parameters, with detailed analysis provided in Appendix H.4 in [7].

7.4.2 End-to-End Performance. Mosformer is the first maliciously secure 3PC inference framework supporting large-scale transformers such as BERT and GPT. To evaluate its effectiveness, we compare it with state-of-the-art semi-honest frameworks and also implement a semi-honest Mosformer by removing malicious-verification steps. Specifically, BOLT [40], BumbleBee [34], and SHAFT [22] are 2PC semi-honest frameworks, while PUMA [13] and Ditto [51] are 3PC semi-honest frameworks.

Online Performance. Table 8 and Table 9 compare the online inference runtime and communication cost of Mosformer under both semi-honest and malicious security models against state-of-the-art 2PC and 3PC frameworks. Results for BumbleBee [34] and SHAFT [22] are obtained by re-running their open-source implementations in our evaluation environment, while other baseline results are taken directly from their respective papers.

As shown in Table 8, our maliciously secure protocol outperforms existing semi-honest 2PC frameworks, achieving up to 9.0× speedup in LAN, 7.1× in WAN, and 12.9× lower communication on BERT-base. On GPT2-base, it yields 2.5–3.2× LAN speedup, 1.9–2.5× WAN speedup, and up to 2.7× reduction in communication. These results indicate that our maliciously secure protocol provides stronger security guarantees while maintaining competitive online efficiency compared to existing semi-honest 2PC frameworks.

Table 9 further shows that Mosformer also outperforms state-of-the-art 3PC frameworks PUMA and Ditto. Our semi-honest variant achieves up to 4.0× speedup and 9.2× lower communication on BERT-base, and up to 5.1× and 17.4× reductions, respectively, on GPT2-base. Notably, even under malicious security, Mosformer demonstrates comparable or superior online performance relative to prior semi-honest 3PC baselines, suggesting its potential for practical deployment.

Offline Performance. Table 8 and Table 9 present a comprehensive comparison of offline performance between Mosformer and state-of-the-art semi-honest 2PC and 3PC frameworks. We observe that most existing frameworks, such as BOLT, BumbleBee, PUMA, and Ditto, do not involve offline costs, suggesting that they are either designed primarily for online-only evaluation or do not explicitly distinguish the precomputation phase. In contrast, Mosformer incurs a non-trivial offline phase due to its precomputation-based design, which relies on advanced primitives such as VDPFs. For example, as shown in Table 8, on GPT2-base, Mosformer (semi-honest) incurs 160.43s of offline runtime over LAN with 20.18 GB of communication. In the malicious setting, the offline cost increases to 273.83s and 33.67 GB. By comparison, SHAFT incurs a smaller offline overhead, with 20.08 seconds of runtime and 2.51 GB of communication over LAN.

These results indicate that Mosformer introduces additional offline overhead in exchange for significant improvements in online performance. Such a trade-off is well-suited for deployment scenarios where offline precomputation is feasible or can be amortized across multiple inferences. We also note that several prior works [18, 52] have proposed more efficient techniques for DPF key generation and compression. These techniques are orthogonal to our contributions and could be incorporated into our framework to further reduce offline costs. Exploring such optimizations constitutes a key direction for our future work.

Table 8: Comparisons of runtime and communication costs with SOTA 2PC works, including BOLT (Oakland'24), BumbleBee (NDSS'25), and SHAFT (NDSS'25)

Model	Framework	Security Model	Runtime (s) over LAN		Runtime (s) over WAN		Communication Cost (GB)	
			Online	Offline	Online	Offline	Online	Offline
BERT-base (128 input tokens)	BOLT	Semi-honest	533.4	0	1014	0	59.61	0
	BumbleBee	Semi-honest	184.86	0	292.41	0	6.40	0
	SHAFT	Semi-honest	171.31	39.36	437.03	100.76	10.46	4.92
	Mosformer (ours)	Semi-honest	20.09	323.28	49.74	827.60	1.15	40.42
	Mosformer (ours)	Malicious	59.47	547.66	143.02	1339.14	4.60	67.35
GPT2-base (64 input tokens)	BumbleBee	Semi-honest	123.91	0	189.62	0	2.77	0
	SHAFT	Semi-honest	96.31	20.08	244.56	51.41	5.76	2.51
	Mosformer (ours)	Semi-honest	9.70	160.43	23.25	413.28	0.45	20.18
	Mosformer (ours)	Malicious	39.26	273.83	98.89	672.57	3.07	33.67

*We evaluate our protocol using 16 threads for matrix multiplication, while the remaining components are executed in a single-threaded manner. BumbleBee and SHAFT are also evaluated with 16 threads, whereas BOLT is evaluated with 32 threads.

Table 9: Comparisons of runtime and communication costs with SOTA 3PC works, including PUMA [13] and Ditto [51]

Model	Framework*	Security Model	Runtime (s) over LAN†		Runtime (s) over WAN†		Communication Cost (GB)	
			Online	Offline	Online	Offline	Online	Offline
BERT-base (128 input tokens)	PUMA	Semi-honest	43.98	0	444.43	0	10.59	0
	Ditto	Semi-honest	30.58	0	303.5	0	4.35	0
	Mosformer (ours)	Semi-honest	10.88	63.79	93.11	832.48	1.15	40.42
	Mosformer (ours)	Malicious	25.63	109.46	226.02	1342.37	4.60	67.35
GPT2-base (64 input tokens)	PUMA	Semi-honest	38.33	0	357.65	0	7.82	0
	Ditto	Semi-honest	29.41	0	233.32	0	5.18	0
	Mosformer (ours)	Semi-honest	5.70	33.01	65.45	419.06	0.45	20.18
	Mosformer (ours)	Malicious	17.38	54.75	175.37	677.62	3.07	33.67

*We evaluate our protocol using 16 threads for matrix multiplication, while the remaining components are executed in a single-threaded manner. The thread configurations of PUMA and Ditto are not reported in their original papers.
†As Ditto is not open-source, we adopt its reported network settings (5 Gbps/0.4 ms LAN, 400 Mbps/40 ms WAN) for consistent evaluation.

8 Conclusion

We present Mosformer, the first maliciously secure 3PC inference framework for large-scale transformers such as BERT and GPT2. It employs constant-round secure comparison and lookup protocols via VDPF for efficient non-linear operations, and integrates secure modulus conversion to cut overhead while preserving accuracy. Compared to prior maliciously secure 3PC frameworks limited to vanilla transformer blocks, Mosformer achieves up to 5.3× speedup and 4.3× lower communication, while surpassing state-of-the-art semi-honest 2PC/3PC frameworks on BERT and GPT2 inference. Although our work makes meaningful progress toward secure and efficient transformer inference, running modern large language models under cryptographic settings remains impractical at present. Future work may focus on addressing such gaps by overcoming the underlying scalability and efficiency bottlenecks.

Acknowledgments

We thank all the anonymous reviewers for their comments and suggestions. This paper is supported by the National Natural Science Foundation of China (No. 62402358, No. 62220106004, No. 92267204), the National Key R&D Program of China (No. 2023YFB3107500), the Young Talent Fund of Association for Science and Technology in Shaanxi, China (No. 20240138), the Key R&D Program of Shandong Province of China (No. 2023CXPT056), the Open Topics from the Lion Rock Labs of Cyberspace Security (under the project #LRL24004), the Fundamental Research Funds for the Central Universities (No. ZDRC2202, ZYTS25081, KYFZ25005), the Xidian University Specially Funded Project for Interdisciplinary Exploration (No. TZJHF202502), the China Scholarship Council (CSC), and the Double First-Class Overseas Research Project of Xidian University.

References

[1] Yoshimasa Akimoto, Kazuto Fukuchi, Youhei Akimoto, and Jun Sakuma. 2023. Privformer: Privacy-preserving transformer with mpc. In *2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P)*. IEEE, 392–410.

[2] Jianli Bai, Xiangfu Song, Xiaowu Zhang, Qifan Wang, Shujie Cui, Ee-Chien Chang, and Giovanni Russello. 2023. Mostree: Malicious Secure Private Decision Tree Evaluation with Sublinear Communication. In *Proceedings of the 39th Annual Computer Security Applications Conference*. 799–813.

[3] Foteini Baldimtsi, Dimitrios Papadopoulos, Stavros Papadopoulos, Alessandra Scafuro, and Nikos Triandopoulos. 2017. Server-aided secure computation with off-line parties. In *Computer Security–ESORICS 2017: 22nd European Symposium on Research in Computer Security, Oslo, Norway, September 11-15, 2017, Proceedings, Part I 22*. Springer, 103–123.

[4] Elette Boyle, Niv Gilboa, and Yuval Ishai. 2015. Function secret sharing. In *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 337–367.

[5] Elette Boyle, Niv Gilboa, and Yuval Ishai. 2016. Function secret sharing: Improvements and extensions. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 1292–1303.

[6] Tianyu Chen, Hangbo Bao, Shaohan Huang, Li Dong, Binxing Jiao, Daxin Jiang, Haoyi Zhou, Jianxin Li, and Furu Wei. 2022. THE-X: Privacy-Preserving Transformer Inference with Homomorphic Encryption. In *Findings of the Association for Computational Linguistics: ACL 2022*. 3510–3520.
[7] Ke Cheng, Yuheng Xia, Anxiao Song, Jiaxuan Fu, Wenjie Qu, Yulong Shen, and Jiaheng Zhang. 2025. Mosformer: Maliciously Secure Three-Party Inference Framework for Large Transformers. Cryptology ePrint Archive, Paper 2025/1510. https://eprint.iacr.org/2025/1510
[8] Krzysztof Marcin Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Quincy Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. 2021. Rethinking Attention with Performers. In *International Conference on Learning Representations*.
[9] Vanya Cohen and Aaron Gokaslan. 2020. OpenGPT-2: Open language models and implications of generated text. *XRDS: Crossroads, The ACM Magazine for Students* 27, 1 (2020), 26–30.
[10] Anders Dalskov, Daniel Escudero, and Marcel Keller. 2021. Fantastic four: Honest-Majority Four-Party secure computation with malicious security. In *USENIX Security 21*. 2183–2200.
[11] Leo de Castro and Anitgoni Polychroniadou. 2022. Lightweight, maliciously secure verifiable function secret sharing. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 150–179.
[12] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*. 4171–4186.
[13] Ye Dong, Wen-jie Lu, Yancheng Zheng, Haoqi Wu, Derun Zhao, Jin Tan, Zhicong Huang, Cheng Hong, Tao Wei, and Wenguang Chen. 2023. Puma: Secure inference of llama-7b in five minutes. *arXiv preprint arXiv:2307.12533* (2023).
[14] Jiaxuan Fu, Ke Cheng, Yuheng Xia, Anxiao Song, Qianxing Li, and Yulong Shen. 2024. Private decision tree evaluation with malicious security via function secret sharing. In *ESORICS*. Springer, 310–330.
[15] Jun Furukawa, Yehuda Lindell, Ariel Nof, and Or Weinstein. 2017. High-throughput secure three-party computation for malicious adversaries and an honest majority. In *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 225–255.
[16] Craig Gentry. 2009. *A Fully Homomorphic Encryption Scheme*. Ph. D. Dissertation. Stanford University.
[17] Chao Guo, Ke Cheng, Jiaxuan Fu, Ruolu Fan, Zhao Chang, Zhiwei Zhang, and Anxiao Song. 2023. GFS-CNN: A GPU-friendly secure computation platform for convolutional neural networks. *Journal of Networking and Network Applications* 3, 2 (2023), 66–72.
[18] Xiaojie Guo, Kang Yang, Xiao Wang, Wenhao Zhang, Xiang Xie, Jiang Zhang, and Zheli Liu. 2023. Half-tree: Halving the cost of tree expansion in COT and DPF. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 330–362.
[19] Kanav Gupta, Neha Jawalkar, Ananta Mukherjee, Nishanth Chandran, Divya Gupta, Ashish Panwar, and Rahul Sharma. 2024. SIGMA: Secure GPT Inference with Function Secret Sharing. *Proceedings on Privacy Enhancing Technologies* (2024).
[20] Meng Hao, Hongwei Li, Hanxiao Chen, Pengzhi Xing, Guowen Xu, and Tianwei Zhang. 2022. Iron: Private inference on transformers. *Advances in Neural Information Processing Systems* 35 (2022), 15718–15731.
[21] Xiaoyang Hou, Jian Liu, Jingyu Li, Yuhan Li, Wen-jie Lu, Cheng Hong, and Kui Ren. 2023. Ciphergpt: Secure two-party gpt inference. *Cryptology ePrint Archive* (2023).
[22] Andes Y. L. Kei and Sherman S. M. Chow. 2025. SHAFT: Secure, Handy, Accurate, and Fast Transformer Inference. *NDSS* (2025).
[23] Marcel Keller. 2020. MP-SPDZ: A versatile framework for multi-party computation. In *Proceedings of the 2020 ACM SIGSAC conference on computer and communications security*. 1575–1590.
[24] Brian Knott, Shobha Venkataraman, Awni Hannun, Shubho Sengupta, Mark Ibrahim, and Laurens van der Maaten. 2021. Crypten: Secure multi-party computation meets machine learning. *Advances in Neural Information Processing Systems* 34 (2021), 4961–4973.
[25] Nishat Koti, Mahak Pancholi, Arpita Patra, and Ajith Suresh. 2021. SWIFT: Super-fast and robust Privacy-Preserving machine learning. In *USENIX Security 21*. 2651–2668.
[26] Nishant Kumar, Mayank Rathee, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. 2020. Cryptflow: Secure tensorflow inference. In *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 336–353.
[27] Dacheng Li and Rulin Shao. 2023. MPCFormer: Fast, performant and private transformer inference with MPC. *ICLR 2023* (2023).
[28] Yun Li, Yufei Duan, Zhicong Huang, Cheng Hong, Chao Zhang, and Yifan Song. 2023. Efficient 3PC for binary circuits with application to Maliciously-Secure DNN inference. In *USENIX Security 23*. 5377–5394.
[29] Zhengyi Li, Kang Yang, Jin Tan, Wen-jie Lu, Haoqi Wu, Xiao Wang, Yu Yu, Derun Zhao, Yancheng Zheng, Minyi Guo, et al. 2025. Nimbus: Secure and Efficient Two-Party Inference for Transformers. *Advances in Neural Information Processing Systems* 37 (2025), 21572–21600.
[30] Yehuda Lindell and Ariel Nof. 2017. A framework for constructing fast MPC over arithmetic circuits with malicious adversaries and an honest-majority. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 259–276.
[31] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. 2024. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437* (2024).
[32] Xiaoning Liu, Yifeng Zheng, Xingliang Yuan, and Xun Yi. 2022. Securely outsourcing neural network inference to the cloud with lightweight techniques. *IEEE Transactions on Dependable and Secure Computing* 20, 1 (2022), 620–636.
[33] Tao Lu, Haoyu Wang, Wenjie Qu, Zonghui Wang, Jinye He, Tianyang Tao, Wenzhi Chen, and Jiaheng Zhang. 2024. An efficient and extensible zero-knowledge proof framework for neural networks. *Cryptology ePrint Archive* (2024).
[34] Wen-jie Lu, Zhicong Huang, Zhen Gu, Jingyu Li, Jian Liu, Kui Ren, Cheng Hong, Tao Wei, and WenGuang Chen. 2025. BumbleBee: Secure Two-party Inference Framework for Large Transformers. *NDSS* (2025).
[35] Jinglong Luo, Yehong Zhang, Zhuo Zhang, Jiaqi Zhang, Xin Mu, Hui Wang, Yue Yu, and Zenglin Xu. 2024. SecFormer: Fast and Accurate Privacy-Preserving Inference for Transformer Models via SMPC. In *Findings of the Association for Computational Linguistics ACL 2024*. 13333–13348.
[36] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer Sentinel Mixture Models. arXiv:1609.07843 [cs.CL]
[37] Payman Mohassel and Peter Rindal. 2018. ABY3: A mixed protocol framework for machine learning. In *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*. 35–52.
[38] Payman Mohassel and Yupeng Zhang. 2017. Secureml: A system for scalable privacy-preserving machine learning. In *2017 IEEE symposium on security and privacy (SP)*. IEEE, 19–38.
[39] OpenAI. 2023. March 20 ChatGPT outage: Here's what happened. https://openai.com/blog/march-20-chatgpt-outage. Accessed: 2025-04-11.
[40] Qi Pang, Jinhao Zhu, Helen Möllering, Wenting Zheng, and Thomas Schneider. 2024. Bolt: Privacy-preserving, accurate and efficient inference for transformers. In *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 4753–4771.
[41] Wenjie Qu, Yijun Sun, Xuanming Liu, Tao Lu, Yanpei Guo, Kai Chen, and Jiaheng Zhang. 2025. zkGPT: An Efficient Non-interactive Zero-knowledge Proof Framework for LLM Inference. In *USENIX Security 25*.
[42] Wenjie Qu, Wengrui Zheng, Tianyang Tao, Dong Yin, Yanze Jiang, Zhihua Tian, Wei Zou, Jinyuan Jia, and Jiaheng Zhang. 2025. Provably Robust Multi-bit Watermarking for {AI-generated} Text. In *USENIX Security 25*. 201–220.
[43] Wenjie Qu, Yuguang Zhou, Wang Bo, Zheng Wengrui, Yuexin Li, Zinyuan Jia, and Jiaheng Zhang. 2025. RepoMark: A Code Usage Auditing Framework for Code Large Language Models. *arxiv.org/pdf/2508.21432* (2025).
[44] Deevashwer Rathee, Mayank Rathee, Nishant Kumar, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. 2020. Cryptflow2: Practical 2-party secure inference. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 325–342.
[45] Anxiao Song, Jiaxuan Fu, Xutong Mu, XingHui Zhu, and Ke Cheng. 2024. L-secnet: Towards secure and lightweight deep neural network inference. *Journal of Networking and Network Applications* 3, 4 (2024), 171–181.
[46] A Vaswani. 2017. Attention is all you need. *Advances in Neural Information Processing Systems* (2017).
[47] Sameer Wagh. 2022. Pika: Secure computation using function secret sharing over rings. *Proceedings on Privacy Enhancing Technologies* (2022).
[48] Sameer Wagh, Divya Gupta, and Nishanth Chandran. 2018. Securenn: Efficient and private neural network training. *Cryptology ePrint Archive* (2018).
[49] Sameer Wagh, Shruti Tople, Fabrice Benhamouda, Eyal Kushilevitz, Prateek Mittal, and Tal Rabin. 2021. Falcon: Honest-Majority Maliciously Secure Framework for Private Deep Learning. *Proceedings on Privacy Enhancing Technologies* (2021).
[50] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2019. Glue: A multi-task benchmark and analysis platform for natural language understanding. In *ICLR 2019*.
[51] Haoqi Wu, Wenjing Fang, Yancheng Zheng, Junming Ma, Jin Tan, and Lei Wang. 2024. Ditto: quantization-aware secure inference of transformers upon MPC. In *ICML*. 53346–53365.
[52] Pengzhi Xing, Hongwei Li, Meng Hao, Hanxiao Chen, Jia Hu, and Dongxiao Liu. 2025. Distributed Function Secret Sharing and Applications. *NDSS* (2025).
[53] Andrew C Yao. 1982. Protocols for secure computations. In *23rd annual symposium on foundations of computer science (sfcs 1982)*. IEEE, 160–164.
[54] Jiaheng Zhang. 2023. *Efficient Zero-Knowledge Proofs: Theory and Practice*. Ph. D. Dissertation. UC Berkeley.
[55] Jiaheng Zhang, Zhiyong Fang, Yupeng Zhang, and Dawn Song. 2020. Zero knowledge proofs for decision tree predictions and accuracy. In *CCS 20*. 2039–2053.
[56] Jiawen Zhang, Xinpeng Yang, Lipeng He, Kejia Chen, Wen-jie Lu, Yinghao Wang, Xiaoyang Hou, Jian Liu, Kui Ren, and Xiaohu Yang. 2025. Secure transformer inference made non-interactive. *NDSS* (2025).

A Runtime Percentage of Secure Comparison Protocol

Privformer [1] adopts the maliciously secure comparison protocol in Falcon [47], which constitutes a substantial portion of the total execution time, ranging from half in LAN to over 80% in WAN settings. (see Figure 3).

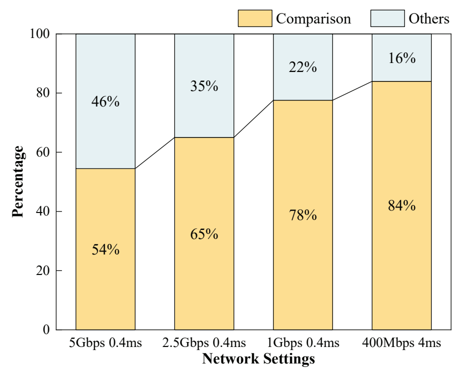


Figure 3: Runtime Percentage of Secure Comparison Protocol of Privformer under Different Network Settings

B Transformer

The transformer architecture typically consists of an input embedding layer followed by stacked encoder and decoder modules. Each encoder block typically begins with an attention mechanism, followed by a feed forward network and two normalization operations.

Token embeddings. The transformer represents natural language input as a sequence of tokens, each encoded as a d_{model}-dimensional vector. Tokens are mapped to embeddings using a token embedding matrix $W_e \in \mathbb{R}^{d_{model} \times N_v}$, where N_v denotes the vocabulary size.

Attention. The attention mechanism enables the model to assign different weights to parts of the input sequence, allowing it to focus on task-relevant information, particularly in long sequences. Self-attention is a specific form of attention that captures dependencies among elements within the same input sequence. Given input matrices (Q, K, V), self-attention is computed as:

$$\text{Attention}(Q, K, V) = \text{Softmax}(QK^T / \sqrt{d_{model}})V \tag{5}$$

where Q, K, and V denote the query, key, and value matrices, respectively.

Feed Forward Network (FFN). The feed-forward network consists of two fully connected layers and an activation function to capture nonlinear relationships in the sequence. Given an input matrix X, FFN is defined as:

$$\text{FFN}(X) = \text{GELU}(XW_1 + b_1)W_2 + b_2 \tag{6}$$

where W_1, W_2, b_1, b_2 are the weight and bias parameters of the two fully connected layers, respectively, and GELU is the Gaussian Error Linear Unit activation function.

Layer Normalization (LayerNorm). Layer normalization stabilizes training by normalizing activations within each layer. For an input vector $x \in \mathbb{R}^k$, with mean $m = \sum(x_i)/k$ and variance $v = (\sum(x_i - m)^2)/k$, the normalized output $y \in \mathbb{R}^k$ is given by:

$$y_i = \gamma \cdot \frac{x_i - m}{\sqrt{v}} + \beta = \gamma \cdot \frac{z_i}{\sqrt{\sum z_i^2 / k}} + \beta \tag{7}$$

where $z_i = x_i - m$, and $\gamma, \beta \in \mathbb{R}$ are learnable parameters.

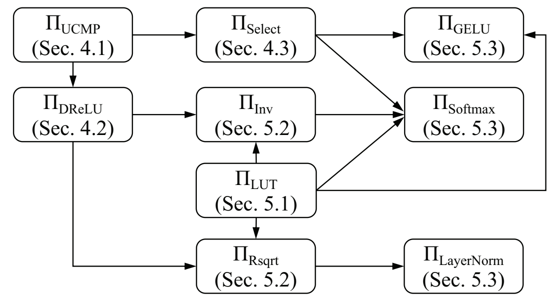


Figure 4: Protocols Dependencies

C Maliciously Secure 3PC Protocols

This section first presents the formal descriptions of our proposed maliciously secure 3PC protocols. The dependencies among these protocols are illustrated in Figure 4.

C.1 Select

Algorithm 4 shows a maliciously secure 3PC protocol (Π_{Select}) for Select functionality, where $\mathsf{Select}(x, y) = 1\{x \geq 0\} \cdot y$. Given $\binom{3}{2}$-shared values $\langle x \rangle$ and $\langle y \rangle$, the protocol Π_{Select} outputs $\langle y \rangle$ if $x \geq 0$, and $\langle 0 \rangle$ otherwise.

Algorithm 4 Secure Select Protol (Π_{Select})

Input: $\binom{3}{2}$-shared $\langle x \rangle$ and $\langle y \rangle$, where $x, y \in \mathbb{Z}_{2^\ell}$.
Output: $\binom{3}{2}$-shared $\langle v \rangle$ where $v = y$ when $x \geq 0$, otherwise 0.

The initial steps remain consistent with the first 18 steps of Π_{DReLU} and the party P_b has obtained $[\![\theta^{b+2}]\!]$ and $[\theta^{b+1}]\!]$.
The three parties jointly compute $\langle my \rangle \leftarrow \mathsf{Mul}(\langle y \rangle, \langle \alpha \rangle)$.
P_b and P_{b+1} ($b \in \mathbb{Z}_3$) do:
$[\![v_0]\!] = [\![y]\!]_{b+1} \cdot (1 - [\![\theta^{b+2}]\!])$, $[\![mv_0]\!] = [\![my]\!]_{b+1} \cdot (1 - [\![\theta^{b+2}]\!])$
P_b and P_{b+2} ($b \in \mathbb{Z}_3$) do:
$[\![v_1]\!] = [\![y]\!]_b \cdot (1 - [\![\theta^{b+1}]\!])$, $[\![mv_1]\!] = [\![my]\!]_b \cdot (1 - [\![\theta^{b+1}]\!])$
P_0, P_1, and P_2 collaboratively compute:
$[\![v]\!] = [\![v_0]\!] + [\![v_1]\!]$, $[\![mv]\!] = [\![mv_0]\!] + [\![mv_1]\!]$
$\langle v \rangle \leftarrow \mathsf{Reshare}([\![v]\!])$, $\langle mv \rangle \leftarrow \mathsf{Reshare}([\![mv]\!])$
Invoke $\mathcal{F}_{\mathsf{MacCheck}}$ on $\langle v \rangle$ and $\langle mv \rangle$. Abort if the check fails.
return $\langle v \rangle$.

C.2 Lookup Table

Algorithm 5 shows a maliciously secure 3PC protocol Π_{LUT} for LUT functionality. Given $\binom{3}{2}$-shared $\langle x \rangle$ and public table $\vec{T}$, where the size of the table is n and $\ell_t = \lceil \log_2 n \rceil$, the protocol Π_{LUT} outputs $\langle res \rangle$ where $res = \vec{T}[x]$.

Algorithm 5 Lookup Table (Π_{LUT})

Input: $\binom{3}{2}$-shared $\langle x\rangle$ and public table $\vec{T}$, where the size of the table is n and $\ell_t = \lceil \log_2 n \rceil$.
Output: $\binom{3}{2}$-shared $\langle res\rangle$ where $res = \vec{T}[x]$.
[Setup] Upon initialization, the party P_b ($b \in \mathbb{Z}_3$) does:
Randomly samples $r^b \leftarrow \mathbb{Z}_{2^{\ell_t}}$ and $[[r^b]]_0 \leftarrow \mathbb{Z}_{2^\ell}$.
Compute $[[r^b]]_1 = r^b - [[r^b]]_0$.
$(k_0^b, k_1^b) \leftarrow \mathsf{VDPF.Gen}(1^\lambda, r^b, 1)$.
P_b Send $([[r^b]]_0, k_0^b)$ to P_{b+2} and $([[r^b]]_1, k_1^b)$ to P_{b+1}
P_{b+1} and P_{b+2} jointly run the VDPF key verification protocol in [11] to check the well-formness of VDPF keys. If the check fails, abort.
P_{b+1} and P_{b+2} expand its VDPF key on domain $\mathbb{Z}_{2^{\ell_t}}$ to produce a $\binom{2}{2}$-shared vector $[[\vec{v}]]$. Then, P_{b+1} and P_{b+2} jointly compute:
$[[t]] = \sum_{i=0}^{2^{\ell_t}} [[\vec{v}[i]]]$
$[[s]] = [[r]] - \sum_{i=0}^{2^{\ell_t}} i \cdot [[\vec{v}[i]]]$
P_{b+1} and P_{b+2} open $[[t]], [[s]]$ then check if $t = 1$ and $s = 0$. If the check fails, abort.
The parties call $\langle \alpha\rangle \leftarrow \mathcal{F}_{rand}(\mathbb{Z}_{2^\ell})$ to share a MAC key $\alpha \in \mathbb{Z}_{2^\ell}$ and jointly generate a shared value $\langle 1\rangle = ([[1]]_b, [[1]]_{b+1})$.
[Evaluation] Upon receiving the shared input $\langle x\rangle$ and the lookup table $\vec{T}$, the party P_b ($b \in \mathbb{Z}_3$) does:
Set $\langle r^b\rangle = ([[r^b]]_0, [[r^b]]_1)$, $\langle r^{b+1}\rangle = (0, [[r^{b+1}]]_0)$, and $\langle r^{b+2}\rangle = ([[r^{b+2}]]_1, 0)$
for $i \in \{0, 1, 2\}$ **do**
  $\langle \hat{x}_i\rangle = \langle x\rangle - \langle r^i\rangle$
  Party P_{i+1} obtains $\hat{x}_i$ by invoking $\mathcal{F}_{\mathsf{recon}}(\langle \hat{x}_i\rangle, i+1)$
  Party P_{i+2} obtains $\hat{x}_i$ by invoking $\mathcal{F}_{\mathsf{recon}}(\langle \hat{x}_i\rangle, i+2)$
end for
$[[\vec{u}]]^{b+1} = [[\vec{v}]]^{b+1} \ggg \hat{x}_{b+1}$
$[[\vec{u}]]^{b+2} = [[\vec{v}]]^{b+2} \ggg \hat{x}_{b+1}$
$w_0^{b+1} = \sum_{i=0}^{2^{\ell_t}-1} [[\vec{u}[i]]]_0^{b+1} \cdot \vec{T}[i]$
$w_1^{b+2} = \sum_{i=0}^{2^{\ell_t}-1} [[\vec{u}[i]]]_1^{b+2} \cdot \vec{T}[i]$
The party P_b ($b \in \mathbb{Z}_3$) does:
$[[res^b]] = w_0^{b+1} \cdot [[1]]_b + w_1^{b+2} \cdot [[1]]_{b+1}$
$[[m_{res}^b]] = w_0^{b+1} \cdot [[\alpha]]_b + w_1^{b+2} \cdot [[\alpha]]_{b+1}$
$\langle res\rangle \leftarrow \mathsf{Reshare}([[res]])$
$\langle m_{res}\rangle \leftarrow \mathsf{Reshare}([[m_{res}]])$
Invoke $\mathcal{F}_{\mathsf{MacCheck}}$ on $\langle res\rangle$ and $\langle m_{res}\rangle$. Abort if the check fails.
return $\langle res\rangle$.

C.3 Reciprocal Square Root

The detailed steps of the secure reciprocal square root protocol are provided in Algorithms 6. In this protocol, the following lookup tables are used:

- $\vec{T}_{\mathsf{exp}}[i] = \left\lfloor (b^i/2^f)^{-1} \cdot 2^f \right\rfloor$ for $i \in [0, \lceil \log_b 2^{2f} - 1 \rceil]$
- $\vec{T}_{\mathsf{exsqrt}}[i] = \left\lfloor \sqrt{(b^i/2^f)^{-1}} \cdot 2^f \right\rfloor$ for $i \in [0, \lceil \log_b 2^{2f} - 1 \rceil]$
- $\vec{T}_{\mathsf{rsqrt}}[i] = \left\lfloor (\sqrt{i/2^f + 1/b})^{-1} \cdot 2^f \right\rfloor$ for $i \in [0, (1 - 1/b) \cdot 2^f]$

Here, f denotes the number of fractional bits in the fixed-point representation.

Algorithm 6 Reciprocal Square Root (Π_{Rsqrt})

Input: $\binom{3}{2}$-shared $\langle x\rangle$, scaling base b, fixed-point precision f, table $\vec{T}_{\mathsf{exp}}$, table $\vec{T}_{\mathsf{exsqrt}}$, and table $\vec{T}_{\mathsf{rsqrt}}$, where the size of $\vec{T}_{\mathsf{exp}}$ is $n = \lceil \log_b 2^{2f} - 1 \rceil$.
Output: $\binom{3}{2}$-shared $\langle z\rangle$ where $z = 1/\sqrt{x}$.
for $i = 1$ to n **do** in parallel
  $\langle \vec{d}[i]\rangle \leftarrow \Pi_{\mathsf{DReLU}}(\langle x\rangle - b^i)$
end for
$\langle k\rangle = \sum_{i=0}^{n-1} \langle d[i]\rangle$
$\langle t\rangle \leftarrow \Pi_{\mathsf{LUT}}(\langle k+1\rangle, \vec{T}_{\mathsf{exp}})$
$\langle s\rangle \leftarrow \Pi_{\mathsf{LUT}}(\langle k+1\rangle, \vec{T}_{\mathsf{exsqrt}})$
$\langle q\rangle \leftarrow \mathcal{F}_{\mathsf{mul}}^{\langle\cdot\rangle}(\langle x\rangle, \langle t\rangle)$
$\langle w\rangle \leftarrow \Pi_{\mathsf{LUT}}(\langle q - 1/b\rangle, \vec{T}_{\mathsf{rsqrt}})$
return $\langle z\rangle \leftarrow \mathcal{F}_{\mathsf{mul}}^{\langle\cdot\rangle}(\langle s\rangle, \langle w\rangle)$.

C.4 GELU

Since the difference $\delta(x)$ between ReLU and GELU is zero outside the interval $[-4, 4]$, GELU can be expressed as:

$$\mathrm{GELU}(x) = \begin{cases} \mathrm{ReLU}(x), & x \notin [-4, 4] \\ \mathrm{ReLU}(x) - \delta(x), & x \in [-4, 4] \end{cases}$$

Given that $\delta(x)$ is an even function, we only need to precompute function values over the interval $[0, 4]$. By discretizing the input domain using 6-bit precision, the size of the lookup table is restricted to 2^8 entries, significantly improving computational efficiency. To reduce overhead, we design a verifiable GELU protocol based on the CPU-optimized implementation from SIGMA [19], where $\vec{T}_{\mathsf{GELU}}[i] = \lfloor \delta(i/2^6) \cdot 2^f \rfloor$ for $i \in [0, 255]$. The protocol is detailed in Algorithm 7.

Algorithm 7 GELU (Π_{GELU})

Input: Secret sharing value $\langle x\rangle$, and table $\vec{T}_{\mathsf{GELU}}$.
Output: Secret shared value $\langle z\rangle$ where $z = \mathsf{GELU}(x)$.
1: $\langle y\rangle \leftarrow \mathcal{F}_{\mathsf{trunc}}(\langle x\rangle, f - 6)$
2: $\langle p_x\rangle \leftarrow \Pi_{\mathsf{Select}}(\langle y\rangle, \langle x\rangle)$
3: $\langle p_y\rangle \leftarrow \Pi_{\mathsf{Select}}(\langle y\rangle, \langle y\rangle)$
4: $\langle a\rangle = 2 \cdot \langle p_y\rangle - \langle y\rangle$
5: $\langle \delta\rangle = 256 - \langle a\rangle$
6: $\langle i\rangle \leftarrow \Pi_{\mathsf{Select}}(\langle \delta\rangle, \langle a\rangle)$
7: **return** $\langle z\rangle = \langle p_x\rangle - \Pi_{\mathsf{LUT}}(\langle i\rangle, \langle \vec{T}_{\mathsf{GELU}}\rangle)$.

The truncation in step 1 of Algorithm 7 is to limit the range and precision of the input data. When the float precision bit of the input value is the same as that of the elements in table $\vec{T}_{\mathsf{GELU}}$, we can directly use the input to calculate the corresponding index in the table without truncation. The simplified GELU protocol is shown in Algorithm 8.

C.5 Softmax

Softmax consists of three main operations: Max_k, exponentiation, and division. Since division can be implemented via Π_{Inv} and $\mathcal{F}_{\mathsf{mul}}^{\langle\cdot\rangle}$

Algorithm 8 Simplified GELU (Π_{GELU})

Input: Secret sharing value $\langle x\rangle$ with precision bits $f = 6$, and table $\vec{T}_{\mathsf{GELU}}$.
Output: Secret shared value $\langle z\rangle$ where $z = \mathsf{GELU}(x)$.

```
⟨p⟩ = Π_Select(⟨x⟩, ⟨x⟩)
⟨a⟩ = 2 · ⟨p⟩ − ⟨x⟩
⟨δ⟩ = 256 − ⟨a⟩
⟨i⟩ = Π_Select(⟨δ⟩, ⟨a⟩)
⟨z⟩ = ⟨p⟩ − Π_LUT(⟨i⟩, ⟨T_GELU⟩)
```

Algorithm 9 Max_k (Π_{Max})

Input: Secret sharing vector $\langle\vec{x}\rangle$. The length of the vector is k.
Output: Secret shared value $\langle z\rangle$ where $z = \max(\vec{x})$.

```
if k = 1 then
    return ⟨x[k − 1]⟩.
end if
if n mod 2 = 1 then
    Let ⟨x[k]⟩ = ⟨x[k − 1]⟩.
end if
for i = 0 to ⌈k/2⌉ do in parallel
    ⟨x'[i]⟩ = Π_Select(⟨x[i]⟩ − ⟨x[i + ⌈k/2⌉]⟩, ⟨x[i]⟩ − ⟨x[i + ⌈k/2⌉]⟩) + ⟨x[i + ⌈k/2⌉]⟩   ▷ x'[i] = max(x[i], x[i + ⌈k/2⌉])
end for
return Π_Max(⟨x'⟩)
```

Algorithm 10 Exponential (Π_{Exp})

Input: Secret sharing value $\langle x\rangle$ where $x \le 0$, and table $\vec{T}_{\mathsf{exp2}}$.
Output: Secret shared value $\langle v\rangle$ where $v = \exp(x)$.

```
⟨z⟩ = −⟨x⟩/ln 2
⟨z'⟩ = Π_Select(⟨z⟩ − f, f − ⟨z⟩) + ⟨z⟩   ▷ z' = min(z, f) i.e. z' = clip(z, 0, f)
⟨e_z⟩ ← Π_LUT(⟨z'⟩, T_exp2)
⟨p⟩ = ⟨z⟩ · ln 2 + ⟨x⟩
⟨e_p⟩ = 0.3585 · F_mul^⟨·⟩(⟨p⟩ + 1.353, ⟨p⟩ + 1.353) + 0.344
⟨v⟩ ← F_mul^⟨·⟩(⟨e_p⟩, ⟨e_z⟩)
```

Algorithm 11 Softmax (Π_{Softmax})

Input: Secret sharing vector $\langle\vec{x}\rangle$. The length of the vector is n.
Output: Secret shared vector $\langle\vec{z}\rangle$ where $\vec{z} = \mathrm{Softmax}(\vec{x})$.

```
⟨x_max⟩ ← Π_Max(⟨x⟩)
for i = 0 to n − 1 do
    ⟨δ[i]⟩ = ⟨x[i]⟩ − ⟨x_max⟩
end for
for i = 0 to n − 1 do in parallel
    ⟨e[i]⟩ ← Π_exp(⟨δ[i]⟩)
end for
⟨s⟩ = Σ_{i=0}^{n−1} ⟨e[i]⟩
⟨r⟩ ← Π_Inv(⟨s⟩)
for i = 0 to n − 1 do in parallel
    ⟨z[i]⟩ ← F_mul^⟨·⟩(⟨e[i]⟩, ⟨r⟩)
end for
return ⟨z⟩
```

as discussed earlier, we focus here on the secure protocols for Max_k and exponentiation.

Given a shared vector $\langle\vec{x}\rangle$ of length k, the Max_k protocol outputs a share of the maximum element. We recursively reduce the vector length by dividing it into two equal halves (duplicating the last element if k is odd), and applying the Π_{Select} protocol to compute element-wise maxima. This process repeats until a single maximum remains, requiring $\lceil \log_2 k\rceil$ invocations of Π_{Select}. Details are provided in Algorithm 9. To compute $\exp(x - x_{\max})$, we first apply Π_{Max} to obtain $x_{\max}$, and shift the input to $\delta = x - x_{\max} \le 0$. Following BOLT [39], any non-positive input can be decomposed as $x = (-\ln 2)\cdot z + p$ where $z \in \mathbb{Z}_+$ and $p \in (-\ln 2, 0]$, leading to $\exp(x) = \exp(p)\cdot 2^{-z}$. We approximate $\exp(p)$ using the quadratic formula $\exp(p) \approx 0.3585(p + 1.353)^2 + 0.344$. Since computation is performed over $\mathbb{Z}_{2^\ell}$ with f-bit fixed-point precision, we express 2^{-z} as 2^{f-z}. Given that z is non-negative, 2^{-z} can be implemented via lookup table. We compute $z = -x/\ln 2$ and clip it to $[0, f]$ via $\min(z, f)$, then evaluate 2^{-z} using Π_{LUT} over the table $\vec{T}_{\mathsf{exp2}}[i] = 2^{f-i}$ for $i \in [0, f]$. The complete exponentiation protocol is shown in Algorithm 10. Combining Max_k, exponentiation, and division, we construct the secure Softmax protocol, as detailed in Algorithm 11.

C.6 Secure Modulus Conversion Protocols

For a value $x \in \mathbb{Z}_{2^\ell}$ with f-bit precision, which has been shared as x_0, x_1, x_2, the type conversion is based on the following equation:

$$\begin{aligned} x' &= x \cdot 2^{f'-f} \mod 2^{\ell'} \\ &= ((x_0 + x_1 + x_2) \mod 2^{\ell}) \cdot 2^{f'-f} \mod 2^{\ell'} \\ &= (x_0 + x_1 + x_2 - w\cdot 2^{\ell})\cdot 2^{f'-f} \mod 2^{\ell'} \end{aligned} \tag{8}$$

where $x' \in \mathbb{Z}_{2^{\ell'}}$ with f'-bit precision, w represents the wrap counts of the sum of x_i modulo 2^ℓ, i.e., $w = \lfloor (x_0 + x_1 + x_2)/2^\ell \rfloor$.

Downcast. For a fixed-point integer $x \in \mathbb{Z}_{2^\ell}$ with f-bit precision, downcast operation converts it to $x' \in \mathbb{Z}_{2^{\ell'}}$ with f'-bit precision, where $\ell > \ell'$ and $f \ge f'$. Eq. (8) can be expanded as

$$\begin{aligned} x' &= (x_0 + x_1 + x_2 - w\cdot 2^{\ell})\cdot 2^{f'-f} \mod 2^{\ell'} \\ &= (\lfloor x_0\cdot t\rfloor + \lfloor x_1\cdot t\rfloor + \lfloor x_2\cdot t\rfloor - w\cdot 2^{\ell+f'-f} + w') \mod 2^{\ell'} \\ &= ((\lfloor x_0\cdot t\rfloor + \lfloor x_1\cdot t\rfloor + \lfloor x_2\cdot t\rfloor + w') \mod 2^{\ell'} \\ &\quad - w\cdot 2^{\ell+f'-f} \mod 2^{\ell'}) \mod 2^{\ell'} \end{aligned} \tag{9}$$

where t represents $2^{f'-f}$, $w' \in \{0, 1, 2\}$ denotes the potential carry bits from the lower $f - f'$ bits. Since we usually have $\ell + f' - f > \ell'$, e.g. $\ell = 64, f = 16, \ell' = 32, f' = 12$, $w\cdot 2^{\ell+f'-f} \mod 2^{\ell'} = 0$. We can finally get

Algorithm 12 Upcast (Π_{Up})

Input: Secret sharing value $\langle x\rangle$ over $\mathbb{Z}_{2^\ell}$ with f-bit precision.
Output: Secret shared value $\langle x'\rangle$ over $\mathbb{Z}_{2^{\ell'}}$ with f'-bit precision where $x' = x, \ell' > \ell, f' >= f$.

[Setup]

1: The parties randomly samples $\langle r'\rangle^B \leftarrow \mathcal{F}_{rand}((\mathbb{Z}_2)^{\ell'})$.
2: Let $\langle s\rangle^B$ be the most significant shares of $\langle r'\rangle^B$, i.e. $s = \mathsf{MSB}(r')$.
3: The parties randomly samples $\langle r'_1\rangle^B, \langle r'_2\rangle^B \leftarrow \mathcal{F}_{rand}((\mathbb{Z}_2)^{\ell'})$ and $\langle s_1\rangle^B, \langle s_2\rangle^B \leftarrow \mathcal{F}_{rand}(\mathbb{Z}_2)$.
4: Reveal r'_1, s_1 to P_0, P_1 and reveal r'_2, s_2 to P_1, P_2 by invoking $\mathcal{F}_{\mathsf{recon}}$.
5: The parties jointly compute $\langle r'_0\rangle^B = \langle r'\rangle^B - \langle r'_1\rangle^B - \langle r'_2\rangle^B$, $\langle s_0\rangle^B = \langle r\rangle^B - \langle s_1\rangle^B - \langle s_2\rangle^B$ and reveal r'_0, s_0 to P_0, P_1.
6: Let $\langle r'\rangle = (r'_0, r'_1, r'_2)$, $\langle s\rangle = (s_0, s_1, s_2)$.
7: Obtain $\langle r\rangle$ by converting $\langle r'\rangle$ from $\mathbb{Z}_{2^{\ell'}}$ to $\mathbb{Z}_{2^\ell}$.

[Evaluation] Upon receiving $\langle x\rangle$, the party $P_b, b \in \mathbb{Z}_3$ does:

8: $\langle \hat{x}\rangle = \langle x\rangle + \langle r\rangle + 2^{\ell-2}$ and open $\hat{x} \leftarrow \mathcal{F}_{\mathsf{open}}(\langle \hat{x}\rangle)$
9: $\langle \hat{w}\rangle = \langle s\rangle \cdot (1 - \mathsf{MSB}(\hat{x}))$
10: $\langle x'\rangle = (\hat{x} - \langle r'\rangle + \langle \hat{w}\rangle \cdot 2^\ell - 2^{\ell-2}) \cdot 2^{(f'-f)}$

$$\begin{aligned}
x' &= (\lfloor x_0 \cdot t\rfloor + \lfloor x_1 \cdot t\rfloor + \lfloor x_2 \cdot t\rfloor + w') \mod 2^{\ell'} \\
&= ((\lfloor x_0 \cdot t\rfloor) \mod 2^{\ell'} + (\lfloor x_1 \cdot t\rfloor) \mod 2^{\ell'} + (\lfloor x_2 \cdot t\rfloor) \mod 2^{\ell'} \\
&+ w' \mod 2^{\ell'}) \mod 2^{\ell'} \\
&= (x'_0 + x'_1 + x'_2 + w') \mod 2^{\ell'}
\end{aligned} \tag{10}$$

The probabilistic error w' occurs at the lowest significant bit, thus merely having a negligible impact of precision $2^{-f'}$. Consequently, the downcast operation can be done locally by right-shifting $f-f'$ bits followed by modulo $2^{\ell'}$, i.e., $\langle x'\rangle = \langle x\rangle \gg (f-f') \mod 2^{\ell'}$.

Upcast. In opposite, the upcast operation converts a fixed-point integer $x \in \mathbb{Z}_{2^\ell}$ with f-bit precision to $x' \in \mathbb{Z}_{2^{\ell'}}$ with f'-bit precision, where $\ell < \ell'$ and $f \le f'$. However, since $\ell + f' - f < \ell'$ holds in this case, $w \cdot 2^{\ell+f'-f} \mod 2^{\ell'}$ cannot be omitted. We take the random-masked strategy and convert Eq. (8) to

$$x' = ((x + r) \mod 2^\ell - r - \hat{w} \cdot 2^\ell) \mod 2^{\ell'} \tag{11}$$

where $\hat{w} = 1\{x + r > 2^\ell\}$. The problem now reduces to compute another warp $\hat{w}$ and we use the positive heuristic trick in Ditto [49]. By adding a large bias $2^{\ell-2}$ to x, where $x \in [0, 2^{\ell-2} - 1] \cup [2^\ell - 2^{\ell-2}, 2^\ell - 1]$, we can ensure $x' = x + 2^{\ell-2} \in [0, 2^{\ell-1} - 1]$. As a result, $\hat{w} = \mathsf{MSB}(r) \cdot (1 - \mathsf{MSB}(x' + r \mod 2^\ell))$. The details of upcast are shown in Algorithm 12.

D Theoretical Analysis

In this part, we analyze the complexity of our secure protocols ($\Pi_{\mathsf{UCMP}}, \Pi_{\mathsf{DReLU}}, \Pi_{\mathsf{Select}}, \Pi_{\mathsf{LUT}}, \Pi_{\mathsf{Inv}}, \Pi_{\mathsf{Rsqrt}}, \Pi_{\mathsf{GELU}}, \Pi_{\mathsf{Up}}$) and give the security proof under the honest-majority secure model.

D.1 Complexity Analysis

Π_{UCMP}. In the offline phase, for an n-bit secret sharing value, the party P_{b+2} needs to generate the VDPF keys and distribute to the other parties, requiring n DPF keys whose domain is $\mathbb{Z}_{2n}$. Thus, the total key size is $n \cdot ((\lambda + 2) \cdot \log_2 2n + n + 4\lambda)$, where λ is the security parameter of function secret sharing. After receiving the VDPF keys, the other two parties need to verify the n DPF keys by full domain evaluation over $\mathbb{Z}_{2n}$, which need 2 communication rounds with $4n \log_2 2n$ bits communication overhead. In the online phase, the two parties require 1 round communication to open $\vec{\delta}$, whose communication cost is $n \log_2 2n$ bits.

Π_{DReLU}. In addition of the keys required in Π_{UCMP}, Π_{DReLU} requires other auxiliary parameters such as random mask r, MAC key α, et al. The generation and distribution of such keys can be finished in 2 rounds with communication cost being 4ℓ bits. Since the input values of Π_{UCMP} are over ring $\mathbb{Z}_{2^{\ell-1}}$ and each party needs two pairs of Π_{UCMP} keys, the key size is $2(\ell-1) \cdot ((\lambda+2) \cdot \log_2 2\ell + \ell + 4\lambda)$. In the online phase, each party invokes 6 times $\mathcal{F}_{\mathsf{recon}}$ first to open the masked input value, requiring 4 rounds communication and 4ℓ bits cost. And then, each party invokes two times the Π_{UCMP} and B2A, requires 4 rounds communication with $2(\ell-1)\log_2 \ell + 2\ell$ bits. The parties need to convert the $[\![\cdot]\!]$-sharing value to the $\langle\cdot\rangle$-sharing value by reshare, requiring 2 rounds communication and 2ℓ bits communication cost. In the MAC check part, each party needs 8 rounds communication and 8ℓ bits overhead. In summary, the Π_{DReLU} protocol needs 18 rounds communication and $16\ell + 2(\ell - 1)\log \ell$ bits communication cost in the online evaluation phase.

Π_{Select}. Since Π_{Select} has only one extra step of semi-honest security multiplication in the online phase compared to Π_{DReLU}, the online communication overhead of Π_{Select} is 19 rounds communication and $15\ell + 2(\ell-1)\log_2 2\ell + 2$ bits communication cost.

Π_{LUT}. In the offline phase, each party needs to generate the VDPF keys and distribute to the other two parties. The domain of the VDPF keys is $\mathbb{Z}_{2^{\ell_t}}$ and the key size is $(\lambda + 2) \cdot \ell_t + \ell + 4\lambda$, where $\ell_t = \lceil \log_2 n\rceil$ and n is the size of the table. The other two parties need 2 rounds communication to verify the correctness of received keys with the communication cost being $4\ell_t$ bits. In the online phase, the 6 invocations of $\mathcal{F}_{\mathsf{recon}}$ generate 4 communication rounds and 4ℓ bits cost, while the reshare operation needs 2 rounds and 2ℓ bits. Combined with the overhead by $\mathcal{F}_{\mathsf{MacCheck}}$, the total online communication overhead of Π_{LUT} is 14 rounds and 14ℓ cost.

Π_{Inv}. Π_{Inv} requires n calls to Π_{DReLU}, 2 calls to Π_{LUT} and 2 calls to $\mathcal{F}^{\langle\cdot\rangle}_{\mathsf{mul}}$, where $n = \lceil \log_b 2^{2f} - 1\rceil$, b is the scale base. Since Π_{DReLU} can operate on the shift of the input value, we can utilize the same key and operate the n calls in parallel. Additionally, both Π_{DReLU} and Π_{LUT} need MAC check, we can postpone all the check to the end of the protocol and check all the values together. Thus, the online phase needs 1 Π_{DReLU} key and communicates $n \cdot (6\ell + 2(\ell - 1)\log_2 2\ell + 2)$ bits in 10 rounds for Π_{DReLU}, communicates 12ℓ bits in 12 rounds for 2 Π_{LUT}, 16ℓ bits in 12 rounds for 2 $\mathcal{F}^{\langle\cdot\rangle}_{\mathsf{mul}}$ and $(n+2)\cdot 8\ell$ bits in 8 rounds for MAC check. In summarize, the online phase needs 42 communication rounds and $n \cdot (6\ell + 2(\ell-1)\log_2 2\ell + 2) + (28 + 8 \cdot (n+2))\ell$ bits communication cost.

Π_{Rsqrt}. Π_{Rsqrt} needs one more Π_{LUT} call than Π_{Inv}. However, the input of Π_{LUT} in step 5 is the same as that in step 6 of Algorithm 6. We can merge such steps to reduce the communication rounds and computation cost. Thus, the online phase of Π_{Rsqrt} communicates $n \cdot (14\ell + 2(\ell-1)\log_2 2\ell + 2) + 58\ell$ bits in 42 rounds.

Π_{GELU}. Π_{GELU} requires 3 calls to Π_{Select}, 1 Π_{LUT} call and 1 Trunc. We put all the mac checks at the end of the protocol to verify all the calculation results. The Π_{Select} in step 2 and step 3 can be merged due to the same input. Consequently, the online phase needs 38 communication rounds with $57\ell + 4(\ell-1)\log_2 2\ell + 4$ bits.

Π_{Up}. Unlike Downcast protocol can be conducted locally, Π_{Up} needs 6 invocations of $\mathcal{F}_{\mathsf{recon}}$ in the offline phase, which communicates $4(\ell'+1)$ bits in 4 rounds. In the online phase, it only requires 1 invocation of $\mathcal{F}_{\mathsf{open}}$ and communicates 2ℓ bits in 2 rounds.

D.2 Security Proof

We employ the simulation and universal composition (UC) theory [6] to prove that our protocol satisfy the following definition.

Definition 3. *(Three-party Secure Computation [15]). Let $\mathcal{F}$ be the three-party functionality. A protocol Π securely computes $\mathcal{F}$ with abort in the presence of one malicious party, if for every party P_b corrupted by a probabilistic polynomial time (PPT) adversary $\mathcal{A}$ in the real world there exists a PPT simulator S in the ideal world with $\mathcal{F}$, such that*

$$\{IDEAL_{\mathcal{F},\mathcal{S}(z),b}(x_0,x_1,x_2,\lambda)\} \stackrel{c}{\equiv} \{REAL_{\Pi,\mathcal{A}(z),b}(x_0,x_1,x_2,\lambda)\}$$

where $x_b \in \{0,1\}^*$ under the constraint that $|x_0| = |x_1| = |x_2|$, the auxiliary input $z \in \{0,1\}^*$, and the security parameter $\lambda \in \mathbb{N}$. We say that π securely computes $\mathcal{F}$ with abort in the presence of one malicious party with statistical error $2^{-\kappa}$ (κ denotes the statistical parameter) if there exists a negligible function $\mu(\cdot)$ such that the distinguishing probability of the adversary is less than $2^{-\kappa} + \mu(\lambda)$.

For any PPT adversary $\mathcal{A}$, we construct a PPT simulator $\mathcal{S}$ that can simulate the adversary's view with accessing each functionality. In the cases where $\mathcal{S}$ aborts or terminates the simulation, $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs.

Theorem 1. *The setup phase of Π_{UCMP} is securely executed in the presence of a malicious adversary in the three-party honest majority setting, while the evaluation phase is securely executed in the semi-honest two-party setting.*

Proof. **Simulating setup phase**. When the malicious adversary $\mathcal{A}$ corrupts the key provider P_{b+2}, $\mathcal{S}$ emulates the behavior of the honest evaluation parties P_b and P_{b+1}. Upon receiving the VDPF keys from $\mathcal{A}$, $\mathcal{S}$ locally checks whether the keys are correct and abort if not. When the malicious adversary $\mathcal{A}$ corrupts one of the evaluation parties P_b and P_{b+1}, and the key provider P_{b+2} is assumed to be semi-honest, $\mathcal{S}$ generates the VDPF keys and samples random shares $[[\vec{r}[i]]] \xleftarrow{\$} \mathbb{Z}_{2n}$ for $i \in [0, n-1]$. Then $\mathcal{S}$ sends the keys $([[\vec{r}]], \vec{k^\bullet})$ to $\mathcal{A}$. $\mathcal{S}$ proceeds to run the VDPF verification protocol with $\mathcal{A}$, and aborts if $\mathcal{A}$ aborts during the verification. For all other messages sent from honest parties to the corrupted one, the simulator samples them randomly.

The simulation above is indistinguishable from real-world execution. If the adversary provides malformed VDPF keys when acting as the key provider, the verification protocol would abort in both the ideal and real worlds. Since $\mathcal{S}$ performs the same checks and aborts accordingly, the abort behavior remains indistinguishable. When the corrupted party acts as a DPF evaluator, the keys are honestly generated by $\mathcal{S}$, and the verification proceeds identically to the real-world protocol. If $\mathcal{A}$ behaves honestly during key verification, the protocol proceeds successfully; otherwise, it aborts. In all cases, the view of the adversary in the simulation is indistinguishable from that in the real execution, thereby establishing the security of the protocol in the presence of a malicious adversary.

Simulating evaluation phase. The evaluation phase is only executed securely in the semi-honest two-party setting. Given that P_b is corrupted by the adversary, for all intermediate values $[[\vec{u}]]$, $[[\vec{w}]]$, $[[\vec{c}]]$, and $[[\vec{\delta}]]$, $\mathcal{S}$ samples random numbers $[[\vec{u}[i]]]_{sim} \in \mathbb{Z}_{2n}$, $[[\vec{w}[i]]]_{sim} \in \mathbb{Z}_{2n}$, $[[\vec{c}[i]]]_{sim} \in \mathbb{Z}_{2n}$, $[[\vec{\delta}[i]]]_{sim} \in \mathbb{Z}_{2n}$ for $i \in [0, n-1]$. For any message received from the honest party P_{b+1} is similarly simulated by randomly sampling.

The simulation is indistinguishable from real-world execution. According to the definition of secret sharing, the elements of $[[\vec{x}]]$ are uniformly distributed over $\mathbb{Z}_{2n}$ and each bit of y is also over $\mathbb{Z}_2$. As a result, the elements of $[[\vec{u}]]$, $[[\vec{w}]]$, $[[\vec{c}]]$ are all random numbers over $\mathbb{Z}_{2n}$. Since the elements of $[[\vec{\alpha}]]$ are random numbers over $\mathbb{Z}_{2n}$, $[[\vec{\delta}[i]]]$ are random numbers over $\mathbb{Z}_{2n}$. Consequently, Therefore, the simulated shares are statistically indistinguishable from those produced during real execution. Moreover, since both P_b and P_{b+1} follow the same protocol and operate over identically distributed secret shares, the same simulation strategy can be applied symmetrically when the adversary corrupts P_{b+1}.

Overall, the simulation is indistinguishable from real-world execution.

□

Theorem 2. *Π_{DReLU} securely compute $\mathcal{F}_{DReLU}$ in the $\{\mathcal{F}_{UCMP}, \mathcal{F}_{recon}\}$-hybrid model in the presence of a malicious adversary in the three-party honest-majority setting.*

Proof. **Simulating setup phase**. For the generation of keys used in Π_{UCMP}, simulation follows directly from the setup phase simulator of Π_{UCMP}. We now describe how the simulator handles the generation and verification of other auxiliary parameters.

When the corrupted party P_b is the parameter provider, $\mathcal{S}$ emulates the behavior of the honest parties P_{b+1} and P_{b+2}. Upon receiving the parameters keys from $\mathcal{A}$, $\mathcal{S}$ locally checks whether the parameters are correct and abort if not. If an honest party generates the parameters, $\mathcal{S}$ samples and sends random shares to $\mathcal{A}$. Specifically, $\mathcal{S}$ samples $[[r]]_{sim} \in \mathbb{Z}_{2^\ell}$, $[[c]]_{sim} \in \mathbb{Z}_2$, and $[[\vec{y}[i]]]_{sim} \in \mathbb{Z}_{2(\ell-1)}$, for $i \in [0, \ell-2]$. $\mathcal{S}$ proceeds to run the verification protocol with $\mathcal{A}$ and aborts if $\mathcal{A}$ aborts. For all other messages sent from honest parties to the corrupted one, the simulator samples them randomly.

The simulation above is indistinguishable from real-world execution. If the adversary provides malformed parameters when acting as the key provider, the verification protocol would abort in both the ideal and real worlds. Since $\mathcal{S}$ performs the same checks and aborts accordingly, the abort behavior remains indistinguishable. When the corrupted party receives honestly generated parameters, $\mathcal{S}$ simulates the corresponding shares and runs the verification protocol identically to the real-world interaction. If $\mathcal{A}$ behaves honestly during the verification, the protocol proceeds successfully; otherwise, it aborts. In all cases, the view of the adversary in the simulation is indistinguishable from that in the real execution, thereby establishing the security of the protocol in the presence of a malicious adversary.

Simulating evaluation phase. Due to the symmetric structure of the Π_{DReLU} protocol, without loss of generality, we assume that the malicious adversary $\mathcal{A}$ corrupts party P_b, while parties P_{b+1} and P_{b+2} behave honestly.

If the protocol does not abort during the simulation of the setup phase, all VDPF keys and other parameters generated by $\mathcal{A}$ are correct. The remaining challenge in the online phase lies in ensuring that $\mathcal{A}$ follows the protocol honestly, and in enabling the simulator $\mathcal{S}$ to detect any deviation by extracting additive errors introduced during the interactive steps. We assume that the shared message $\langle x \rangle$ is correctly shared/simulated initially, meaning the honest parities hold correct and consistent RSS shares.

All local computations can be simulated by $\mathcal{S}$ easily. $\mathcal{S}$ only needs to simulate interactions between honest and corrupted parties and abort with indistinguishable probability. Most steps in the evaluation phase are local, with only a few involving interaction.

First, securely reconstructing $\hat{x}$: since $\langle \hat{x} \rangle = \langle x \rangle + \langle r \rangle$ and both $\langle x \rangle$ and $\langle r \rangle$ are correct RSS sharings, $\hat{x}$ is consistent. $\mathcal{S}$ checks whether $\mathcal{A}$ sends the correct value using share from the honest party: 1) if $\mathcal{A}$ sends an incorrect share, $\mathcal{S}$ aborts; 2) otherwise, $\mathcal{S}$ samples a random $\hat{x}_{sim} \in \mathbb{Z}_{2^\ell}$, derives consistent shares for P_{b+1} and P_{b+2} using $\hat{x}_{sim}$ and P_b's RSS shares, and sends them to $\mathcal{A}$. The parties then open $\hat{x}$ to $\mathcal{A}$.

Second, interactive steps such as computing θ, and resharing $[[v]]$ and $[[mv]]$, may allow $\mathcal{A}$ to introduce errors. The simulator $\mathcal{S}$ simulates these steps as follows: $\mathcal{S}$ receives $[[v^*]]_b$ and $[[mv^*]]_b$ from the corrupted party. It locally computes the correct values $[[v]]_b$ and $[[mv]]_b$ using knowledge of the relevant shares. Then, $\mathcal{S}$ computes $\Delta_v = [[v^*]] - [[v]]$ and $\Delta_m = [[mv^*]] - [[mv]]$. If either $\Delta_v \neq 0$ or $\Delta_m \neq 0$, $\mathcal{S}$ aborts at the end of the protocol. This simulation is statistically indistinguishable from real-protocol execution, with statistical error $\frac{1}{2^\ell - 1}$.

Additionally, during the opening of $\hat{x}$, $\mathcal{S}$ ensures that the honest parties' shares (whether simulated or derived from local data) are consistent with the share held by the corrupted party. This ensures the reconstruction of $\hat{x}$ remains indistinguishable from the real protocol execution. For the re-sharing of values such as $[[v]]$ and $[[mv]]$, the simulator randomly samples P_{b+2}'s share and sends it to $\mathcal{A}$. In the real-world protocol, this value is generated using a pseudorandom function (PRF) F, and thus the simulated share is computationally indistinguishable due to the PRF's security.

Overall, the simulation is indistinguishable from real execution. □

THEOREM 3. *Π_{Select} securely compute $\mathcal{F}_{Select}$ in the $\{\mathcal{F}_{UCMP}, \mathcal{F}_{recon}\}$-hybrid model in the presence of a malicious adversary in the three-party honest-majority setting.*

PROOF. Since the setup phase of Π_{Select} is identical to that of Π_{DReLU}, the simulation can be directly inherited by invoking the setup-phase simulator of Π_{DReLU}.

In the evaluation phase, the only distinction between Π_{Select} and Π_{DReLU} lies in an additional invocation of a semi-honest multiplication protocol required to compute the product my. The malicious adversary $\mathcal{A}$ can add errors in this step. Such errors can affect the resharing phase, during which $\mathcal{S}$ receives a possibly corrupted share $[[mv^*]]$. To extract this error, since $\mathcal{S}$ knows all necessary shares to compute $[[mv]]$, $\mathcal{S}$ computes $\Delta_m = [[mv^*]] - [[mv]]$. If $\Delta_m \neq 0$, $\mathcal{S}$ aborts at the end of the protocol. All remaining operations in Π_{Select} are functionally equivalent to those in Π_{DReLU} and can thus be simulated using the corresponding components of the Π_{DReLU} simulator.

Consequently, this simulation is statistically indistinguishable from real-protocol execution, with statistical error $\frac{1}{2^\ell - 1}$. □

THEOREM 4. *Π_{LUT} securely compute $\mathcal{F}_{LUT}$ in the $\{\mathcal{F}_{recon}\}$-hybrid model in the presence of a malicious adversary in the three-party honest-majority setting.*

PROOF. **Simulating setup phase**. When the corrupted party P_b is the VDPF key provider, $\mathcal{S}$ emulates the behavior of the honest parties P_{b+1} and P_{b+2}. Upon receiving a pair of VDPF keys from $\mathcal{A}$, $\mathcal{S}$ locally checks whether the keys are correct and abort if not. If an honest party generates the parameters, $\mathcal{S}$ create a pair of VDPF keys and samples a random number $[[r]]_{sim} \in \mathbb{Z}_{2^\ell}$, then sends $([[r]]_{sim}, k_{sim})$ to $\mathcal{A}$. $\mathcal{S}$ proceeds to run the VDPF verification protocol with $\mathcal{A}$ and aborts if $\mathcal{A}$ aborts. For all other messages sent from honest parties to the corrupted one, the simulator samples them randomly.

The simulation above is indistinguishable from real-world execution. If the adversary provides malformed VDPF keys when acting as the key provider, the verification protocol would abort in both the ideal and real worlds. Since $\mathcal{S}$ performs the same checks and aborts accordingly, the abort behavior remains indistinguishable. When the corrupted party acts as a DPF evaluator, the keys are honestly generated by $\mathcal{S}$, and the verification proceeds identically to the real-world protocol. If $\mathcal{A}$ behaves honestly during the verification, the protocol proceeds successfully; otherwise, it aborts. In all cases, the view of the adversary in the simulation is indistinguishable from that in the real execution, thereby establishing the security of the protocol in the presence of a malicious adversary.

Simulating evaluation phase. Due to the symmetric structure of the Π_{LUT} protocol, without loss of generality, we assume that the malicious adversary $\mathcal{A}$ corrupts party P_b, while parties P_{b+1} and P_{b+2} behave honestly.

If the protocol does not abort during the simulation of the setup phase, all VDPF keys generated by $\mathcal{A}$ are correct. The remaining challenge in the online phase lies in ensuring that $\mathcal{A}$ follows the protocol honestly, and in enabling the simulator $\mathcal{S}$ to detect any deviation by extracting additive errors introduced during the interactive steps. We assume that the shared message $\langle x \rangle$ is correctly shared/simulated initially, meaning the honest parities hold correct and consistent RSS shares.

All local computations can be simulated by $\mathcal{S}$ easily. $\mathcal{S}$ only needs to simulate interactions between honest and corrupted parties and abort with indistinguishable probability. Most steps in the evaluation phase are local, with only a few involving interaction.

First, securely reconstructing $\hat{x}$: since $\langle \hat{x} \rangle = \langle x \rangle + \langle r \rangle$ and both $\langle x \rangle$ and $\langle r \rangle$ are correct RSS sharings, $\hat{x}$ is consistent. $\mathcal{S}$ checks whether $\mathcal{A}$ sends the correct value using share from the honest party: 1) if $\mathcal{A}$ sends an incorrect share, $\mathcal{S}$ aborts; 2) otherwise, $\mathcal{S}$

samples a random $\hat{x}_{sim} \in \mathbb{Z}_{2^\ell}$, derives consistent shares for P_{b+1} and P_{b+2} using $\hat{x}_{sim}$ and P_b's RSS shares, and sends them to $\mathcal{A}$. The parties then open $\hat{x}$ to $\mathcal{A}$.

Another part is from resharing $[[res]]$ and $[[m_{res}]]$ back to RSS sharing, which may allow $\mathcal{A}$ to introduce errors. The simulator $\mathcal{S}$ simulates these steps as follows: $\mathcal{S}$ receives $[[res^*]]_b$ and $[[m_{res}{}^*]]_b$ from the corrupted party. It locally computes the correct values $[[res]]_b$ and $[[m_{res}]]_b$ using knowledge of the relevant shares. Then, $\mathcal{S}$ computes $\Delta = [[res^*]] - [[res]]$ and $\Delta_m = [[m^*_{res}]] - [[m_{res}]]$. If either $\Delta \neq 0$ or $\Delta_m \neq 0$, $\mathcal{S}$ aborts at the end of the protocol. This simulation is statistically indistinguishable from real-protocol execution, with statistical error $\frac{1}{2^\ell - 1}$.

Additionally, during the opening of $\hat{x}$, $\mathcal{S}$ ensures that the honest parties' shares (whether simulated or derived from local data) are consistent with the share held by the corrupted party. This ensures the reconstruction of $\hat{x}$ remains indistinguishable from the real protocol execution. For the re-sharing of values such as $[[res]]$ and $[[m_{res}]]$, the simulator randomly samples P_{b+2}'s share and sends it to $\mathcal{A}$. In the real-world protocol, this value is generated using a pseudorandom function (PRF) F, and thus the simulated share is computationally indistinguishable due to the PRF's security.

Overall, the simulation is indistinguishable from real execution. □

THEOREM 5. *Π_{Up} securely compute $\mathcal{F}_{Up}$ in the $\{\mathcal{F}_{recon}, \mathcal{F}_{open}\}$-hybrid model in the presence of a malicious adversary in the three-party honest-majority setting.*

PROOF. Simulation is done using the hybrid argument. Π_{Up} are sequential combinations of local computations and invocations of $\mathcal{F}_{\mathsf{recon}}$ and $\mathcal{F}_{\mathsf{open}}$, Thus, this protocol can be simulated using the corresponding simulators. □

THEOREM 6. *Π_{Inv}, Π_{Rsqrt}, Π_{GELU}, Π_{Max}, Π_{Exp} securely compute $\mathcal{F}_{Inv}$, $\mathcal{F}_{Rsqrt}$, $\mathcal{F}_{GELU}$, $\mathcal{F}_{Max}$, $\mathcal{F}_{Exp}$ respectively in the $\{\mathcal{F}_{DReLU}$, $\mathcal{F}_{Select}$, $\mathcal{F}_{LUT}$, $\mathcal{F}^{\langle\cdot\rangle}_{mul}$, $\mathcal{F}_{trunc}\}$-hybrid model in the presence of a malicious adversary in the three-party honest-majority setting.*

PROOF. Simulation is done using the hybrid argument. Π_{Inv}, Π_{Rsqrt} are sequential combinations of local computations and invocations of $\mathcal{F}_{\mathsf{DReLU}}$, $\mathcal{F}_{\mathsf{LUT}}$, and $\mathcal{F}^{\langle\cdot\rangle}_{\mathsf{mul}}$. Π_{GELU} simply composes local computation and $\mathcal{F}_{\mathsf{trunc}}$, $\mathcal{F}_{\mathsf{Select}}$, and $\mathcal{F}_{\mathsf{LUT}}$. Π_{Max} consists of $\mathcal{F}_{\mathsf{Select}}$. Π_{Exp} consists of $\mathcal{F}_{\mathsf{Select}}$, $\mathcal{F}_{\mathsf{LUT}}$, and $\mathcal{F}^{\langle\cdot\rangle}_{\mathsf{mul}}$. Thus, simulations over these protocols follow directly from composing the corresponding simulators. □

E Calculation Ranges of Secure Protocols

As shown in Figure 5, we apply modulus conversion to enable precision-aware computation in non-linear layers. We summarize the valid input ranges for which each nonlinear protocol (e.g., ReLU, Softmax, GELU) produces correct results over various rings, as shown in Table 10, where ℓ denotes the bit length of the ring and f the fixed-point precision. Importantly, based on domain knowledge and empirical observation, the inputs encountered during large transformer inference consistently fall within these ranges. As a result, the accuracy of cryptographic inference is preserved, since all protocols operate within their validity domains.

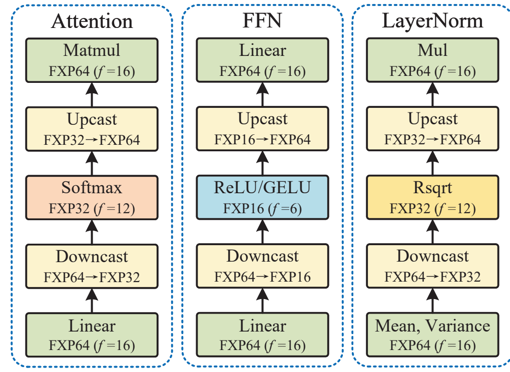


Figure 5: Modulus Conversion in Non-linear Layers of Transformer Model

Table 10: Valid and Observed Value Ranges of Non-linear Layers Under Varying Ring Settings

Protocol	ℓ	f	Valid Range	Observed Range
Inv	32	12	$(0, 4096)$	$(1, 128)$
Rsqrt	32	12	$(0, 4096)$	$(0, 2276)$
ReLU/GELU	16	6	$(-512, 512)$	$(-64, 142)$

F Choice of Scale Base in Inverse and Rsqrt

We analyze the relationship between the scale base and the precision of our secure inverse protocol, which can be extended analogously to the reciprocal square root. The protocol is based on input range normalization and lookup-table evaluation, where the overall approximation error is determined by the table precision and the injectivity of the input-to-index mapping.

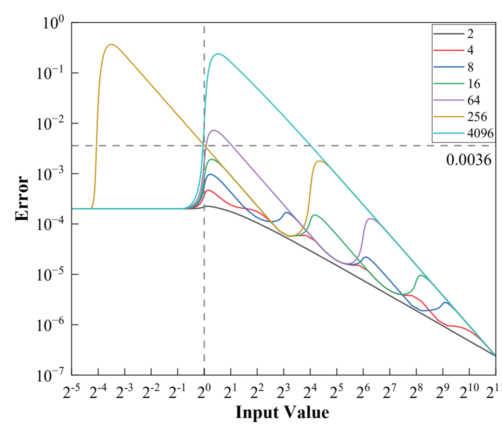


Figure 6: Calculation Error of Secure Inverse Protocol with different Scale Base

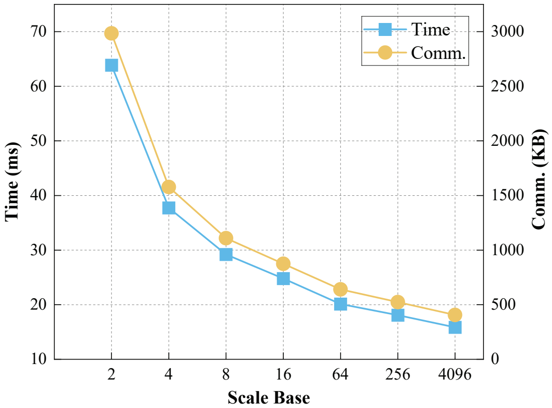


Figure 7: Runtime (ms) and Communication Costs (s) of Secure Inverse Protocol with Different Scale Base in LAN Setting (Input Size $n = 1000$)

For inputs in the interval $[b^k, b^{k+1})$, we normalize values to $[1/b, 1)$ and map them to integers in $[\lfloor 2^f/b \rfloor, 2^f - 1]$, resulting in a table of size $\lceil (1 - 1/b) \cdot 2^f \rceil$. When $b^{k+1} \leq 2^f$, the mapping is injective, and each input corresponds to a unique table entry. In this case, the error is tightly bounded by the table precision (e.g., less than 10^{-3} for $f = 12$). However, when $b^{k+1} > 2^f$, the mapping becomes many-to-one, resulting in increased approximation error. For an input $x \in [b^k, b^{k+1})$, the approximated inverse is given by

$$\hat{x}^{-1} = \left\lfloor \frac{2^f}{\lfloor x \cdot b^{-(k+1)} \cdot 2^f \rfloor} \right\rfloor \cdot \frac{2^f}{b^{k+1}},$$

and the resulting error can be expressed as

$$\epsilon = \left| \frac{2^f}{x} - \hat{x}^{-1} \right|.$$

We evaluate this error under various scale bases $b \in \{2, 4, 8, 16, 64, 256, 4096\}$ as shown in Figure 6. We observe that all result in approximation errors below 10^{-2} for $b \in \{2, 4, 8, 16, 64\}$, making them suitable for secure inverse computation. In the context of large transformer inference, where inverse inputs are typically greater than 1, we find that $b = 256$ achieves a maximum error of 0.0036, which is sufficient to maintain inference accuracy. In contrast, when $b = 4096$, the error becomes significantly larger in the input range $[1, 16]$, rendering it unsuitable for practical use.

Moreover, the number of DReLU invocations required in the protocol is $\lceil \log_b 2^{2f} - 1 \rceil$, so using a larger base b reduces both computation and communication overhead. As shown in Figure 7, the protocol overhead decreases as the scale base increases. Considering both accuracy and efficiency, we select $b = 256$ as the default scale base for secure Transformer inference in Mosformer, as it offers the best trade-off.

G Model Configurations in Our Experiments

The three transformer models including the vanilla transformer [44], BERT-base [12], and GPT2-base [9] are parameterized by three hyperparameters: the number of blocks B, the hidden dimension D, and the number of attention heads H, as detailed in Table 11. The vanilla transformer consists of 6 encoder and 6 decoder blocks; BERT is encoder-only, while GPT2 is decoder-only.

Table 11: Model configurations used in the experiments

Model	Parameters	B	D	H
Vanilla Transformer	$45M$	12	512	8
BERT-base	$110M$	12	768	12
GPT2-base	$124M$	12	768	12

H Experimental Result

H.1 Performance of Π_{GELU}

Table 12 presents the performance of Π_{GELU} across varying input sizes. Π_{GELU} incurs higher overhead due to its non-linear complexity, taking 0.472s/1.135s (LAN/WAN) and 33.95 MB for 10^5 inputs, with 38 rounds. These results demonstrate the scalability and efficiency of Π_{GELU} under realistic workloads.

Table 12: Online Performance of Π_{GELU} in terms of running time (s) and communication costs (MB)

#Inputs	Scheme	LAN	WAN	Comm.	Rounds
10^4	Ours	0.058	0.195	3.395	38
10^5	Ours	0.472	1.135	33.951	38

H.2 Component-Level Performance Analysis

Figure 2 provides a detailed breakdown of Mosformer's implementation of core transformer components, including attention layers, feed-forward networks, layer normalization and our deferred MAC check optimization. These components are all built upon our efficient non-linear protocols such as Π_{Select}, Π_{Exp}, and Π_{Rsqrt}.

H.2.1 Attention Layers. Attention layers, including multi-head attention (MHA) and masked multi-head attention (MMHA), are critical components of the Transformer architecture. These layers involve non-linear operations, notably the softmax function, which includes maximum selection, exponentiation, and reciprocal computation. Mosformer faithfully implements softmax, requiring $\log d_k$ invocations of Π_{select}, one Π_{exp}, and one Π_{Inv}, where d_k denotes the dimension of each attention head (typically 64). In contrast, Privformer replaces softmax in MHA/MMHA with ReLU Attention [8], which uses two ReLU activations and three matrix multiplications, thereby reducing the number of communication rounds. However, thanks to Mosformer's efficient implementation of secure softmax, exponentiation, and reciprocal protocols, it achieves a 5–6.6× speedup and a 7.2× reduction in communication cost over Privformer in the attention layers. Compared to Falcon+, Mosformer achieves a 2.8–4.8× speedup, while reducing communication cost and communication rounds by 1.9× and 16.7×, respectively.

H.2.2 Feed-Forward Network (FFN). The FFN layer comprises matrix multiplications and ReLU activations. Mosformer employs optimized protocols for secure non-linear activations, whereas Privformer directly adopts Falcon's protocols for implementing the FFN layer. As a result, Mosformer achieves a 2–3.5× speedup over both Privformer and Falcon+ in FFN computation.

H.2.3 Layer Normalization (LayerNorm). LayerNorm involves computing the mean and variance of an input vector, followed by scaling and shifting using trainable parameters. This process requires secure protocols for multiplication and reciprocal square root. While Mosformer's Π_{Rsqrt} protocol is more efficient than that of Privformer, the overall impact on runtime is limited in LAN settings due to the small input dimension (e.g., 512 or 768). However, in WAN settings, Mosformer achieves a 2.3× and 2.9× speedup over Privformer and Falcon+, respectively, primarily due to the significantly reduced number of communication rounds.

Notably, Falcon+ merges truncation and reshare steps for secure multiplication and performs post-truncation verification, which may result in correctness issues (see Appendix I). In contrast, Mosformer verifies results prior to truncation, ensuring correctness at the cost of slightly higher communication. Nevertheless, the improved reliability and WAN performance make Mosformer's LayerNorm implementation more robust in practice.

H.2.4 Deferred MAC Checks. To further reduce the number of communication rounds, we defer all MAC checks to the end of the encoder and decoder. These two consolidated verifications require only 16 rounds of communication and account for just 5% of the total runtime. This design significantly contributes to the overall efficiency of our framework.

H.3 Effect of Lookup Table Configuration on Secure Transformer Inference

Mosformer employs a secure lookup table protocol to approximate the computation of inverse and reciprocal square root functions, which are fundamental to the non-linear components in Transformer architectures. While this approximation enables efficient and secure inference, it also introduces a moderate loss in accuracy. This subsection systematically evaluates the effect of different lookup table configurations on the accuracy and runtime of secure transformer inference.

Recall that to reduce the runtime of secure inference under the malicious security model, Mosformer uses two lookup tables (i.e., $\vec{T}_{\mathsf{exp}}$ and $\vec{T}_{\mathsf{inv}}$) to securely approximate the inverse function, rather than relying on a single lookup table as in conventional approaches. For the reciprocal square root computation, Mosformer employs three lookup tables: $\vec{T}_{\mathsf{exp}}$, $\vec{T}_{\mathsf{exsqrt}}$, and $\vec{T}_{\mathsf{rsqrt}}$. Specifically, our secure inverse and reciprocal square root protocols utilize the following precomputed lookup tables (LUTs):

- $\vec{T}_{\mathsf{exp}}[i] = \lfloor (b^i/2^f)^{-1} \cdot 2^f \rfloor$ for $i \in [0, \lceil \log_b 2^{2f} - 1 \rceil]$
- $\vec{T}_{\mathsf{inv}}[i] = \lfloor (i/2^f + 1/b)^{-1} \cdot 2^f \rfloor$ for $i \in [0, (1-1/b) \cdot 2^f]$
- $\vec{T}_{\mathsf{exsqrt}}[i] = \left\lfloor \sqrt{(b^i/2^f)^{-1}} \cdot 2^f \right\rfloor$ for $i \in [0, \lceil \log_b 2^{2f} - 1 \rceil]$
- $\vec{T}_{\mathsf{rsqrt}}[i] = \left\lfloor (\sqrt{i/2^f + 1/b})^{-1} \cdot 2^f \right\rfloor$ for $i \in [0, (1-1/b) \cdot 2^f]$

Table 13: Effect of Lookup Table Configuration on Secure Transformer Inference without Modulus Conversion (Plaintext Accuracy: 90.6%)

Scale Base (b)		2	4	8	16	64	256	512	4096
LUT Size	$\lvert\vec{T}_{\mathsf{exp}}\rvert$	32	16	11	8	6	4	4	3
	$\lvert\vec{T}_{\mathsf{inv}}\rvert$	32768	49152	57344	61440	64512	65280	65408	65520
	$\lvert\vec{T}_{\mathsf{exsqrt}}\rvert$	32	16	11	8	6	4	4	2
	$\lvert\vec{T}_{\mathsf{rsqrt}}\rvert$	32768	49152	57344	61440	64512	65280	65408	65520
Accuracy w/o M.C.		90.6	90.6	90.6	90.4	90.4	90.2	89.6	53.3
Acc. Drop (%)		0	0	0	0.2	0.2	0.4	1.1	41.2
Runtime (s)		110.03	105.7	102.35	100.29	99.52	98.46	98.46	95.98

Table 14: Effect of Lookup Table Configuration on Secure Transformer Inference with Modulus Conversion (Plaintext Accuracy: 90.6%)

Scale Base (b)		2	4	8	16	64	256	512	4096
LUT Size	$\lvert\vec{T}_{\mathsf{exp}}\rvert$	24	12	8	6	4	3	3	1
	$\lvert\vec{T}_{\mathsf{inv}}\rvert$	512	3072	3584	3840	4032	4080	4088	4095
	$\lvert\vec{T}_{\mathsf{exsqrt}}\rvert$	24	12	8	6	4	3	3	1
	$\lvert\vec{T}_{\mathsf{rsqrt}}\rvert$	512	3072	3584	3840	4032	4080	4088	4095
Accuracy w/ M.C.		90.4	90.4	90.2	90.2	89.6	89.1	61.3	53.3
Acc. Drop (%)		0.2	0.2	0.4	0.4	1.1	1.7	32.3	41.2
Runtime (s)		67.36	62.75	61.17	60.16	59.89	59.47	59.48	57.22

Here, f denotes the number of fractional bits in the fixed-point representation. According to the structure of the above LUTs, given a fixed f, the size of each table depends on the scale base b. Table 13 summarizes the sizes of the lookup tables under different scale bases, along with the corresponding accuracy and runtime of secure transformer inference without modulus conversion, where f is set to 16. Table 14 presents the corresponding results with modulus conversion, using $f = 12$.

As shown in Table 13, increasing the scale base b affects the sizes of the lookup tables differently. While exponential-related tables such as $\vec{T}_{\mathsf{exp}}$ and $\vec{T}_{\mathsf{exsqrt}}$ shrink with increasing b, inverse-related tables such as $\vec{T}_{\mathsf{inv}}$ and $\vec{T}_{\mathsf{rsqrt}}$ grow accordingly. Nevertheless, the overall runtime decreases as b increases, since exponential-related computation dominates the total cost and benefits most from smaller $\vec{T}_{\mathsf{exp}}$ and $\vec{T}_{\mathsf{exsqrt}}$. For instance, the runtime drops from 110.03s at $b = 2$ to 95.98s at $b = 4096$. However, this improvement in efficiency comes at the cost of reduced accuracy. While the accuracy remains stable (90.2% ∼ 90.6%) for $b \leq 256$, it degrades sharply for larger b, dropping to 53.3% at $b = 4096$, with a corresponding accuracy loss of 41.2%. These results highlight a trade-off between computational efficiency and model accuracy: smaller scale bases yield higher accuracy but incur longer runtime, while larger bases improve performance at the expense of inference accuracy.

Table 14 shows the results under the same scale base settings when modulus conversion (M.C.) is applied. The accuracy remains well-preserved for $b \leq 256$, but drops sharply when b becomes large (e.g., 61.3% at $b = 512$ and 53.3% at $b = 4096$), following a similar trade-off between accuracy and runtime as observed without modulus conversion.

H.4 Ablation Studies on Modulus Conversion Hyperparameters

Given that hyperparameter settings in modulus conversion can significantly affect model utility, we perform ablation studies across

Table 15: Effect of Bit Precision in Linear Layers on QNLI Accuracy for BERT-base (Plaintext: 90.6%)

f_{Linear}	19	18	17	16	15	14	13
Accuracy	89.1	89.1	89.1	89.1	89.1	88.3	88.3
Acc. Drop (%)	1.7	1.7	1.7	1.7	1.7	2.5	2.5

*In this evaluation, linear layers are computed over $\mathbb{Z}_{2^{64}}$, Softmax and Rsqrt over $\mathbb{Z}_{2^{32}}$ with 12-bit precision, and ReLU/GELU over $\mathbb{Z}_{2^{16}}$ with 6-bit precision.

Table 16: Effect of Bit Precision in Softmax/Rsqrt on QNLI Accuracy for BERT-base (Plaintext: 90.6%)

$f_{\text{Softmax/Rsqrt}}$	15	14	13	12	11	10	9
Accuracy	54.8	45.7	89.1	89.1	45.7	50.3	41.1
Acc. Drop (%)	39.5	49.6	1.7	1.7	49.6	44.5	54.6

*In this evaluation, Softmax and Rsqrt are computed over $\mathbb{Z}_{2^{32}}$, linear operations over $\mathbb{Z}_{2^{64}}$ with 16-bit precision, and ReLU/GELU over $\mathbb{Z}_{2^{16}}$ with 6-bit precision.

Table 17: Effect of Bit Precision in GELU/ReLU on QNLI Accuracy for BERT-base (Plaintext: 90.6%)

$f_{\text{GELU/ReLU}}$	9	8	7	6	5	4	3
Accuracy	52.6	53.3	88.3	89.1	87.6	88.3	88.3
Acc. Drop (%)	42.0	41.1	2.5	1.7	3.3	2.5	2.5

*In this evaluation, GELU and ReLU are computed over $\mathbb{Z}_{2^{16}}$, linear operations over $\mathbb{Z}_{2^{64}}$ with 16-bit precision, and Softmax and Rsqrt over $\mathbb{Z}_{2^{32}}$ with 12-bit precision.

several linear and nonlinear operations to validate the robustness of our design choices. Figure 5 illustrates all transformer layers involved in the modulus conversion process, categorized into three types: linear operations, ReLU/GELU activations, and Softmax/Rsqrt functions. Tables 15–17 report the accuracy of BERT-base on the QNLI task under varying bit precision configurations for each layer type.

Table 15 presents the effect of bit precision in linear operations on the QNLI accuracy of the BERT-base model, with a plaintext baseline of 90.6%. The results show that when the bit precision f_{Linear} is set to 15–19, the model maintains an accuracy of 89.1%, corresponding to a limited accuracy drop of 1.7%. However, reducing the precision to 14 or 13 bits results in a more noticeable accuracy degradation. These findings indicate that at least 15-bit precision in linear operations is needed to maintain acceptable accuracy in secure inference, while lower precision leads to non-negligible degradation.

Table 16 presents the impact of bit precision in Softmax and reciprocal square root (Rsqrt) operations on the QNLI accuracy of the BERT-base model. Precision settings of 12–13 bits yield 89.1% accuracy with a modest 1.7% drop, while lowering the precision to 11 bits or below leads to significant degradation. This stems from quantization limitations; for instance, $f = 11$ offers a minimum positive value of $2^{-11} \approx 0.00049$, which cannot represent smaller intermediate results (e.g., 0.0002), introducing rounding errors. Conversely, using 14–15 bits under $\mathbb{Z}_{2^{32}}$ fixed-point may also impair precision due to insufficient representability of relatively large values under the fixed-point setting in $\mathbb{Z}_{2^{32}}$. These results suggest that 12–13 bits offer a favorable trade-off.

Table 17 presents the impact of bit precision in GELU and ReLU activations on the QNLI accuracy of the BERT-base model. The optimal accuracy is achieved at a bit precision of $f_{\text{GELU/ReLU}} = 6$, while higher settings such as 8 or 9 result in significantly degraded performance. This is because, with a 16-bit fixed-point representation and $f \geq 8$, the representable data range is limited to approximately $[-128, 128)$, which fails to fully cover the actual input range of $[-64, 142)$ (as shown in Table 10), resulting in quantization errors that degrade accuracy. Interestingly, the accuracy under 3 or 4 bits remains relatively high despite large numerical deviations from the plaintext results. This is primarily because QNLI is a binary classification task, where prediction correctness depends only on the relative order of the two output logits. As long as the score ranking is preserved, even significant absolute errors may not alter the final classification.

I Security Pitfalls in Falcon's Maliciously Secure Multiplication Protocol

In the semi-honest setting, Falcon adopts replicated secret sharing to compute secure multiplication. Given two secret-shared values $\langle x \rangle$ and $\langle y \rangle$, each party locally computes $[[z]]_b = [[x]]_b[[y]]_b + [[x]]_{b+1}[[y]]_b + [[x]]_b[[y]]_{b+1}$ for $b \in \mathbb{Z}_3$, resulting in a $(3, 3)$-share of $z = xy$. This value is then reshared to obtain a $(3, 2)$-sharing. Since the values are represented as fixed-point integers over $\mathbb{Z}_{2^\ell}$, the product z doubles the precision, and the lower f bits must be truncated to maintain the correct scale. ABY3 [36] proposes a truncation protocol using a random masking strategy, where each party generates random shares $\langle r \rangle$ and $\langle r' \rangle = \langle r/2^f \rangle$, and reveals $\hat{z} = z + r$. The final truncated result is computed locally as $\hat{z}/2^f - \langle r' \rangle$, incurring at most one bit of error. ABY3 further optimizes the protocol by combining the truncation reveal and multiplication reshare steps, reducing the overall communication.

Under the malicious model, this optimization becomes problematic. An adversary can deviate arbitrarily during multiplication or truncation, making verification essential. Falcon attempts to detect such behavior by integrating a consistency check using a Beaver triple $(\langle a \rangle, \langle b \rangle, \langle c \rangle)$ satisfying $ab = c$. The parties compute $\rho = x - a$ and $\sigma = y - b$, reveal them via $\mathcal{F}_{\text{open}}$, and check whether

$$\delta = z/2^f - (\rho \cdot \sigma - \rho \cdot b - a \cdot \sigma - c)/2^f = 0.$$

However, evaluating this expression securely requires a maliciously secure truncation of the verification polynomial. Existing truncation protocols [36] incur communication overhead, which undermines the performance gain Falcon aims to achieve. On the other hand, local truncation is communication-free but introduces over-ring errors, making the verification result unreliable.

In the publicly available implementation of Falcon, all-zero triples are used in the check phase, effectively forcing the verification to pass by construction. While this avoids the need for truncation or additional communication, it compromises soundness: a malicious party can forge incorrect results without being detected. As a result, this optimization is insecure in practice and cannot be used in a genuinely malicious adversary model.