Iron: Private Inference on Transformers

Meng Hao[1*] **Hongwei Li**[1†] **Hanxiao Chen**[1] **Pengzhi Xing**[1] **Guowen Xu**[2] **Tianwei Zhang**[2]

[1]University of Electronic Science and Technology of China
[2]Nanyang Technological University
{menghao,hanxiao.chen,p.xing}@std.uestc.edu.cn
hongweili@uestc.edu.cn
{guowen.xu,tianwei.zhang}@ntu.edu.sg

Abstract

We initiate the study of private inference on Transformer-based models in the client-server setting, where clients have private inputs and servers hold proprietary models. Our main contribution is to provide several new secure protocols for matrix multiplication and complex non-linear functions like Softmax, GELU activations, and LayerNorm, which are critical components of Transformers. Specifically, we first propose a customized homomorphic encryption-based protocol for matrix multiplication that crucially relies on a novel compact packing technique. This design achieves $\sqrt{m}\times$ less communication (m is the number of rows of the output matrix) over the most efficient work. Second, we design efficient protocols for three non-linear functions via integrating advanced underlying protocols and specialized optimizations. Compared to the state-of-the-art protocols, our recipes reduce about half of the communication and computation overhead. Furthermore, all protocols are numerically precise, which preserve the model accuracy of plaintext. These techniques together allow us to implement Iron, an efficient Transformer-based private inference framework. Experiments conducted on several real-world datasets and models demonstrate that Iron achieves $3 \sim 14\times$ less communication and $3 \sim 11\times$ less runtime compared to the prior art.

1 Introduction

Transformer-based models [1–5] have realized tremendous success in the fields of natural language processing (NLP) and computer vision (CV) due to their strong representation capabilities. As a new neural network architecture, the Transformer [1] mainly utilizes the self-attention mechanism to compute representations without sequence-aligned recurrence or convolution. Following this work, a number of Transformer variants, such as BERT [2] and GPT [3] in NLP, ViT [4] and Swin Transformer [5] in CV, have achieved state-of-the-art performance on lots of real-world tasks.

The success of Transformers and other big models facilitates emerging inference services and applications [6, 7]. In particular, a service provider trains a complex model based on the Transformer, and deploys it as a paid inference service, e.g., machine translation and question answering. A client queries this service with his input samples and obtains the desired responses. Unfortunately, current inference systems suffer from serious privacy concerns [8]. On the one hand, clients need to send confidential inputs to the service provider, which could compromise the data privacy of these clients if the provider is untrusted. On the other hand, it is undesirable for the provider to distribute the proprietary Transformer-based model to clients, since he needs a large amount of data and computation resources to construct the model [9]. Therefore, there exists a gap between

[*]This work was done at NTU as a visiting student.
[†]Corresponding author

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

unprecedented performance and privacy constraints, which motivates our study of private Transformer inference.

Private inference aims to protect server's model weights from clients, while guaranteeing that the server learns no information about clients' private inputs. Recently, private inference on traditional neural networks (e.g., convolutional neural networks) have been approached by using secure 2-party computation (2PC) techniques [10–14]. However, due to essentially different structures, private Transformer inference brings several new challenges. First, *Transformer-based models use lots of high-dimensional matrix multiplications*, rather than matrix-vector multiplications widely studied in prior works. While we can straightforwardly extend prior matrix-vector multiplication protocols to our setting, unfortunately, even the most efficient design [14] incurs heavy communication because of interacting a large amount of ciphertexts. Second, *Transformer-based models use complex math functions like Softmax, GELU activations [15], and LayerNorm, in each block*, rather than crypto-friendly non-linear functions such as ReLU and Maxpool. Existing methods either use precision-impaired high-order polynomial approximations [16, 11] or only support limited math functions for specific scenarios [17]. Even worse, all of these approaches are computationally intensive and often require a large amount of communication (for more related works, please refer to Appendix A.5). To facilitate the widespread adoption of Transformer-based inference services in privacy-critical scenarios, designing efficient protocols for the above complex operations is of paramount importance.

In this paper, we design `Iron`, an efficient hybrid cryptographic framework for private Transformer inference without revealing any sensitive information about the server's model weights or clients' inputs. `Iron` contributes several new specialized protocols for the complicated operations in Transformers to alleviate the performance overhead. Specifically, we first propose a customized homomorphic encryption-based protocol for matrix multiplications. Our insight is to pack more plaintext inputs into a single ciphertext by devising a compact packing method, while preserving the functionality of matrix multiplication. Compared to the most efficient matrix-vector multiplication solution implemented in Cheetah [14], we can achieve $\sqrt{m}\times$ (m is the number of rows of the output matrix) improvement in terms of communication overhead, which is about $8\times$ reduction for various Transformer models. Second, we carefully design efficient protocols for Softmax, GELU, and LayerNorm. These protocols are built upon SIRNN [17], the state-of-the-art cryptographic framework for private inference on recurrent neural networks, and make several customized optimizations, such as reducing the overhead of exponentiation in Softmax and simplifying GELU and LayerNorm. These optimizations achieve $1.3 \sim 1.8\times$ less runtime and $1.4 \sim 1.8\times$ less communication on three non-linear functions. Furthermore, these protocols are numerically precise, which preserve the model accuracy of plaintext. We also give a formal security proof for our designed protocols to demonstrate the security guarantee.

Based on the above efficient components, we implement a private Transformer inference framework, `Iron`, and conduct end-to-end experiments with various BERT architectures [2] (BERT-Tiny, BERT-Medium, BERT-Base, and BERT-Large) on GLUE benchmarks [18]. Note that `Iron` is readily extended to other Transformer-based models (e.g., ViT) since they share very similar architectures and same operations. Experimental results show that `Iron` achieves $3 \sim 14\times$ less communication and $3 \sim 11\times$ less runtime costs over SIRNN on four BERT models. Moreover, compared with the general-purpose state-of-the-art framework MP-SPDZ [16], `Iron` has up to two orders of magnitude improvement in terms of both communication and computation efficiency.

A concurrent work [19] proposed a privacy-preserving Transformer inference with homomorphic encryption, called THE-X. Below, we illustrate some important differences in terms of protocol design and security. (1) Protocol design. Our work aims to design new efficient protocols for the complex operations of Transformer-based models, while orthogonal to ours, THE-X replaces them with crypto-friendly operations. For example, THE-X replaces GELUs with simpler operations, i.e., ReLUs, and Softmax with the combination of ReLU and polynomials. (2) Security. Our work achieves more rigorous privacy protection than THE-X. Specifically, our work uses homomorphic encryption and secret sharing techniques to hide private information (including intermediate results) of all layers. Such rigorous privacy guarantee is in line with recent state-of-the-art private inference works [14, 17]. However, in THE-X, the inputs of each non-linear layer are leaked to the client, which may causes severe privacy leakages in real-world applications [13]. Therefore, our work may be used to enhance the security of THE-X.

2 Preliminaries

2.1 Threat Model

As shown in the left part of Figure 1, `Iron` works in a general private inference scenario, where the server P_0 holds a Transformer-based model M with private weights w, while the client P_1 holds a private input x. Our framework enables the client to query the server's inference service and learn the output of the model on its input, i.e., $M(w, x)$. Same as prior works [14, 17], we consider an honest-but-curious adversary that passively corrupts either the server or the client, but not both. Such an adversary follows the protocol specification exactly, but may try to learn more information[3] than allowed (e.g., the model's weights or inference inputs) via analyzing the data it receives. In Appendix A.1.2, we give a more formal description of the threat model for security analysis.

2.2 Cryptographic Primitives

All our protocols are built on the 2-out-of-2 additive secret sharing (ASS) technique [21, 22] over the ring $\mathbb{Z}_{2^\ell}$, in which an ℓ-bit input x is split into two random shares $\langle x\rangle_0$, $\langle x\rangle_1$, held by P_0 and P_1, respectively, such that $x = \langle x\rangle_0 + \langle x\rangle_1 \bmod \mathbb{Z}_{2^\ell}$. When $\ell = 1$, i.e., over $\mathbb{Z}_2$, we use $\langle x\rangle^{\mathrm{B}}$ to denote boolean shares. In our protocols, we use $\langle x\rangle$ to denote that P_b holds $\langle x\rangle_b$ for $b \in \{0, 1\}$. The security [21] guarantees that given a share $\langle x\rangle_0$ or $\langle x\rangle_1$, the value of x is perfectly hidden. This secret-sharing property is maintained throughout our private inference scheme. ASS naturally supports linear operations without communication. For instance, to compute $cx + y$ with the constant c and secret shares $\langle x\rangle$ and $\langle y\rangle$, P_b can locally compute $\langle z\rangle_b = c\langle x\rangle_b + \langle y\rangle_b$, where $\langle z\rangle_0 + \langle z\rangle_1 = cx + y \bmod 2^\ell$. To achieve more functionalities under shared inputs, we require to invoke advanced homomorphic encryption or oblivious transfer techniques [22].

Notations. Let $x \nmid y$ means x is not a divisor of y. We use bold lower-case letters (e.g., $\mathbf{x}$) to represent vectors, and bold upper-case letters (e.g., $\mathbf{X}$) to denote matrices. Like prior works [17, 14], we encode inputs with the fixed-point representation denoted by Fix (refer to Appendix A.1.1 for details). Let $\mathbb{A}_{N,p}$ denote the set of integer polynomials $\mathbb{A}_{N,p} = \mathbb{Z}_p[x]/\left(x^N + 1\right)$. We use the circumflex of lower-case letters (e.g., $\hat{a}$) to represent a polynomial, and $\hat{a}[i]$ to denote the i-th coefficient of $\hat{a}$. Given polynomials $\hat{x}, \hat{y} \in \mathbb{A}_{N,p}$, the product $\hat{z} = \hat{x} \cdot \hat{y}$ over $\mathbb{A}_{N,p}$ is defined as

$$\hat{z}[i] = \sum_{0 \le j \le i} \hat{x}[j]\hat{y}[i-j] - \sum_{i < j < N} \hat{x}[j]\hat{y}[N-j+i] \bmod p. \tag{1}$$

Additively Homomorphic Encryption (AHE) [23, 24]. This encryption scheme additionally enables linearly homomorphic operations on ciphertexts. Specifically, an AHE scheme is a tuple of algorithms $\mathsf{AHE} = (\mathsf{KeyGen}; \mathsf{Enc}; \mathsf{Dec}; \mathsf{Eval})$ with the parameter $\{N, q, p\}$ and the following syntax: 1) $\mathsf{KeyGen}\left(1^k\right) \to (pk, sk)$: on input a security parameter κ, KeyGen is a randomized algorithm that outputs a public key $pk \in \mathbb{A}_{N,q}$ and a secret key $sk \in \mathbb{A}_{N,q}$. 2) $\mathsf{Enc}(pk, \hat{m}) \to \hat{c}$: the encryption algorithm Enc takes a plaintext polynomial $\hat{m} \in \mathbb{A}_{N,p}$ and encrypts it using pk into a ciphertext polynomial $\hat{c} \in \mathbb{A}_{N,q}$. 3) $\mathsf{Dec}(sk, \hat{c}) \to \hat{m}$: on input sk and a ciphertext $\hat{c}$, the (deterministic) decryption algorithm Dec recovers the plaintext message $\hat{m}$. 4) $\mathsf{Eval}(pk, \hat{c}_1, \hat{c}_2, \mathsf{func}) \to \hat{c}$: on input pk, two ciphertexts $\hat{c}_1, \hat{c}_2$ containing $\hat{m}_1, \hat{m}_2$, and a linear function func, Eval outputs a new ciphertext $\hat{c}$ encrypting $\mathsf{func}\,(\hat{m}_1, \hat{m}_2)$. Let $\boxplus, \boxminus, \boxtimes$ denote homomorphic addition, homomorphic subtraction and homomorphic multiplication with a plaintext, respectively. `Iron` builds the matrix multiplication protocol on the Brakerski-Fan-Vercauteren (BFV) scheme [25, 26], which is one of the state-of-the-art lattice-based homomorphic encryption solutions.

Oblivious Transfer. The 1-out-of-k oblivious transfer (OT) [27] is denoted by k-OT_ℓ, where one party is the sender with k messages $x_0, \ldots, x_{k-1} \in \{0, 1\}^\ell$ and the other party is the receiver with an index $i \in [k]$. The receiver learns x_i as the output, and the sender learns nothing. Additionally, we also use the 1-out-of-2 correlated OT, denoted by 2-COT_ℓ [28], which is defined as follows: the sender inputs a correlation $x \in \mathbb{Z}_{2^\ell}$, the receiver inputs a choice bit $i \in \{0, 1\}$, and the protocol outputs a random element $r \in \mathbb{Z}_{2^\ell}$ to the sender and $r + i \cdot x$ to the receiver. k-OT_ℓ and 2-COT_ℓ require $2\lambda + k\ell$ and $\lambda + \ell$ bits of communication, respectively, and are executed in 2 rounds. The OT protocols are widely used to build the underlying protocols [13, 17] in Figure 3 we rely on.

[3]Like prior works [17, 14], `Iron` does not hide the information that can be indirectly extracted from the inference results. Study of mitigation solutions (e.g., differential privacy [20]) is beyond the scope of this work.

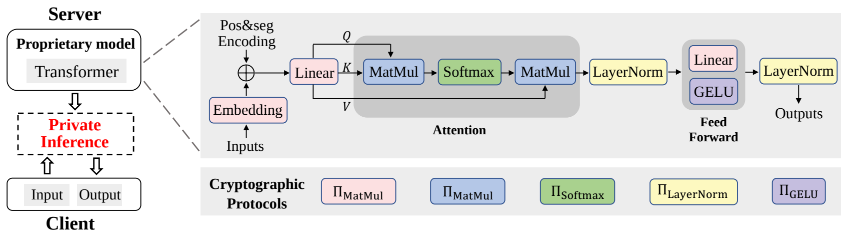

Figure 1: Overview of private Transformer inference

3 Overview

3.1 Transformer Architecture

The Transformer is an encoder-decoder architecture, where both parts have a similar structure. Hence, we mainly focus on the encoder below. The encoder is composed of a stack of identical blocks, each with two sub-layers, i.e., a multi-head self-attention mechanism and a feed-forward network, as shown in Figure 1. Besides, residual connection and layer normalization (LayerNorm) are employed around each of the two sub-layers. We describe a block of the encoder as follows.

Attention layer. An attention function can be described as mapping a query $\mathbf{X}_Q$ and a set of key-value pairs $(\mathbf{X}_K, \mathbf{X}_V)$ to a weighted sum of the values, where the weight is computed by a metric of the query with the corresponding key [1]. This function can be formalized as below:

$$\text{Attention}(\mathbf{X}_Q, \mathbf{X}_K, \mathbf{X}_V) = \text{Softmax}\left(\mathbf{X}_Q \mathbf{X}_K^T / \sqrt{d}\right) \mathbf{X}_V, \tag{2}$$

where $\mathbf{X}_Q, \mathbf{X}_K, \mathbf{X}_V$ are different linear projections of the input $\mathbf{X}$, i.e., $\mathbf{X}_Q = \mathbf{X}\mathbf{W}_Q, \mathbf{X}_K = \mathbf{X}\mathbf{W}_K, \mathbf{X}_V = \mathbf{X}\mathbf{W}_V$, and d is the dimension of representations. Multi-head attention extends the above mechanism to H parallel attention layers and is illustrated in Appendix A.1.3.

Feed-forward layer. A fully connected feed-forward layer consists of two linear transformations with a GELU activation in between, where GELU is the Gaussian Error Linear Unit function [15]. This layer can be represented as follows:

$$\text{FeedForward}(\mathbf{X}) = \text{GELU}(\mathbf{X}\mathbf{W}_1 + \mathbf{b}_1)\mathbf{W}_2 + \mathbf{b}_2. \tag{3}$$

In addition to the above encoder-decoder blocks, an embedding layer is employed at the beginning of the model to convert input tokens $\mathbf{X}_{\text{input}}$ to continuous feature vector representations. This is formulated as $\mathbf{X} = \mathbf{X}_{\text{input}}\mathbf{W}_E$, where $\mathbf{W}_E$ is the embedding lookup table.

3.2 Private Transformer Inference

According to the required crypographic operations, the layers of Transformers can be broken into two categories - *linear* and *non-linear*.

1) Linear layers: these include embedding, matrix multiplication in Attention, and fully-connected layer. All protocols of these operations rely crucially on the matrix multiplication protocol Π_{MatMul} in Section 4.1. It lies in the setting where P_0 and P_1 take as input the matrices $\mathbf{X}$ and $\mathbf{Y}$, and learn the sharings $\langle \mathbf{Z} \rangle_0$ and $\langle \mathbf{Z} \rangle_1$, respectively, such that $\mathbf{Z} = \mathbf{X}\mathbf{Y}$.

2) Non-linear layers: these consist of Softmax, GELU and LayerNorm. The non-linear operations are directly evaluated by exploiting our proposed protocols in Section 4.2, i.e., Π_{GELU}, Π_{Softmax}, $\Pi_{\mathsf{LayerNorm}}$. All of these take the additive shares $\langle \mathbf{X} \rangle$ as input, and output the additive shares $\langle \mathbf{Y} \rangle = \Pi_{\mathsf{Func}}(\langle \mathbf{X} \rangle)$, where $\mathsf{Func} \in \{\mathsf{GELU}, \mathsf{Softmax}, \mathsf{LayerNorm}\}$.

Overview of private inference. In the entire private inference process, we maintain the following invariant: P_0 and P_1 begin with additive shares of the input to the layer, and end with additive shares (over the same ring $\mathbb{Z}_{2^\ell}$) of the output of the layer after the protocol. This allows us to stitch protocols for arbitrary layers sequentially to obtain a secure computation scheme for any Transformer-based models. To clearly understand our scheme, we take an example of private inference between P_0 and P_1 on the embedding layer and the first encoder block, as shown in the right part of Figure 1. Here, our example uses the designed protocols as a black box, i.e., Π_{MatMul}, Π_{GELU}, Π_{Softmax}, and $\Pi_{\mathsf{LayerNorm}}$, and we discuss how to efficiently implement these protocols in Section 4.

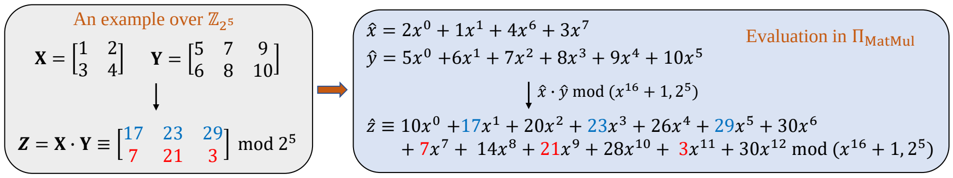

Figure 2: A toy example of our matrix multiplication protocol.

In the embedding layer, P_0 and P_1 take as inputs $\mathbf{W}_E$ and $\mathbf{X}_{\text{input}}$, and invoke the matrix multiplication protocol Π_{MatMul} to compute the input embedding $\mathbf{X}$. This is represented as $\langle \mathbf{X} \rangle = \Pi_{\mathsf{MatMul}}(\mathbf{X}_{\text{input}}, \mathbf{W}_E)$. In the attention layer, Π_{MatMul} is also invoked to generate the query, key and value matrices. For instance, $\langle \mathbf{X}_Q \rangle = \Pi_{\mathsf{MatMul}}(\langle \mathbf{X} \rangle_1, \mathbf{W}_Q) + \langle \mathbf{X} \rangle_0 \mathbf{W}_Q$, where $\langle \mathbf{X} \rangle_0 \mathbf{W}_Q$ can be computed locally by P_0. The same idea is used in the generation of $\langle \mathbf{X}_K \rangle$ and $\langle \mathbf{X}_V \rangle$. Then, we compute $\mathbf{X}_{QK} = \mathbf{X}_Q \mathbf{X}_K^T$ that requires two invocations of Π_{MatMul}. The formulation is $\langle \mathbf{X}_{QK} \rangle = \Pi_{\mathsf{MatMul}}(\langle \mathbf{X}_Q \rangle_0, \langle \mathbf{X}_K \rangle_1) + \Pi_{\mathsf{MatMul}}(\langle \mathbf{X}_Q \rangle_1, \langle \mathbf{X}_K \rangle_0) + \langle \mathbf{X}_Q \rangle_0 \langle \mathbf{X}_K \rangle_0^T + \langle \mathbf{X}_Q \rangle_1 \langle \mathbf{X}_K \rangle_1^T$, where the last two terms can be evaluated by P_0 and P_1 locally. After that, we evaluate $\langle \widetilde{\mathbf{X}}_{QK} \rangle = \Pi_{\mathsf{Softmax}}(\langle \mathbf{X}_{QK} \rangle / \sqrt{d})$, followed by computing $\langle \widetilde{\mathbf{Z}} \rangle = \langle \widetilde{\mathbf{X}}_{QK} \mathbf{X}_V \rangle$ by two invocations of Π_{MatMul}. Then, the two parties invoke the LayerNorm protocol to evaluate $\langle \mathbf{Z} \rangle = \Pi_{\mathsf{LayerNorm}}(\langle \widetilde{\mathbf{Z}} \rangle)$. In the feed-forward layer, they compute $\langle \widetilde{\mathbf{Y}}_1 \rangle = \Pi_{\mathsf{MatMul}}(\langle \mathbf{Z} \rangle_1, \mathbf{W}_1) + \langle \mathbf{Z} \rangle_0 \mathbf{W}_1$, followed by $\langle \widetilde{\mathbf{Y}} \rangle = \Pi_{\mathsf{GELU}}(\langle \widetilde{\mathbf{Y}}_1 \rangle)$. After that, they compute $\langle \widetilde{\mathbf{Y}}_2 \rangle = \Pi_{\mathsf{MatMul}}(\langle \widetilde{\mathbf{Y}} \rangle_1, \mathbf{W}_1) + \langle \widetilde{\mathbf{Y}} \rangle_0 \mathbf{W}_1$, where $\langle \widetilde{\mathbf{Y}} \rangle_0 \mathbf{W}_1$ can be computed locally by P_0. Finally, the evaluation of the encoder completes after computing $\langle \mathbf{Y} \rangle = \Pi_{\mathsf{LayerNorm}}(\langle \widetilde{\mathbf{Y}}_2 \rangle)$.

4 Supporting Protocols

4.1 Protocol for Matrix Multiplication

As shown in Section 1, naively extending well-studied matrix-vector multiplication protocols [29, 13, 14] to our matrix multiplication setting results in a significant communication overhead, mainly due to frequently interacting ciphertexts. In this work, we build a specialized matrix multiplication protocol on top of the most efficient protocol, Cheetah [14]. Recall that the plaintext of an AHE scheme is a polynomial, which can pack a large number of inputs to amortize the overhead [29, 13]. The key contribution of our proposed protocol is a more compact input packing approach.

Our starting point is that polynomial multiplication implies vector inner product, if we arrange the coefficients properly [14]. As shown in Equation 1, when multiplying two polynomials of degree-(N-1), the (N-1)-th coefficient of the resulting polynomial is the inner product of the two coefficient vectors in opposite orders. By using an appropriate arrangement of coefficients, this idea can be extended to our matrix multiplications, since they consist of a set of inner products. To this end, we give the definitions of two input packing functions $\pi_{\mathsf{L}} : \mathbb{Z}_{2^\ell}^{m \times n} \to \mathbb{A}_{N,2^\ell}$ and $\pi_{\mathsf{R}} : \mathbb{Z}_{2^\ell}^{n \times k} \to \mathbb{A}_{N,2^\ell}$ as follows:

$$\begin{aligned} \hat{x} &= \pi_{\mathsf{L}}(\mathbf{X}), \text{ s.t.}, \hat{x}\left[i \cdot n \cdot k + (n-1) - j\right] = \mathbf{X}[i,j], \text{ for } i \in [m], j \in [n] \\ \hat{y} &= \pi_{\mathsf{R}}(\mathbf{Y}), \text{ s.t.}, \hat{y}\left[j \cdot n + i\right] = \mathbf{Y}[i,j], \text{ for } i \in [n], j \in [k] \end{aligned}$$

where all other coefficients of $\hat{x}$ and $\hat{y}$ are set to 0. Multiplication of polynomials $\hat{z} = \hat{x} \cdot \hat{y}$ directly gives the result of matrix multiplication $\mathbf{Z} = \mathbf{X}\mathbf{Y} \mod 2^\ell$ in some of $\hat{z}$'s coefficients. We formalize this process as below, and give a toy example in Figure 2.

Theorem 4.1. *Assuming $mnk \leq N$ and given two polynomials $\hat{x} = \pi_L(\mathbf{X})$ and $\hat{y} = \pi_R(\mathbf{Y})$, the matrix multiplication $\mathbf{Z} = \mathbf{X}\mathbf{Y} \mod 2^\ell$ can be evaluated via the product $\hat{z} = \hat{x} \cdot \hat{y}$, where $\mathbf{Z}[i,j]$ is computed in $\hat{z}\left[i \cdot n \cdot k + (j+1) \cdot n - 1\right]$ for $i \in [m]$ and $j \in [k]$.*

We defer the proof to Appendix A.2.1. When $mnk > N$, we first partition the matrices $\mathbf{X}, \mathbf{Y}$ into sub-matrices of $m_w \times n_w$ and $n_w \times k_w$ elements, respectively, such that $m_w n_w k_w \leq N$. Zero-padding is required when $m_w \nmid m$, $n_w \nmid n$ or $k_w \nmid k$. The protocol is shown in Algorithm 1.

Complexity and security analysis. For complexity, totally, two parties interact $\frac{k}{k_w}(\frac{m}{m_w} + \frac{n}{n_w})$ ciphertexts, and operate with $O(mnk/N)$ homomorphic additions and multiplications. We formalize the selection of parameters m_w, n_w, k_w as an optimization problem, to minimize the communication cost. Through our analysis in Appendix A.2.2, in general scenarios, our method theoretically achieves $\sqrt{m}$ communication improvement over Cheetah [14]. Notably, when $mnk \le N$, we reduce the communcation cost by a factor of m, since Cheetah encodes each row of the matrix into a ciphertext. Besides, the security proof is shown in Appendix A.2.3.

Algorithm 1 Secure Matrix Multiplication Protocol

Input: P_0 holds $\mathbf{X} \in \mathbf{Z}_{2^\ell}^{m\times n}$, and P_1 holds $\mathbf{Y} \in \mathbb{Z}_{2^\ell}^{n\times k}$.

Output: P_0 and P_1 get $\langle \mathbf{Z}\rangle_0, \langle \mathbf{Z}\rangle_1 \in \mathbb{Z}_{2^\ell}^{m\times k}$, respectively, where $\mathbf{Z} = \mathbf{XY}$.

1: P_0, P_1 compute the partition window size $0 < m_{\mathrm{w}} \le m, 0 < n_{\mathrm{w}} \le n$ and $0 < k_{\mathrm{w}} \le k$ such that $m_{\mathrm{w}} n_{\mathrm{w}} k_{\mathrm{w}} \le N$, and set $n' = \lceil n/n_{\mathrm{w}} \rceil$, $m' = \lceil m/m_{\mathrm{w}} \rceil$, and $k' = \lceil k/k_{\mathrm{w}} \rceil$.
2: P_1 partitions the matrix $\mathbf{Y}$ into block matrices $\mathbf{Y}_{\beta,\gamma} \in \mathbb{Z}_{2^\ell}^{n_{\mathrm{w}}\times k_{\mathrm{w}}}$ for $\beta \in [n']$ and $\gamma \in [k']$.
3: P_1 encodes the matrices to polynomials $\hat{y}_{\beta,\gamma} = \pi_R(\mathbf{Y}_{\beta,\gamma})$ for $\beta \in [n']$ and $\gamma \in [k']$. Then P_1 sends to P_0 the ciphertexts $\{\mathrm{CT}_{\beta,\gamma} = \mathsf{Enc}(\hat{y}_{\beta,\gamma})\}$
4: P_0 partitions the matrix $\mathbf{X}$ into block matrices $\mathbf{X}_{\alpha,\beta} \in \mathbb{Z}_{2^\ell}^{m_{\mathrm{w}}\times n_{\mathrm{w}}}$. P_0 samples $\mathbf{R} \in \mathbb{Z}_{2^\ell}^{m\times k}$ uniformly at random and partitions the matrix $\mathbf{R}$ into $\mathbf{R}_{\alpha,\gamma} \in \mathbb{Z}_{2^\ell}^{m_{\mathrm{w}}\times k_{\mathrm{w}}}$.
5: P_0 encodes the two block matrices to polynomials: 1) $\hat{x}_{\alpha,\beta} = \pi_L(\mathbf{X}_{\alpha,\beta})$ for $\alpha \in [m']$ and $\beta \in [n']$, and 2) for $\alpha \in [m']$ and $\gamma \in [k']$, encoding $\mathbf{R}_{\alpha,\gamma}$ into $\hat{r}_{\alpha,\gamma}$ according to Theorem 4.1.
6: On receiving the ciphertexts $\{\mathrm{CT}_{\beta,\gamma}\}$ from P_1, P_0 operates $\mathrm{CT}'_{\alpha,\gamma} = \boxplus_{\beta\in[m']}(\hat{x}_{\alpha,\beta} \boxtimes \mathrm{CT}_{\beta,\gamma}) \boxminus \hat{r}_{\alpha,\gamma}$ for $\alpha \in [m']$ and $\gamma \in [k']$. Then P_0 sends to P_1 the ciphertexts $\left\{\mathrm{CT}'_{\alpha,\gamma}\right\}$.
7: P_0 outputs $\mathbf{R} \bmod 2^\ell$ as the share $\langle \mathbf{Z}\rangle_0$.
8: On receiving the ciphertexts $\left\{\mathrm{CT}'_{\alpha,\gamma}\right\}$ from P_0, P_1 computes $\langle \hat{z}_{\alpha,\gamma}\rangle_1 = \mathsf{Dec}(\mathrm{CT}'_{\alpha,\gamma})$ that are decoded to $\langle \mathbf{Z}\rangle_1$ using the method in Theorem 4.1.

4.2 Protocols for Non-linear Functions

Our non-linear protocols rely on several underlying protocols from the state-of-the-art works [13, 17]. In Figure 3, we enumerate the inputs and outputs of these protocols and then use them as a black box (See Appendix A.3.2 for details). On this basis, we provide efficient protocols for Softmax, GELU, and LayerNorm with specific optimizations. Additional details and security analysis are shown in Appendices A.3.3 and A.3.4.

Multiply $\Pi_{\mathrm{Mul_{OT}}}$	**Compare** Π_{CMP}	**NegExp** Π_{nExp}	**RecipSqrt** Π_{rSqrt}	**Recip** Π_{Recip}
Input:	Input:	Input:	Input:	Input:
• P_b: $\langle x\rangle_b$, $\langle y\rangle_b$	• P_b: $\langle x\rangle_b$	• P_b: $\langle x\rangle_b$	• P_b: $\langle x\rangle_b$	• P_b: $\langle x\rangle_b$
Output:	Output:	Output:	Output:	Output:
• P_b: $\langle z\rangle_b$	• P_b: $\langle z\rangle_b^B$	• P_b: $\langle z\rangle_b$	• P_b: $\langle z\rangle_b$	• P_b: $\langle z\rangle_b$
s.t. $z = xy$	s.t. $z = 1\{x > 0\}$	s.t. $z = e^x, x<0$	s.t. $z = \frac{1}{\sqrt{x}}, x>\epsilon$	s.t. $z = 1/x$

Figure 3: Underlying protocols from [13, 17]

4.2.1 Softmax

To evaluate attention layers, we need an efficient protocol to compute Softmax on secret-shared values. In particular, for a vector $\mathbf{x} \in \mathbb{Z}_{2^\ell}^d$, the Softmax function is denoted as $\mathsf{Softmax}_i(\mathbf{x}) = e^{x_i} / \sum_{j\in[d]} e^{x_j}$ for $i \in [d]$. The main challenge is to efficiently compute the underlying exponential function. Following the idea from [11, 30], we first normalize the input vector by $\mathbf{x} - \max_{i\in[d]} x_i$, which is always negative, and then invoke the existing exponential protocol Π_{nExp} in Figure 3 that only evaluates exponentiation on negative inputs. A simple analysis shows $\mathrm{softmax}(\mathbf{x} - \max_{i\in[d]} x_i)$ is equal to $\mathrm{softmax}(\mathbf{x})$. We evaluate $\max$ using a tree-reduction protocol, denoted by $\Pi_{\max}$. Specifically, we arrange the vector $\mathbf{x} \in \mathbb{Z}_{2^\ell}^d$ into a 2-ary tree with the depth of $\log d$, and evaluate the tree in a top-down fashion [31]. In each comparison of two secret-shared elements x_i and x_j, we reduce it to the invocations of Π_{CMP} and $\Pi_{\mathrm{Mul_{OT}}}$, i.e., $\max(x_i, x_j) = \Pi_{\mathrm{Mul_{OT}}}(x_i - x_j, \Pi_{\mathrm{CMP}}(x_i - x_j)) + x_j$. With the above insight, the Softmax protocol is detailed in Algorithm 2.

SIRNN [17] also provides a solution for the generic exponential protocol by extending Π_{nExp}. The idea is that the exponential of x equals $\frac{1}{\mathsf{nExp}(-x)}$ if $x \ge 0$, and $\mathrm{nExp}(x)$ otherwise, where nExp is the same as the exponential function except for negative inputs. Compared to our solution, realizing the softmax function with this generic exponential protocol additionally requires d calls to the reciprocal protocol Π_{Recip} and multiplication protocol $\Pi_{\text{MUT}_{\text{OT}}}$. Besides, compared with the generic library MP-SPDZ [16] that provides native support for exponentiation, our protocols achieve orders of magnitude improvement as shown in Section 5.2.

Algorithm 2 Secure Softmax Protocol

Input: P_0, P_1 hold $\langle \mathbf{x}\rangle_0 \in \mathbb{Z}^d_{2^\ell}$, $\langle \mathbf{x}\rangle_1 \in \mathbb{Z}^d_{2^\ell}$, respectively.
Output: P_0, P_1 get $\langle \mathbf{y}\rangle_0 \in \mathbb{Z}^d_{2^\ell}$, $\langle \mathbf{y}\rangle_1 \in \mathbb{Z}^d_{2^\ell}$, respectively, where $\mathbf{y} = \mathsf{Softmax}(\mathbf{x})$.
1: P_0, P_1 invoke $\Pi_{\text{max}}(\mathbf{x})$ to compute $\langle \mathsf{max}(\mathbf{x})\rangle$, where $\mathsf{max}(\mathbf{x}) = \max_{i\in[d]} x_i$.
2: For $i \in [d]$, P_0, P_1 invoke Π_{nExp} on input $\langle \bar{x}_i\rangle$, and learn $\langle e^{\bar{x}_i}\rangle$, where $\bar{x}_i = x_i - \mathsf{max}(\mathbf{x})$.
3: P_0, P_1 invoke Π_{Recip} with inputs $\langle \sum_{i\in[d]} e^{\bar{x}_i}\rangle$ and learn $\langle 1/\sum_{i\in[d]} e^{\bar{x}_i}\rangle$.
4: For $i \in [d]$, P_0, P_1 invoke $\Pi_{\text{Mul}_{\text{OT}}}$ with inputs $\langle 1/\sum_{i\in[d]} e^{\bar{x}_i}\rangle$ and $\langle e^{\bar{x}_i}\rangle$, and set outputs as $\langle y_i\rangle$.

4.2.2 GELU

Rather than crypto-friendly ReLU [13], Transformer-based models utilize GELU activations [15], which can be represented as $\mathsf{GELU}(x) = 0.5x\left(1 + \mathsf{Tanh}\left[\sqrt{2/\pi}\,(x + 0.044715x^3)\right]\right)$. The complete protocol is shown in Algorithm 3, in which we provide two insights to reduce the cost of GELU. First, we present an optimized protocol for the square of a secret-shared input $\langle x\rangle$. This relies on the observation: $x^2 = \langle x\rangle_0^2 + \langle x\rangle_1^2 + 2\langle x\rangle_0\langle x\rangle_1$, where the first two terms can be locally computed by P_0 and P_1. We only invoke OT protocols to compute the cross term $2\langle x\rangle_0\langle x\rangle_1$, and the optimized overhead is half that of the multiplication protocol $\Pi_{\text{Mul}_{\text{OT}}}$.

Second, we further optimize the evaluation of $\mathsf{Tanh}(x) = \frac{e^{2x}-1}{e^{2x}+1}$. We observe that the sign of x is equal to the sign of $\mathsf{Tanh}(x)$, which allows us to leverage the negative exponential protocol Π_{nExp} almost for free. At a high level, our Tanh protocol first learns the sign of the input x, and then evaluates Tanh on the negative input $\bar{x}$ with the constraint $|\bar{x}| = |x|$. Finally, the protocol generates the real output $\mathsf{Tanh}(x)$ that equals $\mathsf{Tanh}(\bar{x})$ if $x \le 0$, and $-\mathsf{Tanh}(\bar{x})$ otherwise. Our leaner Tanh protocol is given in Algorithm 5 of Appendix A.3.1. SIRNN [17] recently proposed the most efficient Tanh protocol with the insight of $\mathsf{Tanh}(x) = 2\mathsf{Sigmoid}(2x) - 1$, where $\mathsf{Sigmoid}(x) = \frac{1}{1+e^{-x}}$. However, the evaluation of Sigmoid uses the same idea as the general exponential protocol, as shown in Section 4.2.1. Compared with the protocol in SIRNN, our recipe for Tanh saves one invocation of the reciprocal protocol Π_{Recip} and multiplication protocol $\Pi_{\text{Mul}_{\text{OT}}}$.

Algorithm 3 Secure GELU Protocol

Input: P_0, P_1 hold $\langle x\rangle_0 \in \mathbb{Z}_{2^\ell}$, $\langle x\rangle_1 \in \mathbb{Z}_{2^\ell}$, respectively.
Output: P_0, P_1 get $\langle y\rangle_0 \in \mathbb{Z}_{2^\ell}$, $\langle y\rangle_1 \in \mathbb{Z}_{2^\ell}$, respectively, where $y = \mathsf{GELU}(x)$.
1: P_0, P_1 invoke $\Pi_{\text{Mul}_{\text{OT}}}$ with inputs $\langle x\rangle$, and set their outputs as $\langle z\rangle = \mathrm{Fix}(\sqrt{2/\pi})(\langle x\rangle + \mathrm{Fix}(0.044715)\langle x\rangle^3)$
2: P_0, P_1 invoke Π_{Tanh} with inputs $\langle z\rangle$, and set their outputs as $\langle \mathsf{Tanh}(z)\rangle$.
3: P_0, P_1 invoke $\Pi_{\text{Mul}_{\text{OT}}}$ with inputs $\langle \mathrm{Fix}(0.5)x\rangle$ and $\langle \mathrm{Fix}(1) + \mathsf{Tanh}(z)\rangle$, and output $\langle y\rangle$.

4.2.3 LayerNorm

For a vector $\mathbf{x} \in \mathbb{Z}^d_{2^\ell}$, the LayerNorm function is denoted by $\mathsf{LayerNorm}_i(\mathbf{x}) = \gamma(x_i - \mu)/\sigma + \beta$ for $i \in [d]$, where $\mu = \sum_{i\in[d]} x_i/d$ and $\sigma = \sqrt{\sum_{i\in[d]}(x_i - \mu)^2}$. In contrast to batch normalization (BN) in CNNs that is evaluated for free [14], LayerNorm requires multiplication and reciprocal square root operations. In our implementation, we observe that the multiplication dominates the overhead of LayerNorm. To address this issue, we adopt the same idea in GELU to compute the square of $x_i - \mu$, which saves half of the communication and computation costs.

Besides, inspired by the optimization in BN [14], we apply a LayerNorm merge technique to further reduce the overhead. Specifically, we make the observation that the weights γ and β of the LayerNorm layer are already known by P_0. As a result, P_0 first multiplies the scale factor γ with the weights of

Table 1: Comparing the runtime (sec) and communication (MB) costs of our matrix multiplication and non-linear protocols with SOTA

Matrix Multiplication							Non-Linear Protocols						
Methods	Dims=(32, 8, 16)		(128, 64, 128)		(128, 768, 768)		Methods	Softmax		LayerNorm		GELU	
	Time	Comm.	Time	Comm.	Time	Comm.		Time	Comm.	Time	Comm.	Time	Comm.
Ours	0.006	0.11	0.066	1.74	1.71	15.45	**Ours**	4.78	206.265	2.34	102.435	0.30	10.07
Cheetah	0.16 (26×)	2.79 (25×)	0.77 (11×)	14.78 (8×)	6.10 (3×)	134.37 (8×)	**SIRNN**	7.95 (1.7×)	347.71 (1.7×)	4.16 (1.8×)	184.42 (1.8×)	0.38 (1.3×)	14.07 (1.4×)
SIRNN	0.04 (6×)	1.34 (12×)	1.59 (23×)	70.08 (40×)	110.33 (64×)	4920.08 (318×)	**MP-SPDZ**	297.75 (62×)	172,837 (837×)	202.75 (86×)	101,642 (992×)	15.34 (51×)	7,908.69 (785×)

the next linear layer. After invoking the linear layer protocol on the scaled weights, P_0 adds the shift weight σ to his additive share. The optimized protocol for LayerNorm is presented in Algorithm 4.

Algorithm 4 Secure LayerNorm Protocol

Input: P_0, P_1 hold $\langle \mathbf{x} \rangle_0 \in \mathbb{Z}_{2^\ell}^d$, $\langle \mathbf{x} \rangle_1 \in \mathbb{Z}_{2^\ell}^d$, respectively.
Output: P_0, P_1 get $\langle \mathbf{y} \rangle_0$, $\langle \mathbf{y} \rangle_1$, respectively, where $\mathbf{y} = \text{LayerNorm}(\mathbf{x})$.
1: For $i \in [d]$, P_0, P_1 invoke $\Pi_{\text{Mul}_{\text{OT}}}$ to compute $\langle (x_i - \mu)^2 \rangle$, where $\mu = \sum_{i \in [d]} x_i / d$.
2: P_0, P_1 invoke Π_{rSqrt} with inputs $\langle \sum_{i \in [d]} (x_i - \mu)^2 \rangle$ to learn output $\langle \frac{1}{\sigma} \rangle$.
3: For $i \in [d]$, P_0, P_1 invoke $\Pi_{\text{Mul}_{\text{OT}}}$ with inputs $\langle \frac{1}{\sigma} \rangle$ and $\langle x_i - \mu \rangle$, and set outputs as $\langle y_i \rangle$.

5 Evaluation

5.1 Experimental Setup

Implementation. `Iron` is built on top of the SEAL library [32] and the EMP toolkit [33] in C++. We also use the EzPC framework [34]. This framework compiles a high-level TensorFlow code to secure computation protocols, which are then executed by our designed cryptographic backends. Like [17], we simulate a LAN network setting, where the bandwidth is 377 MBps and the echo latency is 0.8ms. All the following experiments are performed on AWS c5.9xlarge instances with Intel Xeon 8000 series CPUs at 3.6GHz.

Datasets and Models. We evaluate `Iron` on four NLP models from [35, 36]: BERT-Tiny, BERT-Medium, BERT-Base and BERT-Large. These models are parameterized by three hyper-parameters: the number of blocks, the dimension of representations and the number of input tokens (refer to Appendix A.4.1 for the hyper-parameters of these models). We train the models for four NLP tasks over the datasets of the Stanford Sentiment Treebank (SST-2), the Microsoft Research Paraphrase Corpus (MRPC), the Multi-Genre Natural Language Inference Corpus (MNLI) and the Stanford Question Answering Dataset (QNLI) from GLUE benchmarks [18].

5.2 Microbenchmark Evaluation

Matrix multiplication. In the left part of Table 1, we compare the performance of the proposed matrix multiplication protocol with the state-of-the-art counterparts of Cheetah [14] and SIRNN [17]. For fairness, we follow Cheetah for the parameter setup of homomorphic encryption. Compared with Cheetah, our rumtime is $3 \sim 26\times$ faster, and our communication cost is $8 \sim 25\times$ lower, depending on the input size. Notably, for small-size matrices (e.g., ones with 32×8 and 8×16 elements), our protocol only requires 0.11 MB communication and 6 ms runtime, while Cheetah achieves the computation with 2.79 MB and 160 ms. The reason is that our protocol encrypts the whole matrix into a single ciphertext, while Cheetah could only encrypt each row into a ciphertext, totally m ciphertexts (m is the number of rows of the output matrix). Moreover, compared with SIRNN that implements the most efficient OT-based matrix multiplication protocol, our protocol incurs up to two orders of magnitude less communication and an order of magnitude less runtime.

Non-linear functions. The right part of Table 1 shows the comparison of our Softmax, GELU and LayerNorm protocols with the generic MP-SPDZ framework [16] and the state-of-the-art SIRNN library [17] for math functions. It is worth noting that we implement some functions that these

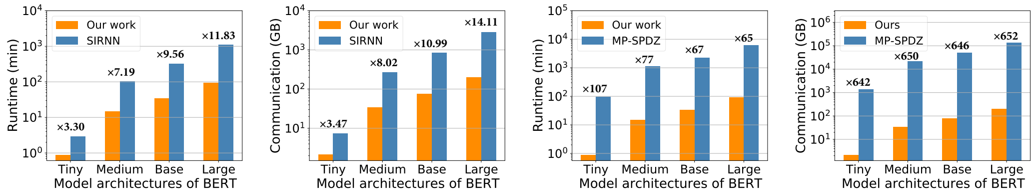

Figure 4: End-to-end comparisons with SIRNN and MP-SPDZ

frameworks did not provide before. In particular, GELU and LayerNorm are implemented in MP-SPDZ by calling its built-in functions for tanh, square root, and reciprocal, while we add Softmax and GELU protocols in SIRNN utilizing sigmoid and exponent functions. As shown in this table, our protocols are orders of magnitude better than those of MP-SPDZ, in terms of both runtime and communication. In particular, for the communication cost, our protocols achieve $785 \sim 993\times$ improvement. Moreover, while our protocols are built upon the underlying protocols from SIRNN, we also achieve $1.3 \sim 1.8\times$ lower runtime and $1.4 \sim 1.8\times$ lower communication due to our customized optimizations. Such an improvement gives us significant savings in communication complexity, since the non-linear layer protocols dominate the overall overhead, as described below.

5.3 End-to-end Inference Evaluation

Comparison with prior methods. In the left part of Figure 4, we evaluate our protocols on 4 BERT models compared with SIRNN [17]. It is observed that our rumtime is $3.3 \sim 11.83\times$ faster than that of SIRNN, and our communication cost is $3.47 \sim 14.11\times$ lower over four models. Moreover, our performance gains scale up as the model size grows. This is because our protocols achieve better amortized overhead when processing large-scale evaluations. We also compare the end-to-end private inference with MP-SPDZ in Figure 4. The results show that our protocols are orders of magnitude better, in terms of both time and communication costs. This is because specialized protocols are more communication efficient than generic alternatives, which is also observed by SIRNN.

Performance breakdown. In Figure 5, we present the runtime and communication breakdown of `Iron` on four BERT models. For clarity, we just report the result of one encoder. Recall that our private inference can be divided into linear protocols and non-linear protocols. For linear protocols, we observe that as the model size increases, the proportion of communication overhead remains approximately constant, accounting for $12 \sim 13\%$. This shows that our compact ciphertext encoding method effectively reduces the size of communication. For non-linear functions, as the model size increases, the proportion of computation overhead gradually decreases, from 84% in BERT-Tiny to 76% in BERT-Large. The savings come from amortizing the communication and computation costs by packing data from large tensors. Despite such advantages, the main bottleneck of our work is the communication overhead of non-linear layers. However, it is an open problem to solve the communication issue while maintaining the model accuracy.

Accuracy comparison with plaintext. In the left part of Figure 6, we show the accuacy of plaintext (float-point) and `Iron` (fixed-point) on the BERT-Tiny model. We observe that the accuracy achieved by `Iron` matches the accuracy of the plaintext TensorFlow code. Specifically, the accuracy loss does not exceed 0.3% over all datasets, and surprisingly, `Iron` exceeds the plaintext baseline on MNLI by 0.85%. Similar results also appear in private CNN inference [13]. Such accuracy advantages experimentally validate that our protocols are numerically precise. Moreover, in the right part of Figure 6, we also compare the accuracy with the plaintext baseline, as the fractional scale varies on MRPC. We observe that `Iron` with the scale of 12 exactly matches the accuracy of the plaintext model. The accuracy loss is lower than 1% when the scale ≥ 6. This conclusion is in line with the prior work [13], namely that neural networks can tolerate stochastic fault behavior [37].

6 Discussion

We discuss possible solutions to further improve the efficiency of private inference on Transformers. They are compatible with our proposed protocols and hence can be directly integrated into our framework.

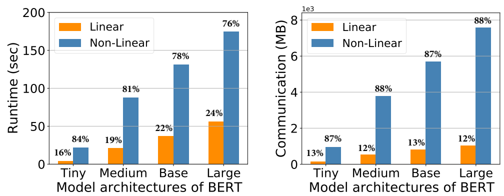

Figure 5: Performance breakdown on BERT.

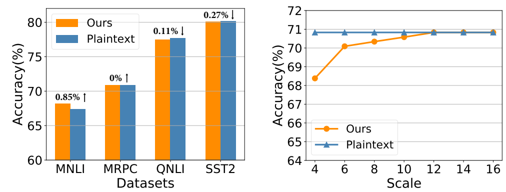

Figure 6: Accuracy comparison with plaintext

Pushing expensive operations into an offline phase. Our framework is readily extended to the preprocessing model, including an offline client-input independent phase and an online input-dependent phase. This paradigm has been instantiated in private CNN inference [38, 8], and the results show that about 99% of the cryptographic overhead can be moved to the offline phase. We briefly outline how to build our protocols with this paradigm. For linear operations, we can generate in advance Beaver's triple in the matrix form [22, 38] using our matrix multiplication protocol. In the online phase, these triples are consumed, and it additionally requires non-cryptographic operations and plaintext communication. For non-linear layers, almost all operations crucially rely on the OT protocols, which can be pre-generated in the offline phase [39, 14]. Like linear protocols, the online phase is cheap.

Using mixed-bitwidths in private inference. `Iron` works with a uniform bitwidth, which is required to be large enough to accommodate all intermediate values. While effectively avoiding integer overflows, our protocols may cause intensive communication, since the performance of non-linear operators depends critically on bitwidths. Taking inspiration from [17, 40], one possible optimization is to employ non-uniform (mixed) bitwidths, operate in low bitwidths and switch to high bitwidths only when necessary. To this end, the mixed-precision model should be generated with proper compilers by using techiniques like quantization [41, 42]. With mixed-bitwidths fixed-point models, our protocols can be integrated seamlessly.

Applying orthogonal model optimizations. We can improve performance by simplifying the model architecture such as model pruning and advanced neural architecture search [43, 44], which have been commonly adopted in private CNN inference [12, 45, 37]. Like prior works [12, 45, 37] that find and tailor models to the requirements of private inference, we can define a proper search space with the goal of decreasing the overhead, e.g., substituting costly Softmax and GELU functions, or reducing the matrices' dimension in matrix multiplication. As mentioned in Section 1, the concurrent work, THE-X [19], has explored replacing complex math functions with HE-friendly alternatives while achieving comparable accuracy.

7 Conclusion

We propose `Iron`, an efficient cryptographic framework for private Tramsformer inference. Specifically, we carefully design a new encoding method for optimizing homomorphic encryption-based matrix multiplication. Further, we devise several communication-efficient non-linear protocols like Softmax, LayerNorm and GELU by integrating advanced secret-sharing primitives. Experimental results show that `Iron` outperforms prior art by up to one order of magnitude in terms of computation and communication overheads. We believe that our novel protocols would help advance the practical instantiations of private Transformer inference.

Acknowledgment

The authors would like to thank the anonymous reviewers for their insightful comments. This work is supported by the Key-Area Research and Development Program of Guangdong Province under Grant 2020B0101360001, National Natural Science Foundation of China under Grants 62020106013, 61972454, and 61802051, Sichuan Science and Technology Program under Grants 2020JDTD0007 and 2020YFG0298, the Fundamental Research Funds for Chinese Central Universities under Grant ZYGX2020ZB027, and Singapore Ministry of Education (MOE) AcRF Tier 2 MOE-T2EP20121-0006.

References

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of NeurIPS*, 2017.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NeurIPS*, 2018.

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

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

[5] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of ICCV*, pages 10012–10022, 2021.

[6] Clarifai general image embedding model. URL `https://www.clarifai.com/models/general-image-embedding`.

[7] Openai api. URL `https://openai.com/blog/openai-api`.

[8] Ryan Lehmkuhl, Pratyush Mishra, Akshayaram Srinivasan, and Raluca Ada Popa. Muse: Secure inference resilient to malicious clients. In *Proceedings of USENIX Security*, pages 2201–2218, 2021.

[9] Yupei Liu, Jinyuan Jia, Hongbin Liu, and Neil Zhenqiang Gong. Stolenencoder: Stealing pre-trained encoders. *arXiv:2201.05889*, 2022.

[10] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *Proceedings of ICML*, pages 201–210, 2016.

[11] Brian Knott, Shobha Venkataraman, Awni Hannun, Shubho Sengupta, Mark Ibrahim, and Laurens van der Maaten. Crypten: Secure multi-party computation meets machine learning. In *Proceedings of NeurIPS*, 2021.

[12] Zahra Ghodsi, Akshaj Kumar Veldanda, Brandon Reagen, and Siddharth Garg. Cryptonas: Private inference on a relu budget. In *Proceedings of NeurIPS*, pages 16961–16971, 2020.

[13] Deevashwer Rathee, Mayank Rathee, Nishant Kumar, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. Cryptflow2: Practical 2-party secure inference. In *Proceedings of ACM CCS*, pages 325–342, 2020.

[14] Zhicong Huang, Wen-jie Lu, Cheng Hong, and Jiansheng Ding. Cheetah: Lean and fast secure two-party deep neural network inference. In *Proceedings of USENIX Security*, 2022.

[15] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

[16] Marcel Keller. Mp-spdz: A versatile framework for multi-party computation. In *Proceedings of ACM CCS*, pages 1575–1590, 2020.

[17] Deevashwer Rathee, Mayank Rathee, Rahul Kranti Kiran Goli, Divya Gupta, Rahul Sharma, Nishanth Chandran, and Aseem Rastogi. Sirnn: A math library for secure rnn inference. In *Proceedings of IEEE S&P*, pages 1003–1020, 2021.

[18] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of ICLR*, pages 535–548, 2018.

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

[20] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of ACM CCS*, pages 308–318, 2016.

[21] Ronald Cramer, Ivan Bjerre Damgård, et al. *Secure multiparty computation*. Cambridge University Press, 2015.

[22] Daniel Demmler, Thomas Schneider, and Michael Zohner. Aby-a framework for efficient mixed-protocol secure two-party computation. In *Proceedings of NDSS*, 2015.

[23] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory*, 31(4):469–472, 1985.

[24] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Proceedings of Eurocrypt*, pages 223–238, 1999.

[25] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. In *Proceedings of Crypto*, pages 868–886, 2012.

[26] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *Cryptology ePrint Archive*, 2012.

[27] Vladimir Kolesnikov and Ranjit Kumaresan. Improved ot extension for transferring short secrets. In *Proceedings of Crypto*, pages 54–70, 2013.

[28] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer and extensions for faster secure computation. In *Proceedings of ACM CCS*, pages 535–548, 2013.

[29] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. Gazelle: A low latency framework for secure neural network inference. In *Proceedings of USENIX Security*, pages 1651–1669, 2018.

[30] Sijun Tan, Brian Knott, Yuan Tian, and David J Wu. Cryptgpu: Fast privacy-preserving machine learning on the gpu. In *Proceedings of IEEE S&P*, 2021.

[31] Théo Ryffel, Pierre Tholoniat, David Pointcheval, and Francis Bach. Ariann: Low-interaction privacy-preserving deep learning via function secret sharing. In *Proceedings of PETS*, pages 291–316, 2022.

[32] Seal. URL https://github.com/microsoft/SEAL.

[33] emp-toolkit. URL https://github.com/emp-toolkit.

[34] Ezpc. URL https://github.com/mpc-msri/EzPC.

[35] Bert. URL https://github.com/google-research/bert.

[36] Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Well-read students learn better: On the importance of pre-training compact models. *arXiv:1908.08962*, 2019.

[37] Zahra Ghodsi, Nandan Kumar Jha, Brandon Reagen, and Siddharth Garg. Circa: Stochastic relus for private deep learning. In *Proceedings of NeurIPS*, 2021.

[38] Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. Delphi: A cryptographic inference service for neural networks. In *Proceedings of USENIX Security*, pages 2505–2522, 2020.

[39] M Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M Songhori, Thomas Schneider, and Farinaz Koushanfar. Chameleon: A hybrid secure computation framework for machine learning applications. In *Proceedings of AsiaCCS*, pages 707–721, 2018.

[40] Siam Umar Hussain, Mojan Javaheripi, Mohammad Samragh, and Farinaz Koushanfar. Coinn: Crypto/ml codesign for oblivious inference via neural networks. In *Proceedings of ACM CCS*, pages 3266–3281, 2021.

[41] Sehoon Kim, Amir Gholami, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. I-bert: Integer-only bert quantization. In *Proceedings of ICML*, pages 5506–5518, 2021.

[42] Sheng Shen, Zhen Dong, Jiayu Ye, Linjian Ma, Zhewei Yao, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. Q-bert: Hessian based ultra low precision quantization of bert. In *Proceedings of AAAI*, pages 8815–8821, 2020.

[43] François Lagunas, Ella Charlaix, Victor Sanh, and Alexander M Rush. Block pruning for faster transformers. In *Proceedings of EMNLP*, 2021.

[44] Paul Michel, Omer Levy, and Graham Neubig. Are sixteen heads really better than one? In *Proceedings of NeurIPS*, 2019.

[45] Nandan Kumar Jha, Zahra Ghodsi, Siddharth Garg, and Brandon Reagen. Deepreduce: Relu reduction for fast private inference. In *Proceedings of ICML*, pages 4839–4849, 2021.

[46] Payman Mohassel and Yupeng Zhang. Secureml: A system for scalable privacy-preserving machine learning. In *Proceedings of IEEE S&P*, pages 19–38, 2017.

[47] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings of FOCS*, pages 136–145, 2001.

[48] Robert E Goldschmidt. *Applications of division by convergence*. PhD thesis, MIT, 1964.

[49] Xiaoqian Jiang, Miran Kim, Kristin Lauter, and Yongsoo Song. Secure outsourced matrix computation and application to neural networks. In *Proceedings ACM CCS*, pages 1209–1222, 2018.

[50] Fabian Boemer, Anamaria Costache, Rosario Cammarota, and Casimir Wierzynski. ngraph-he2: A high-throughput framework for neural network inference on encrypted data. In *Proceedings of the 7th ACM Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, pages 45–56, 2019.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [No]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [No]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [No]
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Appendix

A.1 More Details on Preliminaries

A.1.1 Fixed-Point Encoding

Same as other neural networks, Transformer-based models use floating-point arithmetic, however cryptographic protocols operate on integers. Therefore, we require a float-to-integer conversion [46, 30, 17] to represent a floating-point number $x \in \mathbb{Q}$ into the ring $\mathbb{Z}_{2^\ell}$. Specifically, we first encode it as a fixed-point number, which is parametrized by a scale parameter s that determines the fractional precision. Then, we embed the fixed-point representation into the ring with 2's complement representation. The formulation is $a = \lfloor 2^s \times x \rfloor \in \mathbb{Z}_{2^\ell}$ if x is a non-negative number, and $a = 2^\ell - \lfloor 2^s \times |x| \rfloor \in \mathbb{Z}_{2^\ell}$ if x is a negative number, where s is the length of the (binary) fractional bits and ℓ is the bitwidth of the secret sharing ring. Unless otherwise stated, similar as prior works [14], we set the bitwidth as 37 and the scale as 12 in the fixed-point encoding. Because of the use of the above fixed-point encoding, after multiplication, the scale of the output is $2s$. Therefore, a truncation operation is required to reduce scale. We use the Π_{Trunc} protocol proposed in [13] and improved by [17], which leads to faithful implementation of fixed-point arithmetic. For simplicity, we omit this operation in our protocol description. The overhead of truncation in `Iron` will be reported in Table 4.

A.1.2 Formal Description of the Threat Model

Same as prior private inference works [13, 14, 17], the security of `Iron` is provably provided in the simulation paradigm against static honest-but-curious probabilistic polynomial-time (PPT) adversaries. Namely, a PPT adversary $\mathcal{A}$ passively corrupts either the server or the client at the beginning of the protocol and honestly follows the protocol specification. In the simulation paradigm, two worlds are defined: a real world where the server and the client perform the protocol according to the specification in the presence of $\mathcal{A}$, and an ideal world where the parties send their inputs to a trusted dealer (also called functionality) that executes the evaluation faithfully. The executions in both worlds are coordinated by the environment Env, which chooses the inputs to the parties and plays the role of a distinguisher between the real and ideal executions. It is required that for any adversary, the real-world distribution is computationally indistinguishable to the ideal-world distribution. Some of our protocols invoke sub-protocols and we describe them using the hybrid model. This is similar to a real execution, except that sub-protocols are replaced by the invocations of the corresponding functionality instances. We recap the definition of a private inference protocol in [38] [14] as below.

Definition A.1. A protocol Π_{PI} between the server having as input a model M with weights w and the client having as input a sample x is a private inference protocol against honest-but-curious adversaries if it satisfies the following guarantees: 1) *Correctness*: on every model weights w and every input sample x, the output of the client at the end of the protocol is the correct inference $M(w, x)$. 2) *Security*: For a corrupted client, there exists an efficient simulator Sim_C such that $\mathsf{View}_C^{\text{PI}}$ is computationally indistinguishable $\mathsf{Sim}_C(\mathrm{output})$, where $\mathsf{View}_C^{\text{PI}}$ is the view of the client in the execution of Π_{PI} and output denotes the output of the inference. Similarly, for a corrupted server, there exists an efficient simulator Sim_S such that $\mathsf{View}_S^{\text{PI}}$ is computationally indistinguishable Sim_S.

Notice that the honest-but-curious security proof of `Iron` according to the above definition will follow trivially from sequential composibility of individual sub-protocols [47, 13, 30]. Hence, we require to provide a security proof for our matrix multiplication and non-linear protocols. We refer to Section A.2.3 and A.3.4 for the sub-protocols' security analysis.

A.1.3 Multi-Head Attention

Instead of performing a single attention function, existing Transformer-based models [1, 2] follow a multi-head attention variant, which can be represented as

$$\text{MultiHeadAtten} = \text{Concat}\left(\text{Attention}(\mathbf{X}_{Q,j}, \mathbf{X}_{K,j}, \mathbf{X}_{V,j}), j \in [H]\right) \mathbf{W}_O, \tag{4}$$

where H is the number of heads, and $\mathbf{X}_{Q,j} = \mathbf{X}_Q \mathbf{W}_{Q,j}$, $\mathbf{X}_{K,j} = \mathbf{X}_K \mathbf{W}_{K,j}$, $\mathbf{X}_{V,j} = \mathbf{X}_V \mathbf{W}_{V,j}$ for $j \in [H]$. The main intuition is that multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions [1].

A.2 More Details on the Matrix Multiplication Protocol

A.2.1 Correction Proof of the Matrix Multiplication Protocol

Proof. For each $i \in [m]$ and $j \in [k]$, we write $\epsilon_{i,j} = i \cdot n \cdot k + (j+1) \cdot n - 1$ for simplicity. Based on the description of Section 4.1, for $\epsilon_{i,j} \geq nk$, $\hat{y}[\epsilon_{i,j}] = 0$ holds. Therefore, given the definition of Equation 1, we have $\hat{z}\,[\epsilon_{i,j}] = \sum_{0 \leq \mu < n} \hat{x}[i \cdot n \cdot k + (n-1) - \mu]\hat{y}[j \cdot n + \mu] = \sum_{0 \leq \mu < n} \mathbf{X}[i, \mu]\mathbf{Y}[\mu, j]$, which is exactly $\mathbf{Z}[i][j]$. □

A.2.2 Optimal Parameters Selection in the Matrix Multiplication Protocol

As shown in Section 4.1, the matrix multiplication $\mathbf{X}_{m \times n} \cdot \mathbf{Y}_{n \times k}$ requires the communication of $\frac{m}{m_w}(\frac{n}{n_w} + \frac{k}{k_w})$ ciphertexts. To minimize the ciphertext communication cost, we formalize the selection of the parameters m_w, n_w, k_w as an optimization problem, i.e., $\min_{\{m_w, n_w, k_w\}} \frac{m}{m_w}(\frac{n}{n_w} + \frac{k}{k_w})$, s.t., $m_w n_w k_w \leq N$, where m, n, k, N are constants. Given the difficulty of solving the above multivariate optimization, we figure out a sub-optimal solution. To this end, we first fix $m_w = m$[4], and hence the constraint is transformed to $n_w k_w \leq \frac{N}{m}$. Correspondingly, the new optimization problem is $\min_{\{n_w, k_w\}} \frac{n}{n_w} + \frac{k}{k_w}$, s.t., $n_w k_w \leq \frac{N}{m}$. Then, the following holds:

$$\frac{n}{n_w} + \frac{k}{k_w} = 1/\frac{n_w}{n} + 1/\frac{k_w}{k} \geq \frac{2\sqrt{nk}}{\sqrt{n_w k_w}} \geq \frac{2\sqrt{mnk}}{\sqrt{N}}, \tag{5}$$

where the first inequality is due to $\frac{1}{a} + \frac{1}{b} \geq \frac{2}{\sqrt{ab}}$, and the second inequality comes from $n_w k_w \leq \frac{N}{m}$. As a result, assuming $m_w = m$, the optimal communication size is $\frac{2\sqrt{mnk}}{\sqrt{N}}$ ciphertexts. Notably, it may not have the optimal analytical solution, because the variables must be discrete positive integers, rather than real number. To achieve minimal communication in our implementation, like Cheetah [14], we use the exhaustive testing approach on all the results satisfying the constraint to find the optimal matrix partitioning strategy. Note that this is quite fast due to the small search range.

Communication comparison with Cheetah. As shown in Section 3.1 of [14], the Cheetah's communication overhead is $m(\frac{n}{n_w} + \frac{k}{k_w})$ ciphertexts with the constraint $n_w k_w \leq N$. By using the similar analysis as above, we can obtain its optimal solution, i.e., $\frac{2m\sqrt{nk}}{\sqrt{N}}$. Therefore, we obtain a $\sqrt{m}\times$ communication improvement in an ideal situation (i.e., the optimal analytical integer solution exists).

A.2.3 Security Proof of the Matrix Multiplication Protocol

Theorem A.2. *In presence of an honest-but-curious adversary, the protocol Π_{MatMul} in Algorithm 1 realizes the matrix multiplication functionality, in which P_0 and P_1 take as inputs the matrices $\mathbf{X}$ and $\mathbf{Y}$, and learn the secret shares $\langle \mathbf{Z} \rangle_0$ and $\langle \mathbf{Z} \rangle_1$, respectively, such that $\mathbf{Z} = \mathbf{XY}$.*

Proof. The correctness of Theorem A.2 is directly derived from Theorem 4.1. We below focus on the protocol's security when the server or the client is corrupted. Our security proof follows the simulation paradigm defined in Section A.1.2. In this paradigm we need show that the real-world distribution is computationally indistinguishable to the simulated distribution by the simulator Sim in the ideal world.

Proof of indistinguishability with the corrupted server. The server's view of $\mathsf{View}_S^{\mathrm{MatMul}}$ consists of ciphertexts $\mathrm{CT}_{\beta,\gamma}$. The simulator Sim_S for this view can be constructed as follows:

Given the access to public parameters, Sim_S *outputs ciphertexts* $\widetilde{\mathrm{CT}}_{\beta,\gamma} = \mathsf{Enc}(0)$ *to the server.*

The security against the corrupted server is directly reduced to the semantic security of the underlying homomorphic encryption scheme. Thus we have that the simulated view $\mathsf{View}_S^{\mathrm{MatMul}}$ in the ideal world is computationally indistinguishable from the real-world distribution of the protocol.

Proof of indistinguishability with corrupted clients. The client's view of $\mathsf{View}_C^{\mathrm{MatMul}}$ consists of ciphertexts $\mathrm{CT}'_{\alpha,\gamma}$, and the decryption of these ciphertexts, i.e., $\langle \mathbf{Z} \rangle_1$. The simulator Sim_C for this view can be constructed as follows:

[4] In our setting, m is always less than N.

On receiving the ciphertexts $\mathrm{CT}_{\alpha,\gamma}$ from the client, Sim_C samples uniformly random polynomials $\hat{r}_{\alpha,\gamma} \in \mathbb{A}_{N,2^\ell}$, and computes $\widetilde{\mathrm{CT}}'_{\alpha,\gamma} = \mathsf{Enc}(\hat{r}_{\alpha,\gamma})$. Given the access to the output, Sim_C outputs $\widetilde{\mathrm{CT}}'_{\alpha,\gamma}$ to the client.

Similarly, the ciphertexts $\widetilde{\mathrm{CT}}'_{\alpha,\gamma}$ are computationally indistinguishable from $\widetilde{\mathrm{CT}}_{\alpha,\gamma}$ due to the semantic security. Besides, the values of $\langle \mathbf{Z} \rangle_1$ distribute uniformly in $\mathbb{Z}_{2^\ell}$, which is exactly the same distribution of values in $\hat{r}_{\alpha,\gamma}$. Thus we have that $\mathrm{View}_C^{\mathrm{MatMul}}$ is computationally indistinguishable from the real-world distribution of the protocol. $\square$

A.3 More Details on Non-linear Protocols

A.3.1 Tanh

Algorithm 5 Secure Tanh Protocol

Input: P_0, P_1 hold $\langle x \rangle_0 \in \mathbb{Z}_{2^\ell}$, $\langle x \rangle_1 \in \mathbb{Z}_{2^\ell}$, respectively.
Output: P_0, P_1 get $\langle y \rangle_0 \in \mathbb{Z}_{2^\ell}$, $\langle y \rangle_1 \in \mathbb{Z}_{2^\ell}$, respectively, where $y = \mathsf{Tanh}(x)$.

1: P_0, P_1 parse $\langle x \rangle_0 = \mathrm{msb}_0 \| a_0$ and $\langle x \rangle_1 = \mathrm{msb}_1 \| a_1$, and invoke Π_{CMP} to learn $\langle \mathrm{carry} \rangle_b^B$, where $\mathrm{carry} = 1\{a_0 + a_1 > 2^{\ell-1} - 1\}$, and the inputs are $2^{\ell-1} - a_0 - 1$ and a_1 from P_0 and P_1, respectively. For $b \in \{0,1\}$, P_b outputs $\langle \mathrm{MSB}(x) \rangle_b^B = \langle \mathrm{carry} \rangle_b^B \oplus \mathrm{msb}_b$.
2: P_0, P_1 invoke $\Pi_{\mathrm{Mul_{OT}}}$ with inputs $\langle 2x \rangle$ and $\langle \mathrm{MSB}(x) \rangle^B$, and set outputs as $\langle \bar{x} \rangle$, where $\bar{x} = 2x \cdot \mathrm{MSB}(x) - x$ that is always negative with the constraint of $|\bar{x}| = |x|$.
3: P_0, P_1 invoke Π_{nExp} with negative inputs $\langle 2\bar{x} \rangle$ and learn $\langle e^{2\bar{x}} \rangle$.
4: P_0, P_1 invoke Π_{Recip} with inputs $\langle e^{2\bar{x}} \rangle$, and learns $\langle \bar{y} \rangle$ where $\bar{y} = 1 - \frac{2}{e^{2\bar{x}}+1}$.
5: P_0, P_1 invoke an instance of $\Pi_{\mathrm{Mul_{OT}}}$ on input $\langle \mathrm{MSB}(x) \rangle^B$ and $\langle \bar{y} \rangle$, and learn $\langle y \rangle$, where $y = \bar{y} + \mathrm{MSB}(x) \cdot (-2\bar{y})$.

We present an optimized protocol for Tanh in Algorithm 5, which builds over the protocol used in [17]. Our optimization relies on the observation: when evaluating on x, the sign of $\mathsf{Tanh}(x)$ is the same as that of x. Compared with the protocol in [17], our alternative saves one invocation of the reciprocal protocol Π_{Recip} and multiplication protocol $\Pi_{\mathrm{Mul_{OT}}}$.

A.3.2 Underlying Protocols from [17, 13]

We outline the underlying protocols from existing works [17, 13], and the detailed implementation could be found in the corresponding papers.

- Multiplication ($\mathrm{Mul_{OT}}$): The OT-based multiplication protocol $\Pi_{\mathrm{Mul_{OT}}}$ takes as input $\langle x \rangle \in \{0,1\}^\ell$ and $\langle y \rangle \in \{0,1\}^\ell$ and returns $\langle z \rangle$ such that $z = xy$. The well-known technique is proposed by ABY [22] and optimized in [13]. Currently, the optimal solution invokes 2-COT_i for $i \in \{1, \ldots, \ell\}$ requiring communication $\ell(\lambda + \frac{\ell+1}{2})$ bits with 2 rounds that is equivalent to ℓ instances of $\mathsf{COT}_{\frac{\ell+1}{2}}$. A variant of multiplication is multiplexer[5], which takes as input $\langle x \rangle^B \in \{0,1\}$ and $\langle y \rangle \in \{0,1\}^\ell$ and outputs $\langle z \rangle \in \{0,1\}^\ell$ such that $z = y$ if $x = 1$ and 0 otherwise. The multiplexer protocol Π_{Mux} can be realized by 2 parallel calls of 2-COT_ℓ with communication $2(\lambda + \ell)$ bits and 2 rounds.
- Comparison (CMP): The comparison protocol Π_{CMP} takes as input $\langle x \rangle \in \{0,1\}^\ell$, and returns $\langle z \rangle$ such that $z = 1\{x \geq 0\}$. Recently, [13] gave an efficient protocol for Π_{CMP} with communication less than $\lambda\ell + 14\ell$ bits with $\log \ell$ rounds.
- Exponential on negative inputs (nExp): The exponential protocol Π_{nExp} takes as input $x \in \{0,1\}^\ell$, where $x \leq 0$, and returns $\langle z \rangle$ such that $z = e^x$. The protocol is proposed by [17], which invokes digit decomposition to generate small-length inputs and integrates the OT-based lookup table technique to compute exponential on the small-length inputs.

[5] In the protocol description, we treat these two types of multiplication indiscriminately, but we implement them using different techniques.

- Reciprocal of Square Root (rSqrt): The protocol of square root's reciprocal, Π_{rSqrt}, takes as input $x \in \{0,1\}^\ell$ and returns $\langle z \rangle \in \{0,1\}^\ell$ such that $z = \frac{1}{\sqrt{x}}$. [17] proposed the state-of-the-art OT-based protocol, which relies on the Goldschmidt's algorithm [48] that iterates on an initial approximation.
- Reciprocal (Recip): The reciprocal protocol Π_{Recip} takes as input $x \in \{0,1\}^\ell$ and returns $\langle z \rangle \in \{0,1\}^\ell$ such that $z = 1/x$. The most efficient implementation is proposed in [17] with a similar idea as the protocol Π_{rSqrt}.

A.3.3 Extra Optimization

MSB-known protocol optimization. As pointed out by [17], 2PC protocols could be designed in a far more efficient way when the MSB of the inputs are known. In particular, we optimize truncation and OT-based multiplication in this case. For example, the MSB-known truncation protocol requires $O(\lambda(s+3))$ communication, instead of $O(\lambda(\ell+3))$, where λ is the security parameter, ℓ is the bit length of the secret sharing ring and s is the fractional scale.

We elaborate the optimization for the GELU protocol in Algorithm 3, and the same idea can also be used in the Softmax and LayerNorm protocols. For GELU, we fist compute the shares of MSB(x), instead of calculating it in the latter Tanh protocol, and then in the following sub-process, we use this knowledge to reduce overhead. Moreover, we further observe that the GELU protocol implies more MSB-known operations if proper computation order is considered. We rewrite the GELU formulation as below:

$$\mathsf{GELU}(x) = 0.5\left(x + x\mathsf{Tanh}\left[\sqrt{2/\pi}x\left(1+0.044715x^2\right)\right]\right). \tag{6}$$

We observe that $1+0.044715x^2$ and $x\mathsf{Tanh}\left[\sqrt{2/\pi}x\left(1+0.044715x^2\right)\right]$ are always non-negative, where the latter holds because the sign x equals to that of $\mathsf{Tanh}\left[\sqrt{2/\pi}x\left(1+0.044715x^2\right)\right]$.

A.3.4 Security Proof of Non-linear Protocols

Similar as the security of protocols in [13, 17], our protocols directly follow in the hybrid model. In particular, the security of the Softmax and GELU protocols are easy to see in $(\text{CMP}, \text{nExp}, \text{Recip}, \text{MUL}_{\text{OT}})$-hybrid. Besides, the security of the LayerNorm protocol follows in $(\text{rSqrt}, \text{MUL}_{\text{OT}})$-hybrid.

A.4 More Details on Experimental Evaluation

A.4.1 Additional Experimental Setup

Table 2: Models and hyper-parameters

Models	#Params	Hyper-parameters		
		b	d	t
BERT-Tiny	4.4M	2	128	128
BERT-Medium	41.7M	8	512	128
BERT-Base	110.1M	12	768	128
BERT-Large	340M	24	1024	128

Table 3: Datasets and tasks description

Datasets	#Train	#Test	Task	Domain
SST-2	67K	872	Single-sentence 2-classification	Movie reviews
MRPC	3.7K	408	Sentence pair 2-class paraphrase	News
MNLI	393K	2K	Sentence pair 3-class inference	Misc.
QNLI	105K	2K	Sentence pair 2-class inference	Wikipedia

We evaluation `Iron` on 4 widely used pre-trained BERT models with different hyper-parameters, as shown in Table 2. We denote the number of blocks as b, the dimension of representations as d, and the number of input tokens as t. We always fix the number of self-attention heads to $d/64$ and the size of feed-forward features to $4d$. The end-task models are obtained by stacking a linear classifier on top of the Transformer architectures with fine-tuning. We follow the default fine-tuning hyper-parameters in [35], e.g., batch size 32, learning rate 2×10^{-5} and epoch 3. Notice that any hyper-parameters optimization during the training phase is compatible with our scheme. Besides, we use 4 datasets for different tasks from GLUE [18], which include the Stanford Sentiment Treebank

Table 4: Detailed performance breakdown of our protocols on BERT

Models	Metrics	MatMul	Truncation	GELU	Softmax	LayerNorm	Total
BERT-Tiny	Runtime (Sec)	1.54	2.61	14.65	5.04	2.40	26.24
	Comm. (MB)	29.99	108.66	642.38	214.01	99.02	1094.07
BERT-Medium	Runtime (Sec)	11.25	9.70	58.79	20.24	8.56	108.53
	Comm. (MB)	132.00	404.63	2565.53	856.05	374.53	4332.74
BERT-Base	Runtime (Sec)	22.12	14.87	88.08	30.31	13.05	168.43
	Comm. (MB)	197.68	626.94	3848.30	1284.08	575.23	6532.23
BERT-Large	Runtime (Sec)	36.66	19.50	117.45	40.43	16.65	230.70
	Comm. (MB)	240.05	809.25	5131.06	1712.10	733.83	8626.28

(SST-2), the Microsoft Research Paraphrase Corpus (MRPC), the Multi-Genre Natural Language Inference Corpus (MNLI) and the Stanford Question Answering Dataset (QNLI). Table 3 shows the datasets' details.

A.4.2 Additional Experimental Results

Detailed performance breakdown on BERT. In Table 4, we show the detailed performance breakdown including the communication and computation costs of matrix multiplication, truncation, GELU, softmax and layer normalization. The most expensive non-linear operation is GELU due to its huge number. For example, for each layer of BERT-Base, the number of GELU is 393,216. We also observe that the our linear operation is lightweight in terms of communication.

A.5 Related Works

Recently a quantity of works have designed customized protocols for performing private inference on neural networks, especially convolutional neural networks. These special-purpose protocols improve the computation and communication costs and generally fall into two categories: linear protocols and non-linear protocols. We briefly discuss the progress as below.

Linear protocols. Gazelle [29] proposed an optimized AHE-based linear algebra kernels, which support matrix-vector multiplication and convolution operations. The main innovation is a new packing method to minimize the expensive rotation operations, which is the critical component for the linear algebra. After that, CrypTFlow2 [13] proposed a comprehensive implementation for linear layers, based on both AHE-based and OT-based solution[6]. For the AHE-based solution, they use the protocol from Gazelle, and employ several optimizations such as parallelization and reducing ciphertext size. They observe the AHE-based solution performs better than the OT-based counterpart, especially for large-scale models. More recently, Huang et al. presented Cheetah [14], the most efficient AHE-based linear layer protocols, including matrix-vector multiplication and convolution operations. The improvement comes from a novel input packing technique, which is rotation-free and hence efficient. Moreover, the packing method is compatible with secret sharing in a ring. This support further benefits the subsequent non-linear operations [14]. However, existing protocols are only optimized for matrix-vector multiplication, rather than general matrix multiplication `Iron` relies on. As mentioned earlier, directly extending the most efficient matrix-vector multiplication protocol still causes prohibitively high communication overhead. Therefore, to approach such communication issue, we propose a special-purpose protocol for matrix multiplication, based on the state-of-the-art protocol in Cheetah.

Notice that different from the setting of 2PC private inference, [49] proposed a private *outsourced* inference scheme, which stands for encrypted data and encrypted model. To this end, [49] designs a homomorphic matrix multiplication protocol for multiplying two encrypted matrices, which is fundamentally different our homomorphic multiplication with a plaintext. As a result, it requires to invoke costly homomorphic multiplication and rotation operations, which are about $2 \sim 20\times$ more expensive than the underlying operations of our protocol (refer to Table 9 of [50]).

Non-linear protocols. Although earlier works [29, 38] implemented non-linear function evaluation with garbled circuits (GC), CrypTFlow2 [13] found that these GC-based solutions result in high

[6]The OT-based linear protocol is also used in SIRNN [17]

communication overhead. Therefore, the authors designed optimized OT-based protocols, such as truncation and comparison. These protocols achieve state-of-the-art performance, and can be seen as general underlying building blocks for the design of advanced protocols [17]. Despite the efficiency advantage for truncation and comparison, these protocols can not support complex functions, like exponent in Transformers. Actually, the state-of-the-art general-purpose framework, MP-SPDZ, provides comprehensive protocols. However, as shown in SIRNN [17], the protocols implemented with MP-SPDZ are communication-heavy and computation-intensive. Therefore, SIRNN [17] proposed special-purpose protocols for exponent on negative inputs, sigmoid and reciprocal of square root, which achieve orders of magnitude improvement over MP-SPDZ, both in terms of runtime and communication. However, these functions are still insufficient to implement a private inference framework on Transformers. Therefore, on the basis of the building blocks in [13, 17], we propose new protocols for three non-linear functions that are critical components for Transformers, and make several specialized optimizations. Note that, [11] also proposed a softmax protocol but in an unrealistic setting, where except the client and the server, a trusted third party (TTP) exists and assists to generate correlated randomness to accelerate protocol evaluation. However, in a practical application, it is difficult to have a completely TTP [13, 14]. In contrast to [11], our setting lies in a practical client-server setting, without any unrealistic assumptions.