Iron: Private Inference on Transformers

Anonymous Author(s)
Affiliation
Address
email

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) and at least $3\times$ performance improvement over the state-of-the-art. Second, we design efficient protocols for three non-linear functions via integrating advanced underlying protocols and specialized optimizations. Compared to the most efficient 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, the *first* 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 achieved 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, 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 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

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

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 frequently interacting 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 state-of-the-art 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 are never supported previously. We also 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 the *first* 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.

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[1] 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 [20, 21] 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,

[1] 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 [19]) is beyond the scope of this work.

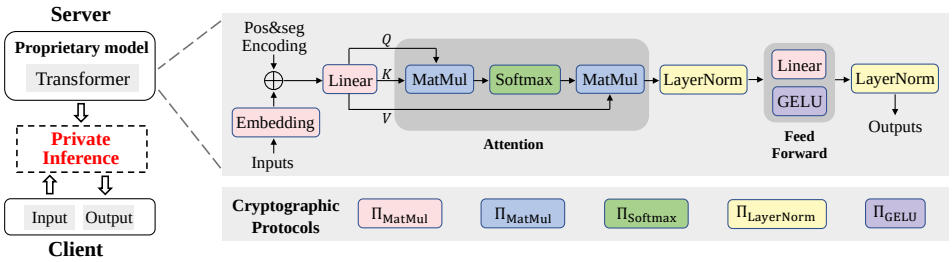

Figure 1: Overview of private Transformer inference

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 [20] 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 and oblivious transfer techniques, which are illustrated as follows.

Notations. 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 $\mathbb{A}_{N,2^\ell}$ denote the set of integer polynomials $\mathbb{A}_{N,2^\ell} = \mathbb{Z}_{2^\ell}[x]/\left(x^N + 1\right)$. 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 $\hat{x}, \hat{y} \in \mathbb{A}_{N,2^\ell}$, the product $\hat{z} = \hat{x} \cdot \hat{y}$ over $\mathbb{A}_{N,2^\ell}$ is defined as

$$\hat{z}[i] = \sum_{0 \le j \le i} \hat{x}[j]\hat{y}[i-j] - \sum_{i < j < N} \hat{x}[j]\hat{y}[N+j-i] \bmod 2^\ell. \tag{1}$$

Linearly Homomorphic Encryption (LHE) [22, 23]. This encryption scheme additionally enables linearly homomorphic operations on the ciphertexts. An LHE scheme is a tuple of algorithms $\mathsf{LHE} = (\mathsf{KeyGen}; \mathsf{Enc}; \mathsf{Dec}; \mathsf{Eval})$ with the parameter $\{N, q, p = 2^\ell\}$ 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 $\mathsf{func}\left(\hat{m}_1, \hat{m}_2\right)$. Note that $\boxplus, \boxtimes \in \mathsf{func}$ denote homomorphic addition and homomorphic multiplication with a constant polynomial, respectively. `Iron` builds the matrix multiplication protocol on the Brakerski-Fan-Vercauteren (BFV) scheme [24, 25], which is one of the state-of-the-art lattice-based LHE solutions.

Oblivious Transfer. The 1-out-of-k oblivious transfer (OT) [26] 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 [27], 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 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., $\mathbf{X}_Q = \mathbf{X}\mathbf{W}_Q, \mathbf{X}_K = \mathbf{X}\mathbf{W}_K, \mathbf{X}_V = \mathbf{X}\mathbf{W}_V$. 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 inputs 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 inputs, 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.

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 $\langle\mathbf{Y}\rangle = \Pi_{\mathsf{LayerNorm}}(\langle\widetilde{\mathbf{Y}}_2\rangle)$.

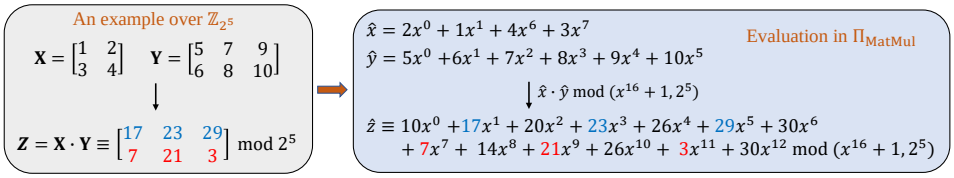

Figure 2: A toy example of our matrix multiplication protocol.

4 Supporting Protocols

4.1 Protocol for Matrix Multiplication

As shown in Section 1, naively extending well-studied matrix-vector multiplication protocols [28, 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 state-of-the-art protocol, Cheetah [14]. Recall that the plaintext of LHE schemes is a polynomial, which can pack a large number of inputs to amortize the overhead [28, 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, 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 $\pi_{\mathrm{L}} : \mathbb{Z}_{2^\ell}^{m \times n} \to \mathbb{A}_{N,2^\ell}$ and $\pi_{\mathrm{R}} : \mathbb{Z}_{2^\ell}^{n \times k} \to \mathbb{A}_{N,2^\ell}$ as follows:

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

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 $\mathbf{Z} = \mathbf{XY} \mod 2^\ell$ 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 $\mathbf{Z} = \mathbf{XY} \mod 2^\ell$ 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 $mnk \leq N$, 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.

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.

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 $\mathsf{Softmax}_i(\mathbf{x}) =$

Algorithm 1 Secure Matrix Multiplication Protocol

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

Output: P_0 and P_1 get $\langle \mathbb{Z}\rangle_0, \langle \mathbb{Z}\rangle_1 \in \mathbb{Z}_{2^\ell}^{m\times k}$, 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 $\hat{y}_{\beta,\gamma} = \pi_L(\mathbf{Y}_{\beta,\gamma})$ 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 $\mathbf{X}_{\beta,\alpha} \in \mathbb{Z}_{2^\ell}^{m_\mathrm{w}\times n_\mathrm{w}}$. P_0 samples $\mathbf{R} \in \mathbb{Z}_{2^\ell}^{m\times k}$ uniformly at random to mask the resulting ciphertexts.

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) $\hat{r}_{\alpha,\gamma} = \pi_R(\mathbf{R}_{\alpha,\gamma})$ for $\alpha \in [m']$ and $\gamma \in [k']$.

6: On receiving the ciphertexts $\{\mathrm{CT}_{\beta,\gamma}\}$ from P_1, P_0 operates $CT_{\alpha,\gamma} = \boxplus_{\beta\in[m']}\left(\mathrm{CT}'_{\alpha,\beta} \boxtimes \hat{x}_{\alpha,\beta}\right)$ for $\beta \in [n']$ and $\gamma \in [k']$. Then P_0 sends to P_1 the ciphertexts $\{\mathrm{CT}_{\alpha,\gamma}\} = \left\{\mathrm{CT}'_{\alpha,\gamma}\right\} \boxminus \hat{r}_{\alpha,\gamma}$.

7: P_0 outputs $\mathbf{R} \bmod 2^\ell$ as the share $\langle \mathbf{Z}\rangle_0$.

8: On receiving the ciphertexts $\{\mathrm{CT}_{\alpha,\gamma}\}$ from P_0, P_1 computes $\langle \hat{z}_{\beta,\gamma}\rangle_1 = \mathsf{Dec}(\mathrm{CT}_{\alpha,\gamma})$ that are decoded to $\langle \mathbf{Z}\rangle_1$ using the method in Theorem 4.1.

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

Figure 3: Underlying protocols from [13, 17]

$e^{x_i}/\sum_{j\in[d]} e^{x_j}$ for $i \in [d]$. The main challenge is to efficiently compute the underlying exponential function. Following the idea from [11, 29], 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 [30]. 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 $\mathsf{Softmax}$ 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 with our solution, it additionally requires d calls to the reciprocal protocol Π_{Recip} and multiplication protocol $\Pi_{\mathrm{MUT_{OT}}}$ in softmax. 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 $\langle \mathbf{x}\rangle_0 \in \mathbb{Z}_{2^\ell}^d$, $\langle \mathbf{x}\rangle_1 \in \mathbb{Z}_{2^\ell}^d$, respectively.

Output: P_0, P_1 get $\langle \mathbf{y}\rangle_0 \in \mathbb{Z}_{2^\ell}^d$, $\langle \mathbf{y}\rangle_1 \in \mathbb{Z}_{2^\ell}^d$, respectively, where $\mathbf{y} = \mathrm{Softmax}(\mathbf{x})$.

1: P_0, P_1 invoke $\Pi_{\max}(\mathbf{x})$ 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 $\Pi_{\mathrm{Mul_{OT}}}$ 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 $\mathsf{GELU}(x) = 0.5x\left(1 + \mathsf{Tanh}\left[\sqrt{2/\pi}\left(x + 0.044715x^3\right)\right]\right)$. The com-

plete 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-only 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 $x \leq 0$, 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 $\langle z \rangle = \text{Fix}(\sqrt{2/\pi})(\langle x \rangle + \text{Fix}(0.044715)\langle x \rangle^3)$
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 $\langle \text{Fix}(0.5)x \rangle$ and $\langle \text{Fix}(1) + \mathsf{Tanh}(z) \rangle$, and output $\langle y \rangle$.

4.2.3 LayerNorm

For a vector $\mathbf{x} \in \mathbb{Z}_{2^\ell}^d$, 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 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 $\langle y \rangle$.

5 Evaluation

5.1 Experimental Setup

Implementation. `Iron` is built on top of the SEAL library [31] and the EMP toolkit [32] in C++. We also use the EzPC framework [33]. This framework compiles high-level TensorFlow code to secure computation protocols, which are then executed by our designed cryptographic backends. Like [17],

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×)

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 [34, 35]: BERT-Tiny, BERT-Medium, BERT-Base and BERT-Large. These models are parameterized by three hyper-parameters: the number of blocks L, the dimension of representations H and the number of input tokens N (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 MNLI, MRPC, SST-2, and QNLI from GLUE benchmarks [18]. We report the accuracy of private inference in Appendix A.4.2, which is comparable to plaintext accuracy due to our numerically precise protocols [17].

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]. 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 shorter 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 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 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 Appendix A.4.2. The results show that our protocols are orders of magnitude better, in terms of both time and communication costs. This is in line with the observation in SIRNN that specialized protocols are more communication efficient than generic alternatives.

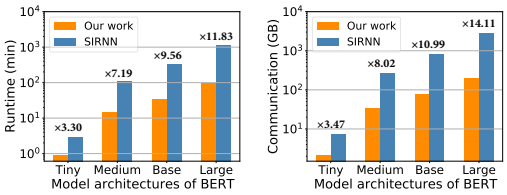

Figure 4: End-to-end inference comparisons

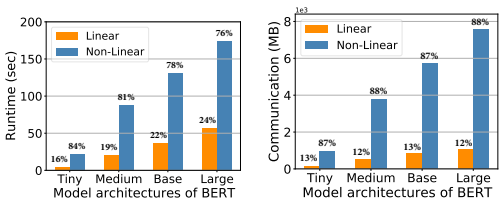

Figure 5: Performance breakdown on BERT.

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 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.

6 Discussion

Pushing expensive operations into an offline phase. Our framework is readily extended to the pre-processing model, including an offline client-input independent phase and an online input-dependent phase. This paradigm has been instantiated in private CNN inference [36, 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 [21, 36] 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 [37, 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, 38], 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 [39, 40]. 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 [41, 42], which have been commonly adopted in private CNN inference [12, 43, 44]. Like prior works [12, 43, 44] 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.

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.

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] 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.

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

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

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

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

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

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

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

[27] 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.

[28] 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.

[29] 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.

[30] 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.

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

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

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

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

[35] 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.

[36] 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.

[37] 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.

[38] 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.

[39] 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.

[40] 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.

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

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

[43] 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.

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

[45] Fabian Boemer, Yixing Lao, Rosario Cammarota, and Casimir Wierzynski. ngraph-he: a graph compiler for deep learning on homomorphically encrypted data. In *Proceedings of the 16th ACM International Conference on Computing Frontiers*, pages 3–13, 2019.

[46] Ziqi Yang, Jiyi Zhang, Ee-Chien Chang, and Zhenkai Liang. Neural network inversion in adversarial setting via background knowledge alignment. In *Proceedings of ACM CCS*, pages 225–240, 2019.

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

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]