BOLT: Privacy-Preserving, Accurate and Efficient Inference for Transformers

Qi Pang[†], Jinhao Zhu[‡], Helen Möllering[◇], Wenting Zheng[†], Thomas Schneider[◇]
[†]*Carnegie Mellon University* [‡]*UC Berkeley* [◇]*Technical University of Darmstadt*
{*qipang, wenting*}*@cmu.edu, jinhao.zhu@berkeley.edu,* {*moellering, schneider*}*@encrypto.cs.tu-darmstadt.de*

***Abstract*—The advent of transformers has brought about significant advancements in traditional machine learning tasks. However, their pervasive deployment has raised concerns about the potential leakage of sensitive information during inference. Existing approaches using secure multiparty computation (MPC) face limitations when applied to transformers due to the extensive model size and resource-intensive matrix-matrix multiplications. In this paper, we present BOLT, a privacy-preserving inference framework for transformer models that supports efficient matrix multiplications and nonlinear computations. Combined with our novel machine learning optimizations, BOLT reduces the communication cost by 10.91×. Our evaluation on diverse datasets demonstrates that BOLT maintains comparable accuracy to floating-point models and achieves 4.8-9.5× faster inference across various network settings compared to the state-of-the-art system.**

***Index Terms*—secure multi-party computation, homomorphic encryption, secure machine learning inference, transformer**

1. Introduction

Transformer models have recently emerged as a game-changing technology. ChatGPT [1] has made the power of transformer-based language models accessible for everyone. Compared to traditional supervised, task-specific learning, large transformer models are trained on huge quantities of unlabeled textual data and are directly useful for a wide variety of applications such as translation, content generation, or question answering. For example, they can be used in the health care domain to identify patterns and risk factors by analyzing electronic health records, medical literature, and clinical notes which can significantly advance diagnosis, treatment, and drug discovery [66], [5], [62].

Transformer is a general neural network architecture that is characterized by the usage of attention mechanisms [71]. Attention mechanisms effectively capture relationships between tokens to model contextual information and capture long-range dependencies in the input token sequence.

However, such powerful models do not come without risks, especially in terms of privacy [46], [69]. ChatGPT is an example of *Machine Learning as a Service* (MLaaS), where a server (OpenAI) hosts a proprietary model, and users input their data into the model and receive a prediction result in return. However, the MLaaS setting raises privacy concerns for both parties: either the user has to upload their private data to the company's servers, or the server needs to store its proprietary model on the user's edge device. ChatGPT operates in the former setting. As a result, serious user data privacy concerns have been raised, which even led to a temporary ban of ChatGPT in Italy [47], [55], [45].

While privacy rights may appear to conflict with effective data analytics, we can employ a cryptographic method known as secure multi-party computation (MPC) to safeguard both data and model privacy without compromising functionality. At a high level, MPC allows n parties $p_1, \ldots, p_n$ with corresponding inputs $x_1, \ldots, x_n$ to learn the output of a public function $f(x_1, \ldots, x_n)$ without revealing each party's x_i to other parties. Each party's data is effectively "encrypted" throughout the entire computation. At the end of the computation, each party only learns the final result without gaining any additional information.

While MPC can be utilized for generic computation, it often leads to substantial computation and communication costs due to the expensive cryptographic primitives. Recent studies have concentrated on developing optimized solutions tailored to specific workloads for the efficient private evaluation and training of convolutional neural networks [49], [50], [25], [41], [60], [4]. However, securing transformers is especially challenging due to multiple reasons:

1) Transformers are considerably larger than convolutional neural networks, containing hundreds of millions to billions of parameters [17], [52], [56], [57].
2) Prior works on private neural networks propose optimized protocols for private matrix-vector multiplications, but transformers necessitate large-scale matrix-matrix multiplications.
3) Securely evaluating hundreds of thousands of inputs for complex non-linear functions in transformers is extremely expensive.

Therefore, new protocols need to be developed for securing transformer inference. To the best of our knowledge, Iron [28] is the state-of-the-art system that investigates how to fully preserve data privacy during standard transformer inference. However, it falls short in offering a practical solution due to its significant performance overhead. For instance, our implementation of Iron requires 280.99 GB communication for an end-to-end inference, taking 216 minutes for an end-to-end inference on the BERT-base model (110 M parameters [17]) under one of the WAN settings

(100 Mbps, 80 ms). As systems and hardware continue to evolve [63], [64], [36], [37], [68], we believe it is essential to reduce network communication since computation can be accelerated and parallelized. The linear and non-linear protocols along with machine learning optimizations in BOLT significantly reduce communication while maintaining efficient computations, marking substantial progress towards practical privacy-preserving transformer inference.

Our contributions. We propose BOLT, a novel privacy-preserving inference system for transformer models, which addresses the aforementioned challenges by reducing the communication costs, and thus improves the end-to-end runtime. BOLT guarantees the confidentiality of the client's input data and also protects the service provider's intellectual property – its model. BOLT integrates cryptographic improvements, accurate and efficient approximations for non-linear functions, and algorithmic enhancements from the perspective of machine learning.

- **Communication-optimized and computation-efficient linear operations.** The first challenge we face in BOLT is that transformers have much more complicated linear operations that have high multiplicative depth. Many past secure inference protocols utilize homomorphic encryption (HE) only for plaintext-ciphertext matrix-vector multiplications, which are present in convolutional neural networks. Iron tackles this by using a mix of HE and MPC, but the techniques result in high communication costs. Instead, BOLT's cryptographic improvements only make use of HE to efficiently save on communication costs. We develop compute-efficient, low-depth algorithms in HE. Our first insight is an alternative interpretation of ciphertext-plaintext matrix-matrix multiplication that results in an optimized packing of ciphertexts. Prior works [35], [28] waste a part of the communication as they leave some ciphertext slots "empty". However, merely changing packing is not enough since it still requires many computationally expensive rotations for the matrix multiplications in transformer models. We propose a second optimization to address this issue: we adapt the *baby-step giant-step* strategy [27], [7] to our matrix multiplication algorithm in order to reduce the number of input ciphertext rotations, and instead and run the rotations later on partial sums of the intermediate result. This reduces the number of ciphertext rotations between 2.33× to 9.33× for the BERT-base model. Finally, we design ciphertext-ciphertext matrix multiplication algorithms that are efficient and have low multiplicative depth in HE.
- **Accurate and efficient non-linear operations.** An accurate and efficient protocol for non-linear operations is also crucial. Iron shows that more than 75% of their total runtime is attributed to non-linear layers, due to the complex computations involved when transferring plaintext non-linear formulas directly into MPC protocols. These include trigonometric and exponential functions which are on their own already very expensive in MPC [59]. To address this, we introduce two highly precise polynomial approximations for GELU and Tanh of order 4 and 5 respectively, along with an optimized Softmax procedure. Given that even four or five multiplications can be expensive considering input dimensions (e.g., 128×3072 for each GELU function in 12 layers of BERT-base), we present an additional optimization based on [51] that is also of independent interest for the secure computation community: a *polynomial pre-processing* technique that allows reducing the number of multiplications for the evaluation of polynomials of order n (Horner's scheme) to approximately $\lceil \frac{n}{2} \rceil$ when the polynomial's coefficients are known in advance.
- **Machine learning optimizations.** We further propose machine learning optimizations to enhance efficiency and accuracy. Specifically, we develop oblivious word elimination using attention scores. These scores, which measure the correlations between input tokens, serve as a metric to rank token importance. We then apply oblivious bitonic sort to discard less contributive tokens and significantly reduce input size. Additionally, we utilize secure computation-aware fine-tuning to bridge the gap between secure and floating-point computations to further improve accuracy.

Combining all our optimizations, BOLT consumes 10.91× less communication and has 4.8× to 9.5× better total run-time than Iron in different network settings. We consider this a big step towards practical privacy-preserving transformer inference. We have open-sourced our code at https://github.com/Clive2312/BOLT.

2. Background

2.1. Notation

We denote by $[n]$ the set $\{0, 1, \ldots, n-1\}$, where $n \in \mathsf{N}$. We use lower case bold letter such as x to represent vectors and x_i to indicate the i-th element of the vector x. Correspondingly, $\mathsf{X} \in \mathsf{R}^{a \times b}$ is a matrix of $a \times b$ real values and $\mathsf{X}_{i,j}$ is the element indexed by row i and column j of the matrix X. $\langle \mathbf{u}, \mathbf{v} \rangle$ is the inner product and $\mathbf{u} \otimes \mathbf{v}$ is the outer product. $\lceil x \rceil$ is the ceiling operation on x, $\lfloor x \rfloor$ is the floor operation, and $\lceil x \rfloor$ is rounding. A multiplexer gate is denoted by $a?b : c$ and indicates if a is true then return b and otherwise return c. $x >> i$ and $x << i$ indicate right and left shifts of a value x by $i \in \mathsf{N}$ positions in binary. λ is the computational security parameter. A box, e.g., $\boxed{\mathbf{A}}$, indicates private values, i.e., either encrypted or secret shared.

2.2. Transformers

In this paper, we focus on the BERT (Bidirectional Encoder Representations from Transformers [17]), a widely used transformer model. We present the BERT architecture in Fig. 1.

Input Format. BERT receives user input as text, which is tokenized using vocabulary and position information before being fed into the transformer model. Each token in the input sentence maps to a high-dimensional vector. Unlike

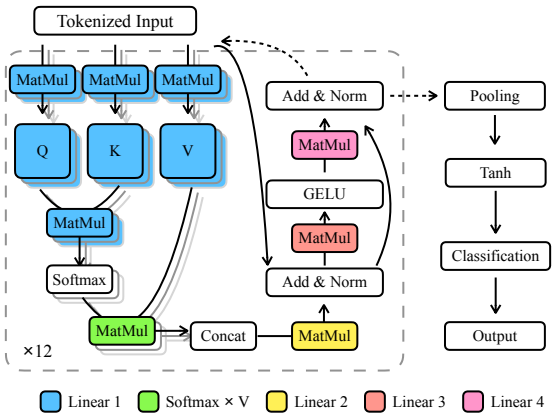

Figure 1: BERT [17]'s high-level architecture.

traditional neural networks like CNNs, the transformer's input is a 2D matrix rather than a 1D vector. Specifically, for an input $\mathbf{X} \in \mathsf{R}^{m \times d}$, m represents the length of tokens and d denotes the model dimension.

Encoder. An encoder component starts with a *multi-head attention* block followed by two normalization operations with feed-forward layers in between. Attention layers [71] capture context and dependencies among words in a sentence. Concretely, an input $\mathsf{X} \in \mathsf{R}^{m \times d}$ is fed into L "headers". Each header linearly projects the input using query, key, and value parameter matrices $\mathsf{W}_h^Q, \mathsf{W}_h^K, \mathsf{W}_h^V \in \mathsf{R}^{d \times d'}$ as well as the respective biases $\mathsf{B}_h^Q, \mathsf{B}_h^K, \mathsf{B}_h^V \in \mathsf{R}^{d'}$, where $h \in [H]$, $d' = d/H$, and H is the number of heads in each multi-head attention layer. The result of the projections are Q_h, K_h and $\mathsf{V}_h \in \mathsf{R}^{m \times d'}$. In BERT-base, $m = 128, d = 768, H = 12$ and $L = 12$. Next, the attention score $\mathsf{Att}_h \in \mathsf{R}^{m \times d'}$ is computed before being fed into a feed-forward network:

$$\mathsf{Att}_h = \mathsf{Softmax}(\frac{\mathsf{Q}_h \times \mathsf{K}_h^T}{\sqrt{d}}) \times \mathsf{V}_h, \tag{1}$$

where the $\mathsf{Softmax}$ function is defined as:

$$\mathsf{Softmax}(\mathsf{X})_{i,j} = \frac{e^{\mathbf{x}_{i,j}}}{\sum_{j \in [d]} e^{\mathbf{x}_{i,j}}}, i \in [m], j \in [d] \tag{2}$$

Then the attention results are concatenated as $\mathsf{Att} = \mathsf{Concat}(\{\mathsf{Att}_h\}_{h \in [H]})$, where $\mathsf{Att} \in \mathsf{R}^{m \times d}$. Further, the attention value is normalized across the layer:

$$\mathsf{LayerNorm}(\mathsf{X})_{i,j} = \frac{\quad_j(\mathsf{X}_{i,j} - \quad_i)}{\quad_i} + \quad_j, \tag{3}$$

where $i \in [m]$, $j \in [d]$, $\quad \in \mathsf{R}^m$ is the mean, and $\quad \in \mathsf{R}^m$ is the variance. $\quad \in \mathsf{R}^d$ and $\quad \in \mathsf{R}^d$ are model parameters.

Transformers employ the GELU activation function between the feed-forward layers due to its favorable curvature and non-monotonicity properties [30], [31]:

$$\mathsf{GELU}(x) = \frac{1}{2}x \cdot (1 + \mathsf{erf}(\frac{x}{\sqrt{2}})), \tag{4}$$

where the Gauss error function is $\mathsf{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$.

Pooler and classifier. For downstream tasks, the output of the L encoder layers is fed into a pooler and classifier layer, and Tanh is usually used as the activation function. The first token of the encoding result (corresponding to the first row of the result matrix) is a special token that is being utilized as the input of pooler and classifier, which output the result.

Fine-tuning. BERT is an encoder model trained on a large public dataset. To enhance its performance on downstream tasks, it requires fine-tuning on smaller datasets, which may be private or contain sensitive information. Fine-tuning is more cost-effective than training and typically results in good accuracy after several epochs.

2.3. Cryptographic primitives

Secure multi-party computation (MPC). MPC techniques allow a set of P mutually distrusting parties $\mathcal{P}_1, \cdots, \mathcal{P}_P$ to collaboratively evaluate an arbitrary function f on their respective private inputs $x_1, \ldots, x_P$ without revealing any information about the inputs except from what can be inferred from the output [74].

The MPC framework [11] that we use to instantiate our protocols in §7 implements 2-out-of-2 additive secret sharing with two parties $\mathcal{P}_1$ and $\mathcal{P}_2$. Here, each $\mathcal{P}_i$ (where $i \in \{1, 2\}$) holds a share $\langle x \rangle_i$ of a secret $x \in \mathsf{Z}_{2^\ell}$ such that $x = \langle x \rangle_1 + \langle x \rangle_2 \mod 2^\ell$, where ℓ is the bit length. While additions can be executed locally "for free", multiplications require communications between the parties and are realized using oblivious transfer (OT).

Homomorphic encryption. Homomorphic encryption (HE) is another important cryptographic primitive for secure computation that allows a party to execute computations on another party's encrypted data without decryption. Recently, there has been a lot of exciting developments in lattice-based HE schemes [8], [21], [24], [9], [13] because of their relative efficiency compared to previous designs. Lattice-based HE schemes were originally designed for *leveled homomorphic encryption*, which can support computation that have limited multiplicative depth. The reason for those limitations is that the cryptographic hardness assumption leads to noise in a ciphertext, and the noise increases with each operation, such that, at some point decryption fails. Thus, controlling the noise growth is crucial for ensuring the correctness of the computation. In this paper, we leverage leveled HE (specifically, BFV [9]) instead of fully homomorphic encryption [23] (FHE), since FHE requires bootstrapping, which is still prohibitively expensive. Key parameters of HE schemes typically consist of dimension of the polynomial ring n, plaintext modulus p, and ciphertext modulus q.

Trade-off between MPC and HE. While MPC and HE both enable secure computation, there is no clear "better" technique. Instead, the choice has to be made based on multiple factors depending on the requirements of the application and setup. HE typically has lower communication cost and higher computational overhead than MPC, especially for complex non-linear computations. In contrast, MPC can be based mostly on efficient symmetric cryptographic

primitives to evaluate the non-linear functions, but requires expensive communication between computing parties.

3. System Overview

System setup. BOLT is a secure inference protocol for transformer models, specifically BERT. The service provider owns and maintains a fine-tuned transformer model, privately offering it to users whose input data is sensitive. This corresponds to a secure two-party computation (2PC, §2.3) scenario. BOLT allows the client and server to engage in a tailored 2PC protocol for privacy-preserving inference.

Threat model. We assume that the two mutually distrusting parties, server $\mathcal{P}_1$ and client $\mathcal{P}_2$, are semi-honest (a.k.a. honest-but-curious). They will correctly follow the protocol specifications while attempting to passively gain additional knowledge that cannot be inferred from the output. Semi-honest security is a common assumption for privacy-preserving machine learning (PPML). It is used in related works on private neural network inference [35], [49], [32], [44], [28]. There are two main reasons for such an assumption: 1) the parties have to trust each other to some extent since the server is providing the inference service; 2) semi-honest performance is much more practical.

BOLT assumes that the model architecture is known to both parties. As is standard in all secure computation protocols, we also do not aim to hide leakage of the inference result. Note that this is inherent to the ideal functionality of those protocols because the result needs to be useful and be revealed to someone, often both of the parties that are involved in the original computation. There are orthogonal techniques like differential privacy [20] that can be combined with BOLT to alleviate such information leakage.

In a typical BERT use case, the service provider will first fine-tune the model on their private datasets. Considering that the original BERT model is trained on a large public dataset, its well-trained tokenization can effectively map sentences to a BERT-friendly space, we freeze the tokenization during private fine-tuning. That is, the sentence tokenization process is public and contains no sensitive information. Furthermore, both parties know the parameter scales and dimensions of each layer.

System architecture. BOLT is a hybrid protocol that uses both HE and MPC. In §4, we discuss our novel efficient protocols for matrix multiplication using HE. §5 discusses our techniques for approximating and optimizing non-linear functions. Since the non-linear functions are very expensive to express in HE, we use secret sharing-based MPC to implement these functionalities. Thus, we adopt a hybrid approach, and support conversions between HE and MPC. This also allows us to automatically reset the HE noise without using expensive bootstrapping operations.

BOLT primarily focuses on optimizing the overall execution cost instead of just the online cost. We consider the overall protocol execution crucial, as merely optimizing for online cost can add additional overall cost and have extra memory/storage overheads [22].

Native computation in MPC and HE operates over rings, and therefore floating point arithmetic is very expensive [41], [58], [16]. However, the parameters of transformer models, as well as all intermediate values in a transformer inference, are typically represented in 32-bit floating point numbers. To achieve both efficiency and accuracy, we encode all values in fixed-point arithmetic. Based on the bit length ℓ and the length of the fractional part $s \in \mathsf{Z}$ (also called the *scale*), a real value $x \in \mathsf{R}$ is converted into its (approximated) fixed-point representation by computing $y = \lfloor x \cdot 2^s \rfloor \mod 2^\ell$. Hence, decoding is defined as $y/2^s$. All the operations of MPC are performed over the ring Z_{2^ℓ}, and all the operations of HE are over the ring Z_p. We utilize the conversions between MPC and HE; see §7.1.2.

While the techniques of BOLT are applicable to all transformers, we use BERT [17] as a running example in the following for the sake of simplicity of presentation.

4. Efficient MatMul in BOLT

A key characteristic of the transformer architecture in BERT is the attention mechanism §2.2, which consists of large matrix multiplications. Similar to prior works [35], [49], [32], [28], we use HE to securely compute them as HE is relatively efficient for linear operations and allows us to save on communication bandwidth and roundtrips compared to MPC. Given the trends in systems and hardware, we believe that it is more important to save on network communication as computations can be accelerated and parallelized. For example, there are several recent exciting works on hardware acceleration of FHE [63], [64], [36], [37], [68] that achieve significant run-time savings.

The challenges that we face in designing our techniques are two-fold. First, BERT has matrix-matrix multiplications instead of matrix-vector multiplications. Second, the linear computation also has increased multiplicative depth leading to tricky ciphertext-ciphertext matrix multiplication. We note that our approach of using HE for computing all building blocks is rather unusual – prior works [35], [49], [32], [34] only have plaintext-ciphertext matrix-vector multiplication or only supports CNN models, and Iron [28] also uses MPC for ciphertext-ciphertext matrix multiplications.

In this section, we propose an efficient protocol for linear operations. Unlike previous protocols in Iron which waste a lot of ciphertext slots, our protocol has compact ciphertext packing and significantly reduces communication (§ 4.1.1). We optimize the computation by minimizing the multiplicative depth and number of rotations (§ 4.1.2). Additionally, we support ciphertext-ciphertext matrix multiplication, which is integral in the attention mechanism of transformers (§ 4.1.3).

4.1. Building blocks

4.1.1. Ciphertext-Plaintext MatMul with Compact Packing. We start with matrix-matrix multiplications that multiply a ciphertext matrix with a plaintext matrix. We will

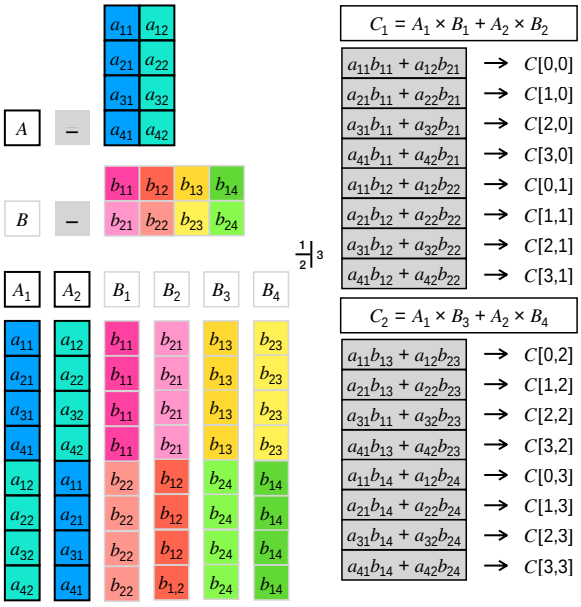

Figure 2: BOLT's compact packing for ciphertext-plaintext matrix-matrix multiplication.

use the example $\boxed{\mathbf{A}} \times \mathbf{B} = \boxed{\mathbf{C}}$, where $\boxed{\mathbf{A}} \in \mathsf{Z}_p^{m \times d_1}, \mathbf{B} \in \mathsf{Z}_p^{d_1 \times d_2}$, in the remaining parts of this subsection for illustration. $\boxed{\mathbf{A}}$ and $\boxed{\mathbf{C}}$ are matrices in ciphertexts.

Current state-of-the-art PPML works that use HE for matrix multiplication are Gazelle [35] and Iron [28]. Both works compute matrix multiplication via inner dot products. When it comes to matrix multiplications, both Gazelle and Iron has sparse packing in the resulting ciphertext. The wasted slots in the ciphertext will induce additional communication overhead and also introduce more multiplications in HE for Gazelle. We present the number of required HE operations in Tab. 1. Interested readers can refer to App. A for more detailed explanation of their methods.

Fig. 2 depicts our improved packing technique, and we make it possible to fully utilize all ciphertext slots. It relies on an alternative interpretation of matrix multiplication that does not explicitly depend on inner dot products. Let's assume we have matrices $\boxed{\mathbf{A}}$, $\mathbf{B}$, and $\boxed{\mathbf{C}}$ as shown in Fig. 2.

If we denote the matrix $\boxed{\mathbf{A}}$ as $@\boxed{\mathbf{A}_1}\ \boxed{\mathbf{A}_2}^{\mathsf{A}}$, where $\boxed{\mathbf{A}_1}, \boxed{\mathbf{A}_2}$ are the columns, then we can clearly see that each column of $\boxed{\mathbf{C}}$ is a *linear combination* of the columns of $\boxed{\mathbf{A}}$, and the scaling factors are scalars from $\mathbf{B}$.

To compute such linear combinations, we pack the ciphertext matrix $\boxed{\mathbf{A}}$ in a column-wise fashion as shown in Fig. 2. However, if we simply multiply each column of $\boxed{\mathbf{A}}$ by the column values of $\mathbf{B}$, we will need to sum the partial products from different columns together. This will result either in underutilized ciphertexts, or require additional rotations, masking, and additions to group the right ciphertexts together. Rotations are expensive in HE [6], and masking will increase multiplicative depth. Instead, we adapt Gazelle's diagonal packing of weight matrix $\mathbf{B}$ and repeat each value b_{ij} #rows of $\boxed{\mathbf{A}}$ times. The number of rows of the left-hand matrix $\boxed{\mathbf{A}}$, namely m, allows us to compactly pack the columns in the input ciphertexts. (As the input token length m for BERT is usually 64, 128, or 256, and the polynomial dimensions n for HE is a power of 2 such as 2048, 4096, 8192, etc.) Note that the output ciphertext automatically encodes matrix $\boxed{\mathbf{C}}$ in the form of columns with the same m as $\boxed{\mathbf{A}}$, i.e., we get a perfectly compact output packing. In Tab. 1, we present the detailed comparison with Gazelle and Iron regarding the number of each operation and the number of ciphertexts both asymptotically and concretely based on the dimensions of BERT-base Linear 1 in Fig. 1. We show that BOLT uses much fewer multiplications compared to Gazelle as the packing is compact and all the slots in the ciphertexts are utilized. Also, Gazelle and Iron have higher communication overhead as they need to pack more ciphertexts. Even though Iron does not need rotations, the communication overhead becomes significant when the matrix size is large. As we will show in §7, BOLT consistently outperforms Iron under different network settings up to a factor of 45.7×. In the following, we will show how to reduce the number of rotations using a baby-step giant-step (BSGS) strategy.

	Gazelle*	Iron	BOLT w/o BSGS	BOLT w/ BSGS
#Mult.	$O(md_1)$ 98304	$O(md_1d_2/n)$ 768	$O(d_1)$ 768	$O(d_1)$ 768
#Rot.	$O(md_1)$ 96768	0 0	$O(d_1)$ 756	$O(\sqrt{m^2d_1^2d_2/n^2})$ 43
#Ct.	$O(md_1/d_2)$ 1664	$O(\sqrt{md_1d_2/n})$ 56	$O(m(d_1+d_2)/n)$ 13	$O(m(d_1+d_2)/n)$ 13

*Assume $n > d_1 > d_2$ for Gazelle.

TABLE 1: Comparison with Gazelle and Iron. The concrete numbers are based on the BERT-base Linear 1 dimensions with $m = 128, d_1 = 768, d_2 = 64, n = 8192$.

4.1.2. Reducing Rotations: A Baby-step Giant-step Strategy. As multiple columns fit into one ciphertext and we would like to fully utilize all ciphertext slots, multiple partial results will end up in the same ciphertext. Thus, rotations are not avoidable to sum those up. In this section, we introduce an optimized protocol that reduces the required number of rotations in ciphertext-plaintext matrix multiplications.

We adapt the baby-step giant-step (BSGS) strategy [27], [7] to our matrix multiplication problem to significantly reduce the number of rotations. The BSGS strategy was originally used for matrix-vector multiplication in HE, but we extend this idea to our matrix-matrix multiplication setting. We pack the ciphertext matrix in column as described in §4.1.1, and we rotate at the column level instead of the element level. Thus, the BSGS cannot be directly applied in the same way as matrix-vector scenarios. Also, in matrix-matrix multiplications, we can also do additional partial

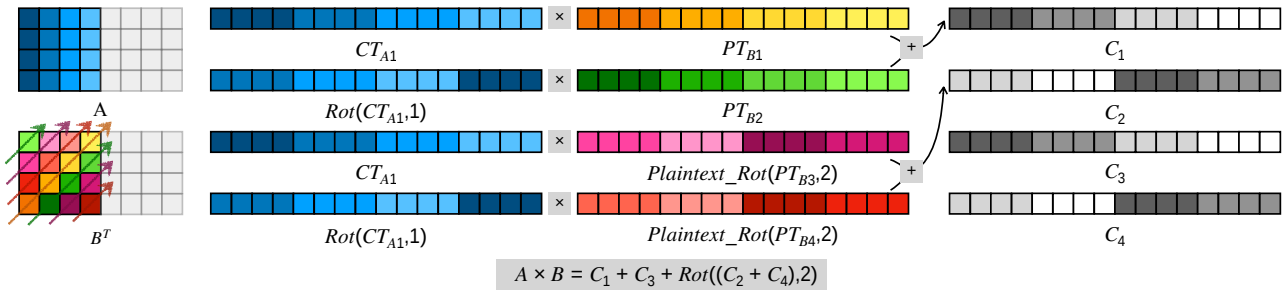

Figure 3: Baby-step giant-step (BSGS) strategy reduces the number of rotations on ciphertexts from $O(d_1)$ to $O(\sqrt{\frac{m^2 d_1^2 d_2}{n^2}})$. This figure shows an example of computing $\boxed{\mathbf{C}} = \boxed{\mathbf{A}} \times \mathbf{B}$, where $\boxed{\mathbf{A}} \in \mathsf{Z}_p^{m \times d_1}$ is encrypted column-wise, and $\mathbf{B} \in \mathsf{Z}_p^{d_1 \times d_2}$ is in plaintext. In this example, $m = 4, d_1 = 8, n = 16$, and $d_2 = 4$. CT_{A1} is a packed ciphertext of the first 4 columns, while $Rot(CT_{A1}, 1)$ is the same ciphertext rotated to the left by 1 column. $PT_{B1}, PT_{B2}, PT_{B3}, PT_{B4}$, are 4 diagonals of matrix $\mathbf{B}$. We repeat the same multiplication as shown in the figure on the right half of the matrix $\boxed{\mathbf{A}}$, whose corresponding packing is CT_{A2} (not shown), and the lower half of the matrix $\mathbf{B}$. We don't explicitly show this step, but assume the results are $\boxed{\mathbf{C}_3}$ and $\boxed{\mathbf{C}_4}$. ($\boxed{\mathbf{C}_1}$, $\boxed{\mathbf{C}_3}$) and ($\boxed{\mathbf{C}_2}$, $\boxed{\mathbf{C}_4}$) can be partially summed together without rotations. From the figure, we can see that BSGS allows us to only rotate the input CT_{A1} and CT_{A2} once, followed by ciphertext SIMD multiplications and partial summations, and finally the partially summed results are rotated.

summations compared to matrix-vector multiplications, especially when the RHS plaintext matrix is tall and skinny.

We visualize the high-level idea of the BSGS strategy in Fig. 3. As described, the column of the private matrix $\boxed{\mathbf{A}}$ is only rotated twice, i.e., each of its ciphertext packings CT_{A1} and CT_{A2} is rotated once. While the other rotations are done on the diagonals of the plaintext matrix $\mathbf{B}$. After summing up the partial results, only one "backwards" rotation is needed to sum up $\boxed{\mathbf{C}}$. The total number of rotations on ciphertexts is therefore 3 instead of 6 in our example.

Consider $\boxed{\mathbf{A}} \in \mathsf{Z}_p^{m \times d_1}$ and $\mathbf{B} \in \mathsf{Z}_p^{d_1 \times d_2}$, and $\boxed{\mathbf{C}} = \boxed{\mathbf{A}} \times \mathbf{B} \in \mathsf{Z}_p^{m \times d_2}$. Using BSGS, asymptotically, we need $O(\sqrt{\frac{m^2 d_1^2 d_2}{n^2}})$ ciphertext rotations. Considering that using our protocol in §4.1.1, we require $O(d_1)$ rotations. Thus, compared to BSGS in the matrix-vector setting, there are additional opportunities for partial summations across the rows if the RHS plaintext matrix is tall and skinny (i.e., d_1 is relatively large and d_2 is relatively small).

We present the detailed comparison with Gazelle and Iron regarding the number of HE operations in Tab. 1, our technique saves about $17.58\times$ rotations compared to BOLT without BSGS on BERT-base model's single-head attention dimensions. We present more concrete savings for other dimensions of BERT-base in App. F.

4.1.3. Ciphertext-Ciphertext MatMul. So far, we have the ciphertext-plaintext matrix multiplication, and in this section, we will discuss the ciphertext-ciphertext matrix multiplication.

To save communication costs, we design an efficient protocol that uses only HE to compute the ciphertext-plaintext as well as the ciphertext-ciphertext matrix-matrix multiplications. The challenge in designing such an algorithm is keeping the multiplicative depth small since higher multiplicative depth will lead to higher noise consumption and larger HE parameters (§2.3). Larger HE parameters, in turn, result in worse performance.

In BERT, there are two different packings that require ciphertext-ciphertext matrix multiplications: 1) the LHS is column-packed and the RHS is row-packed, and 2) the LHS is diagonal-packed and the RHS is column-packed.

Column-packed matrix $\times$ row-packed matrix. We introduce our solution for the first packing scenario. Consider $\boxed{\mathbf{C}} = \boxed{\mathbf{A}} \times \boxed{\mathbf{B}}$, where $\boxed{\mathbf{A}} \in \mathsf{Z}_p^{m \times d_1}$, $\boxed{\mathbf{B}} \in \mathsf{Z}_p^{d_1 \times d_2}$, and $\boxed{\mathbf{C}} \in \mathsf{Z}_p^{m \times d_2}$. $\boxed{\mathbf{A}}$ is column-packed and $\boxed{\mathbf{B}}$ is row-packed. Since both matrices are encrypted, directly using our matrix multiplication technique from §4.1.1 will require replicating the RHS matrix via repacking into many ciphertexts. For the BERT-base dimensions with input token length $m = 128$, repacking the RHS matrix results in $128\times$ more ciphertexts.

Instead, we make the observation that we can use the columns and rows directly and use the *outer product* interpretation of matrix multiplication. Let us again illustrate this view with matrices $\boxed{\mathbf{A}}$ and $\boxed{\mathbf{B}}$. Given the columns $\boxed{A_i}$ and rows $\boxed{B_j}$, $i, j \in [d_1]$, we have that $\boxed{\mathbf{A}} \times \boxed{\mathbf{B}} = \sum_{i \in [d_1]} \boxed{A_i} \otimes \boxed{B_i}$. Therefore, we can similarly use this insight to complete the ciphertext-ciphertext matrix multiplication without additionally expanding and repacking the matrix $\boxed{\mathbf{B}}$ into the diagonal form.

Directly multiplying the left ciphertext and the right ciphertext will result in the *diagonals* of the output matrix. To compute the other diagonals of the matrix, we need to rotate within the rows of $\boxed{\mathbf{B}}$. This requires 2D rotations for the ciphertext — where the dimensions are 128×64 for the BERT-base attention layers — but such a 2D rotation is not supported in BFV given the powers that are used for the roots of unity in the NTT space (in fact, it is unclear

whether it is possible to find the appropriate generators with orders 64 and 128 for $n = 8192$, that also span the entire space). Instead, we rely on partial rotations that need to be corrected by bit masking.

Let's take a look at an example to ease understanding. We assume that full rotations are supported in this example even though BFV supports 2D rotations of size $n/2 \times 2$. A 2×4 matrix encoded in a vector $\langle x_{1,1}, x_{1,2}, x_{1,3}, x_{1,4}, x_{2,1}, x_{2,2}, x_{2,3}, x_{2,4} \rangle$ can be rotated within the first half and the second half to the left by 1 by doing two separate rotations, applying an extra mask, and a summation. The first rotated vector is the ciphertext shifted to the left by 1. The second rotated vector is the ciphertext rotated to the right by 3. The mask vectors are $\langle 1, 1, 1, 0, 1, 1, 1, 0 \rangle$ and $\langle 0, 0, 0, 1, 0, 0, 0, 1 \rangle$. By multiplying the rotated vectors with the masks, and then summing the results, we can get the correct 2D rotation $\tilde{\mathbf{x}} = \langle x_{1,2}, x_{1,3}, x_{1,4}, x_{1,1}, x_{2,2}, x_{2,3}, x_{2,4}, x_{2,1} \rangle$.

The above approach requires an additional logarithmic number of rotations to sum up the partial products within each resulting ciphertext. This results in a non-compact output packing, which can be naively solved by multiplying with another bit mask. However, doing so results in a multiplicative depth of 4 (given $\boxed{\mathbf{A}}$ and $\boxed{\mathbf{B}}$ already have depth 1), which will require larger parameters in BFV and lead to worse performance. Instead, we observe that it is possible to reduce the depth to 3 (note that depth 2 is optimal) by combining the first masking (for correcting rotations) and the second masking (for producing a compact packing). Instead of performing the first masking to get the correct rotations, we can directly execute the multiplications on the partially rotated ciphertexts, which results in partially correct results as well. Following this, we can sum up the partial results before applying the compact packing masking.

To illustrate this in an example, we use the same vectors as before to compute $\mathsf{x} \cdot \mathsf{y}$, where x is rotated by one within the rows. Instead of using x, we can multiply the two partially rotated x with y to get $\langle x_{1,2} \cdot y_{1,1}, x_{1,3} \cdot y_{1,2}, x_{1,4} \cdot y_{1,3}, x_{2,1} \cdot y_{1,4}, x_{2,2} \cdot y_{2,1}, x_{2,3} \cdot y_{2,2}, x_{2,4} \cdot y_{2,3}, x_{1,1} \cdot y_{2,4} \rangle$. Similarly, we get $\langle x_{2,2} \cdot y_{1,1}, x_{2,3} \cdot y_{1,2}, x_{2,4} \cdot y_{1,3}, x_{1,1} \cdot y_{1,4}, x_{1,2} \cdot y_{2,1}, x_{1,3} \cdot y_{2,2}, x_{1,4} \cdot y_{2,3}, x_{2,1} \cdot y_{2,4} \rangle$. Rotating these ciphertexts by half and summing, plus one final masking, returns us the same final result, but with multiplicative depth of only 3. This increases the ciphertext-ciphertext multiplications and rotation numbers, but the reduction in depth allows us to use smaller HE parameters, which increases overall performance.

Diagonal-packed matrix $\times$ column-packed matrix. Now we discuss the second packing scenario, where the LHS is packed in diagonal and the RHS is packed in column. Consider $\boxed{\mathbf{C}} = \boxed{\mathbf{A}} \times \boxed{\mathbf{B}}$, where $\boxed{\mathbf{A}} \in \mathsf{Z}_p^{m \times m}$, $\boxed{\mathbf{B}} \in \mathsf{Z}_p^{m \times d_2}$, and $\boxed{\mathbf{C}} \in \mathsf{Z}_p^{m \times d_2}$. Note that in this scenario, the LHS matrix will always be a squared matrix.

One way to compute the matrix multiplication in this packing is to use Gazelle's packing to repeatedly pack the LHS in diagonal and pack each column of the RHS matrix in a separate ciphertext to enable rotations within columns.

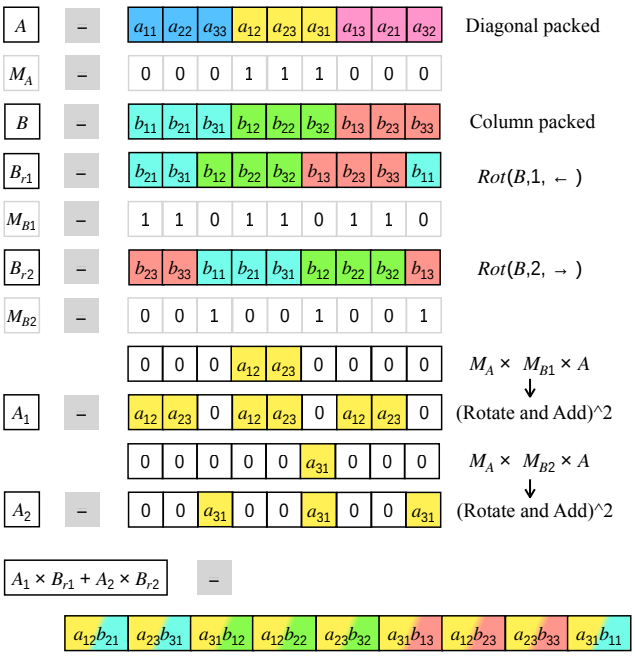

Figure 4: Diagonal-packed matrix $\times$ column-packed matrix.

However, this requires repacking both ciphertexts, resulting in significantly large communication overhead (as shown in Tab. 1, Gazelle's packing is very sparse in matrix-matrix multiplications).

Instead of repacking by communication, we apply masking, rotation, and addition to get the desired repeated packing of the LHS. To enable the rotation within the columns of the RHS, we apply the same strategy as the first packing scenario. Similarly, we combine the mask of the RHS partial rotations with the mask of the LHS packing preparation. In this way, we can complete the matrix-matrix multiplication with 3 multiplicative depth.

We present a concrete example in Fig. 4. When we compute the multiplication between the second diagonal of $\boxed{\mathbf{A}}$ and the 1-step rotation of $\boxed{\mathbf{B}}$, we first construct the plaintext mask m_A, which takes the second diagonal in $\boxed{\mathbf{A}}$, and the rotation masks m_{B1} and m_{B2} for $\boxed{\mathbf{B}}$, which takes the values of $\boxed{\mathbf{B}}$'s partial rotations in correct positions. We first multiply the masks together and then they are multiplied with $\boxed{\mathbf{A}}$. After that, we pack $\boxed{\mathbf{A}}$ via rotation and addition. The packed $\boxed{\mathbf{A}}$, which are $\boxed{\mathbf{A}_1}$ and $\boxed{\mathbf{A}_2}$ are multiplied to the partial rotations of $\boxed{\mathbf{B}}$. Summing these two multiplication results, we will get the multiplication result between the second diagonal of $\boxed{\mathbf{A}}$ and corresponding values in matrix $\boxed{\mathbf{B}}$. Repeat such procedure for each diagonal, and sum the results together, we will get the matrix multiplication result in a compact packing.

4.2. Efficient Linear Computation in BERT

Attention layer (query, key and value). This corresponds to the first three MatMul in Linear 1 of Fig. 1. As we mentioned in §2.2, the input is a 2D matrix $\mathbf{X} \in \mathsf{Z}_p^{m \times d}$. The client will compactly pack the input in column and send the encrypted ciphertext to the server. The server will pack the plaintext model weights $\mathbf{W}_h^Q, \mathbf{W}_h^K, \mathbf{W}_h^V$ in diagonal as we have illustrated in §4.1.1. Then the server uses our protocol in §4.1.1 to compute the multiplication results $\boxed{\mathbf{Q}_h}$, $\boxed{\mathbf{K}_h}$, and $\boxed{\mathbf{V}_h}$. These results are ciphertexts and are also compactly packed in column. We apply BSGS §4.1.2 to reduce rotations, the concrete savings on rotations are deferred to App. F.

Attention layer (query $\times$ key). This corresponds to the last MatMul in Linear 1 of Fig. 1, which computes the $\boxed{\mathbf{Q}_h} \times \boxed{\mathbf{K}_h^T}$. Since both $\boxed{\mathbf{Q}_h}$ and $\boxed{\mathbf{K}_h}$ are packed in column, then $\boxed{\mathbf{K}_h^T}$ is packed in row. Thus, we can use our protocol in §4.1.3 under the column-packed matrix $\times$ row-packed matrix case to compute $\boxed{\mathbf{Q}_h} \times \boxed{\mathbf{K}_h^T}$. The packing of the result ciphertext is also compact. The server will add a uniformly random mask to the multiplication result and send the masked result to the client for decryption. Then, after conversion to MPC (see §7.1.2), the server and the client can jointly evaluate $\mathsf{Softmax}(\mathbf{Q}_h \times \mathbf{K}_h^T)$ in MPC.

Attention layer ($\mathsf{Softmax} \times \mathbf{V}$). After the server and the client get the result of $\mathsf{Softmax}$ in secret sharing, $\boxed{\mathbf{S}_h} = \mathsf{Softmax}(\mathbf{Q}_h \times \mathbf{K}_h^T)$, they will compute $\boxed{\mathbf{S}_h} \times \boxed{\mathbf{V}_h}$. Note now the RHS of the matrix multiplication is a column-packed matrix. We thus pack the LHS in diagonal and use our protocol in §4.1.3 under the scenario of diagonal-packed matrix $\times$ column-packed matrix. Specifically, the client will compactly pack their sharing of $\boxed{\mathbf{S}_h}$ in diagonal, convert to HE §7.1.2, encrypt, and send the diagonal-packed ciphertext to the server. The server will first convert the sharing to HE, add it to the received ciphertext, and finish the remaining computations according to our protocol in §4.1.3.

Feed-forward layers (Linear 2, 3, 4 of Fig. 1). The feed-forward layers in BERT only consist of ciphertext-plaintext matrix multiplications. Thus, we utilize the ciphertext-plaintext protocol in §4.1.1 to compute the feed-forward layers. Similarly, the server and client will first convert their sharings to HE and then the client will compactly pack their sharing in column, encrypt, and send to the server. The server will finish the remaining computations. Note that we apply BSGS §4.1.2 in all the ciphertext-plaintext matrix multiplications. The concrete savings are in App. F.

5. Non-Linear Layers

Securing the non-linear functions in BERT inference presents another challenge due to their complexity in cryptographic primitives. Iron demonstrated that approximately 75% of the total execution time is from non-linear layers.

In this section, we introduce our novel, accurate, and efficient protocols for non-linear operations. Iron does not consider specific properties of the computed functions like symmetry and linearity of GELU, Tanh and $\mathsf{Softmax}$, resulting in huge communication costs. Our accurate approximation design uses these properties of the non-linear functions to significantly reduce the communication overhead (§ 5.2, § 5.3, § 5.4). As an additional optimization, we use Motzkin's polynomial pre-processing [51], [40] (§ 5.1). This method reduces the number of required multiplications by approximately half and can be applied to generic polynomial evaluation where the function is publicly known.

5.1. Motzkin's Polynomial Pre-processing

Horner's method [19], [73] is a popular technique for evaluating polynomials as it is proven to optimal regarding the number of operations, if the polynomial's coefficients are not known in advance [53], [54]. Horner's method requires $n-1$ multiplications for evaluating a polynomial of degree n. But in many applications, such as ML inference, the non-linear functions and their polynomial approximations are publicly available. In such cases, Motzkin [51] showed how to pre-process or reformulate a polynomial so that it only requires approximately $\lceil \frac{n}{2} \rceil$ multiplications [54]. We present the detailed Motzkin's polynomial pre-processing procedure in App. B.

5.2. Accurate GELU Approximation

To the best of our knowledge, Iron [28] proposes the state-of-the-art accurate approximations for GELU (cf. Eq. 4) in MPC. They approximate GELU using the following equation:

$$\mathsf{GELU}(x) = \frac{1}{2}x(1 + \mathsf{Tanh}(\sqrt{2/\pi}(x + 0.04471x^3)))$$

The core part of Iron's approximation is Tanh, they reuse building blocks from SIRNN [59]. Iron further optimized Tanh by leveraging the function's symmetry, such that the evaluation only has to be done on negative inputs. This, however, is still quite costly, as reported by Iron, secure GELU evaluations account for more than 50% of the runtime.

We present a more performant approximation using a 4-degree polynomial. The concrete polynomial parameters can be found in App. C. Multiple observations and design choices lead to our design.

Linearity: We observe that GELU has good linearity when the input is relatively large or small. Given that $\lim_{x\to\infty} \mathsf{erf}(\frac{x}{\sqrt{2}}) = 1$, we have $\lim_{x\to\infty} \mathsf{GELU}(x) = x$. And also $\lim_{x\to-\infty} \mathsf{GELU}(x) = 0$. Thus, we only need accurate approximations for an inputs range. In practice, setting the range to be $[-2.7, 2.7]$ is sufficient to guarantee 1×10^{-3} average absolute errors.

Symmetry: Next, we revisit the original GELU definition [31]: $\mathsf{GELU}(x) = \frac{1}{2}x[1 + \mathsf{erf}(x/\sqrt{2})]$. We observe the symmetry of $g(x) = x \cdot \mathsf{erf}(x/\sqrt{2})$, given that $g(x) = g(-x)$. Thus, we only need to approximate $g(x)$ for half

of the input range, i.e., positive input values. These observations enable us to utilize low-degree polynomial approximations to accurately compute $g(x)$, and thus $\mathsf{GELU}(x)$.

Accuracy: The state-of-the-art work for approximating the plaintext error function $\mathsf{erf}(x)$ on integers for GELU is I-BERT [38]. We take a further step upon their method by approximating $x \cdot \mathsf{erf}(x)$ instead of $\mathsf{erf}(x)$, which enables us to also apply polynomial pre-processing (§5.1) to reach the optimal number of multiplications. Otherwise, if we adopt I-BERT's strategy to approximate $\mathsf{erf}(x)$ in MPC, the polynomial pre-processing will not give us the optimal number of multiplications, because the approximation will be first evaluated on $|x|$, but then multiplied by x. I-BERT used polynomial interpolation techniques [67] with grid-search to determine the polynomial coefficients, while we utilize the Remez method [61] to find the optimal polynomial coefficients. Unlike polynomial interpolation with grid-search, the Remez method guarantees to find an *optimal* polynomial approximation by iteratively adjusting the polynomial coefficients to minimize the maximum error (also known as the maximum deviation) between the polynomial and the target function over the specified interval.

$$\mathsf{ApproxGELU}(x) = \begin{cases} x & \text{if } x > 2.7 \\ a|x|^4 + b|x|^3 + c|x|^2 + & \\ \quad d|x| + e + 0.5x & \text{if } |x| \le 2.7 \\ 0 & \text{if } x < -2.7 \end{cases} \tag{5}$$

Polynomial pre-processing. We further optimize our polynomial evaluation using Motzkin's polynomial pre-processing (§5.1). This effectively reduces the evaluation cost to only two multiplications among secret-shared values instead of three.

$$\begin{aligned} \mathrm{GELU}_{\mathrm{P_0}}(x) =& (g_0|x| + g_1) \cdot |x| + g_2 \\ \mathrm{GELU}_{\mathrm{P_1}}(x) =& (\mathrm{GELU}_{\mathrm{P_0}}(x) + g_0|x| + g_3) \cdot |x| \\ & + g_4 + 0.5x \end{aligned} \tag{6}$$

The average ULP error [59] of our GELU approximation is 4 with scale 12, corresponding to 9.77×10^{-4} error in floating-point, with the input range $[-5, 5]$.

Two comparisons in one shot. To further optimize the efficiency, we additionally tweak the implementation of the polynomial as follows: Naively, we need a total of two comparisons in order to pick the correct interval. We observe that we can remove one of the comparisons. Recall that our approximation in Eq. 6 requires to determine the absolute value $|x| = \mathsf{MSB}(x)? - x : x$. Using the absolute value, we do not need to explicitly distinguish the upper and the lower interval but only compare $\mathrm{b} = |x| > 2.7$. This allows us to output $\mathrm{res} = \mathrm{b}?\mathsf{RELU}(\mathsf{x}) : \mathsf{GELU}_{P_1}(x)$, where $\mathsf{RELU}(\mathsf{x}) = (|x| + x) >> 1$.

5.3. Accurate Tanh Approximation

$\mathsf{Tanh}(x) = \frac{e^{2x}-1}{e^{2x}+1}$ is a hyperbolic function often used in neural networks, primarily because of its symmetry, smoothness, and non-linearity. However, it requires evaluating the exponential function and determining the reciprocal of a secret-shared input, both being costly operations when evaluated securely [59].

Iron instantiates Tanh by combining lookup-table (LUT)-based private protocols for those two non-linear functions from SIRNN [59]. However, evaluating Tanh remains costly due to the long bit length of input.

Similar to our optimizations in GELU, we take the symmetry into consideration and utilize the Remez method (§5.2) to design a polynomial approximation of degree five and three intervals (as the Tanh is an odd symmetric function, we only need 2 comparisons). The concrete polynomial parameters can be found in App. C.

$$\mathsf{ApproxTanh}(x) = \begin{cases} 1 & \text{if } x > 2.855 \\ ax^5 + bx^4 + cx^3 + & \\ \quad dx^2 + ex + f & \text{if } 0 \le x \le 2.855 \\ -\mathsf{ApproxTanh}(-\mathsf{x}) & \text{if } x < 0 \end{cases} \tag{7}$$

Polynomial pre-processing. Again, we apply Motzkin's polynomial pre-processing (§5.1) to reduce the number of required multiplications between secret-shared values to 3:

$$\begin{aligned} \mathrm{TanhP}_0(x) =& (x + t_0) \cdot x + t_1 \\ \mathrm{TanhP}_1(x) =& (\mathrm{TanhP}_0(x) + x + t_2) \cdot \\ & \mathrm{TanhP}_0(x) \cdot t_3 x + t_4 x + t_5, \end{aligned} \tag{8}$$

The average ULP error [59] of our Tanh approximation is 27 with scale 12, corresponding to 6.59×10^{-3} error in floating-point, with the input range $[-5, 5]$.

5.4. Efficient Softmax in MPC

Similar to Tanh, $\mathsf{Softmax}$ (Eq. 2) also requires evaluating the exponential function and computing a division, both inherently require floating point operations. Transformers typically use 32-bit floating-point values, while MPC floating-point protocols are imprecise and much less efficient than integer computations [58], [16].

Iron solves this by computing $\mathsf{Softmax}$ on fixed-point values utilizing LUT-based protocols from SIRNN. The LUT-based method is inefficient in communication especially when the input has a long bit length, as we will show in §7.

Inspired by the plaintext integer-only approximation in I-BERT, we design a secure protocol for the exponential function with low-degree polynomials and without LUTs:

Shifting by x_{max}: Similar to Iron, I-BERT, and most plaintext implementations of $\mathsf{Softmax}$, we normalize each input value $\boxed{x_i}$, $i \in [d]$, where d is the dimension of $\boxed{\mathsf{x}}$. So that all inputs to the exponential function become negative. The normalize value is $\boxed{\mathfrak{e}_i}$, where $\boxed{\mathfrak{e}_i} = \boxed{x_i} - \boxed{x_{max}}$.

Secure integer-only exponential function: As demonstrated by I-BERT, any non-positive number can be decomposed as $\boxed{\mathfrak{e}_i} = (-\ln 2) \cdot \boxed{z} + \boxed{p}$, where $\boxed{z}$ is a non-negative integer and $\boxed{p} \in (-\ln 2, 0]$. It follows that

Algorithm 1 Secure Integer-only Exponential Function

1: **function** EXP($\boxed{\mathfrak{e}_i}, s$) ▷ normalized input, scale
2: $\text{invNegLt} = -\frac{1}{\ln 2}$
3: $\boxed{z} = (\boxed{\mathfrak{e}_i} \cdot \text{invNegLt}) >> s$
4: $\boxed{p} = \text{reduce}(\boxed{z} \cdot \ln 2 + \boxed{\mathfrak{e}_i}, s+2)$
5: $\boxed{z} = \mathsf{Clip}(\boxed{z}, [0, s+1])$
6: **return** $(0.385 * (\boxed{p} + 1.353)^2 + 0.344) >> z$

$\mathsf{exp}(\boxed{\mathfrak{e}_i}) = \mathsf{exp}(\boxed{p}) >> \boxed{z}$. To compute $\mathsf{exp}(\boxed{p})$, I-BERT provides a plaintext approximation for the exponential function in $\boxed{p} \in (-\ln 2, 0]$. Since the range of $\boxed{p}$ is relatively small, we can use a 2-degree polynomial approximation:

$$\mathsf{exp}(\boxed{p}) \approx 0.3585(\boxed{p} + 1.353)^2 + 0.344 \quad (9)$$

We present our protocol in Alg. 1. To extract the integer $\boxed{z}$ in Line 3, we shift $\boxed{\mathfrak{e}_i}/(-\ln 2)$ by the scale s. In Line 4, we compute $\boxed{p}$ by computing the sum of $\ln 2 \cdot \boxed{z}$ and $\boxed{\mathfrak{e}_i}$. As $\boxed{p} \in (-\ln 2, 0]$, we can safely reduce the bit width here by $s+2$ to enhance efficiency, where s is the public scale of the fixed-point representation §7. After we compute $\boxed{p}$, we will evaluate the polynomial in Eq. 9. And then, the result will be shifted by $\boxed{z}$ steps. Since the output of Eq. 9 is bounded by $(0,\ 0.7)$, we can safely clip $\boxed{z}$ to the range $[0, s+1]$ before shifting in Line 5. $\boxed{z}$ is a secret-shared value and cannot be released, we thus apply a sequence of MUX to get the correct shifting result in Line 6, which will be the result of $\mathsf{exp}(\boxed{\mathfrak{e}_i})$. The average ULP error [59] is 0.0059 with scale 12, corresponding to error 1×10^{-6} in floating-point, with input range $[-1000, -0.001]$.

Now we have the secure protocol for evaluating exponential function, in Softmax Eq. 2, we also need to evaluate the reciprocal of the summed exponential function's outputs. However, compared to exponential function, reciprocal is only called on one dimension of the input matrix instead of the entire matrix. Thus, it occupies much less computation cost compared to Softmax, so we reuse the state-of-the-art secure reciprocal protocol from SIRNN and now we can securely and efficiently evaluate Softmax.

6. Machine Learning Optimizations

BOLT develops a secure, end-to-end protocol for inference via a co-design of cryptography and machine learning, and in this section we explain our machine learning optimizations designed to enhance BOLT's accuracy and efficiency. We introduce a word elimination technique to significantly improve BOLT's performance along with secure computation-aware fine-tuning for maintaining high accuracy.

6.1. Oblivious Word Elimination

In this section, we introduce a *word elimination* (W.E.) technique that is applicable to all encoder transformers like BERT, while maintaining high accuracy. Our technique is inspired by the observation in [26] that not every token in the input sentence significantly impacts the model's inference result. Therefore, by eliminating tokens with minor contributions, we can reduce the dimensions of the input matrix and enhance inference efficiency.

Transformers' attention mechanisms offer an ideal metric for ranking input tokens' contributions without gradient computation. The product of query and key matrices ($\mathbf{Q}_h \times \mathbf{K}_h \in \mathsf{R}^{m \times m}$) estimates the correlation among m^2 pairs of the m input tokens, with larger values indicating higher correlation. Summing $\mathbf{Q}_h \times \mathbf{K}_h$ across one axis yields a score vector $\mathsf{s} \in \mathsf{R}^m$. This score vector forms key-value pairs with the input token sequence, allowing us to rank tokens by their scores and discard those contributing minimally.

Based on these observations, we devise an oblivious sorting method to eliminate words with fewer contributions. Specifically, we use bitonic sorting [33] to determine the median of the scores. And then, tokens scoring below the median are eliminated using another round of bitonic sorting. The comprehensive procedure is outlined in Alg. 2 of App. D.

As shown in §7, word elimination introduces negligible accuracy drops, and can significantly improve the performance of BOLT.

6.2. Secure Computation-Aware Fine-Tuning

As we mentioned in §2, we use fixed-point representations in BOLT, which creates a gap between the floating-point model and its fixed-point representation. We aim to maintain high accuracy of the model using reasonably small scales, since large scales and floating-points increase the computation overhead in MPC. To counteract the accuracy loss from this gap, we apply quantization-aware fine-tuning [75]. Like many prior quantization-aware training methods [38], we use symmetric and static quantization to calculate the quantized result of each function during forward propagation. As the quantization process is not differentiable, we simulate the backpropagation by computing the real gradients and then quantizing the gradients. We also consider approximation errors of non-linear functions when fine-tuning the model. Specifically, we substitute floating-point functions of GELU, Softmax, and Tanh with our approximated versions prior to quantization-aware fine-tuning.

Our approximated functions closely match the originals, eliminating the need for additional distillation processes to enhance model accuracy as in MPCFormer [44]. Distillation might reduce accuracy, especially with smaller datasets. As demonstrated in MPCFormer, there are over 5% drops in accuracy on small datasets. In contrast, BOLT's accuracy decline is negligible (around 1%).

To further boost the accuracy of the model, we also take the word elimination into consideration when we fine-tune the model. That is, during the forward propagation, the tokens are dynamically eliminated and the backward propagation can automatically compute the gradients. We note that

fine-tuning is done only once per model, and so its cost can be amortized across many inferences. In the upcoming §7, we will demonstrate that our secure computation-aware fine-tuning sustains BOLT with an accuracy level comparable to floating-point baselines.

7. Evaluation

7.1. Implementation

7.1.1. Libraries and configurations. We implement BOLT using the Secure and Correct Inference (SCI) library from EzPC [11], [58], [60] for ring secret sharing and SEAL [65] library for HE. For Softmax and LayerNorm, we use bit-length $l = 37$ and scale $s = 12$. For GELU, we use bit-length $l = 21$ and scale $s = 11$. The scales of linear layers are consistent with the secure computation-aware fine-tuning §6.2. Please refer to App. E for more details. In BOLT, we choose our parameters according to the standard from [3] and use 128-bit security. Specifically, we choose $n = 8192$, $\log p \approx 29, \log q \approx 218$ for Linear 1 and Softmax $\times$ **V** of Fig. 1. And we choose $\log p \approx 19, \log q \approx 180$ for other linear layers. Because we have ciphertext-ciphertext matrix multiplications in Linear 1 and Softmax$\times$**V**, we need more noise budget and larger scale compared to the layers that only require ciphertext-plaintext matrix multiplications. Thus, we use larger p and q for Linear 1 and Softmax $\times$ **V**. To guarantee circuit privacy, the server performs noise flooding [48], [43], which adds a large noise to the ciphertext before returning it to the client. The noise is chosen large enough to hide additional information in the ciphertext but still guarantee the accuracy of the decryption.

7.1.2. Conversion between MPC and HE. Computation in MPC is performed over a 2^l ring while HE is over a prime. Therefore we need conversion when using results from HE in MPC, and vice versa. Each round of conversion consists of a comparison and a multiplexer. However, when converting from MPC back to HE, the probability that the sharings overflow is $\frac{|x|}{2^l}$, where $|x|$ is the absolute shared secret value. When $|x|$ is small or the ring size is large enough, we can omit the conversion from MPC to HE without affecting the accuracy. Therefore, in our implementation, we only perform conversion from HE to MPC. To reduce the error rate, we extend the result of GELU from 21 bits to 37 bits before converting to HE.

7.1.3. Implementation optimizations. In HE, we perform lazy relinearization to reduce the total number of relinearizations. To reduce the communication cost, we use modulus switching before sending back the ciphertexts to the client. We optimize Layernorm by moving the multiplication with the model weights to HE and by precomputing MSB of $(\mathsf{X}_{i,j} - \ _i)$ to further reduce the communication cost of the rest multiplications.

7.1.4. Iron's System. Because Iron [28] is not open-sourced, we implement Iron's end-to-end system following the protocols described in their paper. We use bit-length $l = 37$ and scale $s = 12$ for secret sharing. We configure the HE with $n = 8192$, $\log q \approx 180$, and $\log p = 37$ to enable noise flooding. Note that Iron requires a larger scale to maintain the model's accuracy as they did not incorporate secure computation-aware fine-tuning. Furthermore, the optimization for LayerNorm in Iron is wrong, which will cause a significant accuracy drop (close to random guessing), as they will break the residual architecture. Thus, we remove Iron's optimization for LayerNorm. We provide detailed discussions on the flaws in Iron's LayerNorm optimization in App. G. We have confirmed the implementations and the performance with the authors of Iron.

Dataset	Plaintext	Iron	w/o W.E.	w/ W.E.
MRPC	90.00 ± 0.23	89.87	90.53	89.95
RTE	69.70 ± 1.50	70.76	69.68	69.31
SST-2	92.36 ± 0.59	92.77	91.74	92.78
STS-B	89.62 ± 0.31	89.41	87.97	88.44

TABLE 2: Acc. of floating-point plaintext, Iron and BOLT.

7.2. Experimental Setup

We evaluate our experiments on AWS EC2 using two c6i.16xlarge instances with 64 vCPUs and 128 GB memory. We use Linux Traffic Control (tc) to simulate different network settings. Under the LAN scenario, we set the bandwidth to 3Gbps and the round-trip latency to 0.8 ms. Our setup for the WAN network consisted of four settings: {100Mbps, 40ms}, {100Mbps, 80ms}, {200Mbps, 40ms}, and {200Mbps, 80ms}. These configurations allow us to evaluate the system's performance under varying bandwidths and latencies. We set the number of threads to 32.

To evaluate the accuracy of our system, we test on four datasets from GLUE benchmark [72], which is widely used to evaluate BERT's performance. These datasets include 3 classification tasks: MRPC, RTE, SST-2, and a regression task: STS-B. Specifically, our fine-tuning is performed on the training sets and the accuracy is evaluated on the official validation sets.

7.3. Accuracy

In Tab. 2, we compare the accuracy of BOLT, with and without word elimination, to that of floating-point plaintext results and Iron. Similar to prior works, we present the F1 score for MRPC, Pearson correlation for STS-B and accuracy for RTE and SST-2. In the second column, we present the accuracy results of the plaintext model on floating-point reported by prior works [76] and in the form of Avg ± STD. Overall, the accuracy achieved by BOLT matches the accuracy of plaintext models on all datasets, with a maximum accuracy loss of 1.3% on the STS-B dataset. Without word elimination (W.E.), we observe 1.04% accuracy drop on SST-2 compared to BOLT with W.E. We attribute this to the conversion approximation, described in §7.1.2, from 2^l ring to prime field. Without W.E., the

	Iron		BOLT w/o W.E.				BOLT w/ W.E.			
Component	**Comm. (MB)**	**Round**	**Comm. (MB)**		**Round**		**Comm. (MB)**		**Round**	
Linear 1	4844.14	38	7.06	686.1×	2	19×	3.18	1524.7×	2	19×
Softmax×V	4918.38	36	9.88	497.6×	2	18×	2.82	1741.6×	2	18×
Linear 2	47.65	2	4.51	10.6×	2	1×	2.26	15.0×	2	1×
Linear 3	95.40	2	9.01	10.6×	2	1×	4.50	21.2×	2	1×
Linear 4	95.21	2	13.52	7.0×	2	1×	6.77	14.1×	2	1×
Softmax	3596.32	252	1447.65	2.5×	232	1.1×	450.74	8.0×	229	1.1×
GELU	7960.00	256	1471.67	5.4×	88	2.9×	776.84	10.2×	88	2.9×
LayerNorm	871.46	218	599.40	1.5×	220	0.99×	290.55	3.0×	220	0.99×
Tanh	20.67	150	16.64	1.2×	110	1.4×	16.64	1.2×	110	1.4×
end-to-end	280.99 GB	13663	59.61 GB	4.71×	10509	1.30×	25.74 GB	10.91×	10901	1.25×

TABLE 3: Communication cost and rounds comparing Iron with BOLT. The costs of the components are for one layer, and there are 12 layers in BERT.

dimensions of matrices in the model are twice as large as the model with W.E. Therefore, given the same error rate, we expect a higher probability of error occurring during the conversion. However, as the results indicate, the conversion approximation has negligible influence on the accuracy. To further improve the accuracy, users can use correct truncation at the cost of doubling the conversion cost in BOLT. This will incur around 3%-7% performance overhead, estimated based on the results in Fig. 5.

7.4. Communication Analysis

In this section, we present the communication cost in terms of the number of bytes and the number of rounds for each component in BOLT. Tab. 3 shows the cost breakdown of each component in a single encoder layer as well as the end-to-end total cost. Additionally, we present the improvement compared to Iron on the right side of the table.

To illustrate the improvement originated from our efficient MatMul and nonlinear functions, we first compare Iron with BOLT w/o W.E. in the following.

7.4.1. Matrix Multiplication. BOLT w/o W.E. achieves 686.1× and 497.6× less communication on Linear 1 and Softmax × **V**. This is because Iron performs ciphertext-ciphertext matrix multiplication using secret sharing, while BOLT computes multiplication between encrypted matrices in HE.

We also achieve 7.0-10.6× less communication on ciphertext-plaintext matrix multiplication. The improvement comes from fewer ciphertexts from compact encoding and smaller ciphertext size from modulus switching.

7.4.2. Non-linear Functions. The expensive operations in non-linear components of Iron are Softmax and GELU, which consume 3.6 GB and 7.9 GB per layer, respectively. BOLT uses 2.5× less communication on Softmax and 5.4× on GELU. For LayerNorm, we achieve 1.5× less communication, which comes from multiplication with model weights in HE and MSB optimization (see §7.1). Additionally, BOLT has fewer communication rounds for Softmax, GELU and Tanh.

7.4.3. Word Elimination. For most of the components in BOLT, word elimination gives us another 2× less communication because the input matrix row size m becomes half of the original size. The cost of Softmax is further reduced by 3.2× as the input dimension of Softmax is $m \times m$. As expected, with W.E., BOLT has more than 2× communication improvement compared to BOLT w/o W.E. (25.74 GB vs. 59.61 GB).

Overall, we reduce nearly 11× in terms of communication cost and 1.3× in terms of communication rounds, which is expected to improve the end-to-end inference performance significantly under practical network conditions.

7.5. End-to-End Performance

We present the end-to-end run-time under various network settings in Fig. 5. With W.E., BOLT consistently outperforms Iron by a magnitude of 4.8× to 9.5×, while without W.E., BOLT still outperforms Iron by 2.6× to 4.1×. In the following, we provide a detailed analysis of the savings in linear and non-linear layers.

7.5.1. Matrix Multiplication. As discussed in §7.4, our linear layers consume much less communication compared to Iron as we have compact packing and smaller ciphertext size and Iron's packing is much more sparse. Thus, we have more run-time gains in the WAN setting where the network is slower. For instance, for Linear 1, BOLT is 45.7× faster w/o W.E. under 100 Mbps, 40 ms, and 88.2× faster w/ W.E. This saving is less significant in the LAN setting, but we note that BOLT is still consistently better than Iron. For example, under 3 Gbps, 0.8 ms network, BOLT w/o W.E. is 2.3× faster than Iron, and BOLT w/ W.E. is 4.6× faster for Linear 1.

BOLT is consistently faster than Iron for Softmax × **V** under different network settings because BOLT utilizes efficient ciphertext-ciphertext matrix multiplication protocols while Iron adopts secret sharing-based matrix multiplications. For instance, even in the LAN setting (3 Gbps, 0.8 ms), BOLT w/o W.E. is 2.5× faster, and BOLT w/ W.E. is 6.1× faster than Iron. In the WAN setting (100 Mbps, 40 ms), BOLT w/o W.E. is 68.3× faster, and BOLT w/ W.E. is 158.0× faster than Iron.

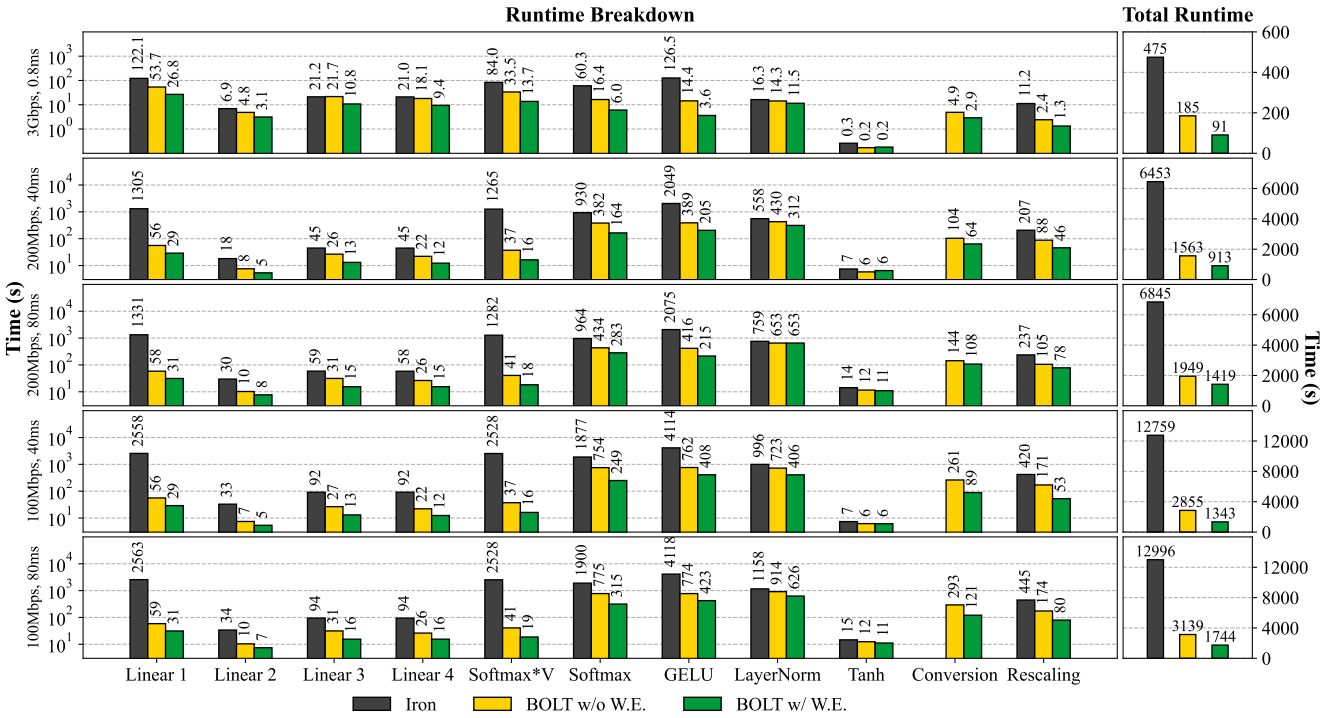

Figure 5: End-to-end inference performance under various network settings, and its breakdown.

7.5.2. Non-linear Functions. Our run-time observations for non-linear functions are consistent with our communication analysis in §7.4, since the major bottleneck for non-linear is communication. Specifically, for Softmax, in the LAN setting (3 Gbps, 0.8 ms), BOLT w/o W.E. is 3.7× faster and BOLT w/ W.E. is 10.0× faster than Iron. In the WAN setting (100 Mbps, 40 ms), BOLT w/o W.E. is 2.5× and BOLT w/ W.E. is 7.5× faster. For GELU, in the LAN setting (3 Gbps, 0.8 ms), BOLT w/o W.E. is 8.8× and BOLT w/ W.E. is 35.1× faster than Iron. In the WAN setting (100 Mbps, 40 ms), BOLT w/o W.E. is 5.4× and BOLT w/ W.E. is 10.1× faster. We notice that BOLT has less magnitude improvements in the WAN setting for non-linear functions and we deem that this is caused by the latency of the network. Our non-linear significantly saves communication costs, but our improvement on communication rounds is less significant (0.99-2.9×). Thus, in the WAN setting where the network latency is high, our run-time improvement is becoming less compared to the LAN setting. Despite this improvement reduction under certain conditions, BOLT still greatly outperforms the state-of-the-art system.

7.5.3. Conversion, Rescaling, and Word Elimination. As shown in Fig. 5, Iron doesn't have conversion costs, because Iron performs HE and MPC computation over the same ring. With our conversion approximation, the conversion overhead is around 2-9% of the total run-time. Considering the savings of HE over the prime field, including compact packing and fewer rotations, such overhead is affordable.

Rescaling is used in both systems to reduce the scale after HE's computation. In BOLT, the input scale for each linear layer varies based on the secure computation-aware fine-tuning, while Iron fixed the input scale to 12. Therefore, compared to Iron, BOLT needs rescaling before converting the result from secret sharing back to HE. However, BOLT has lower rescaling costs because BOLT uses a smaller rescaling factor than Iron. After each HE layer, the output scale for Iron is 24. Therefore it requires a right shift of 12 to scale back. For BOLT, taking Linear 3 as an example, the output scale is already 11. Therefore, no rescaling is required for this layer. The result also shows that BOLT has at least 2.3× speed-up in terms of rescaling.

Word elimination accounts for 1%-5% overhead, depending on the network condition. Word elimination is cheap from the end-to-end performance perspective, as the size of our target sorting array is only m ($m = 128$ in our experiments), and only two rounds of sorting are performed during each inference.

8. Related Work

Privacy-preserving Neural Network Inference. Due to the rapidly growing concerns about data privacy in ML, significant efforts have been made to design efficient cryptographic protocols that securely evaluate ML algorithms, e.g., [39], [77], [29], [41], [78], [50]. Especially private protocols for neural network inference, e.g. [32], [25], [49], [35], [41], [60], [4], [34], were intensively investigated due to their wide-spread applicability and interesting structure consisting of various linear and non-linear computations.

Cryptonets [25] proposed one of the first protocols for HE-based private neural network inference. Gazelle [35], Delphi [49], and Cheetah [32] are hybrid 2PC neural network inference protocols combining HE for matrix-vector multiplications with MPC for the secure evaluation of non-linear activation functions. Cryptflow [41] is a compiler to automatically convert plaintext neural networks from the state-of-the-art ML framework TensorFlow in MPC protocols. CrypTFlow2 [60] and SiRNN [59] optimize efficiency for 2PC protocols of math functions typically used in neural networks, e.g., reciprocal or Tanh.

Private Transformers. Secure evaluation of transformers is very challenging due to their significantly larger size compared to traditional neural networks. So far, several works have investigated how to realize transformer inference in a privacy-preserving manner: THE-X [12], Iron [28], MPCFormer [44], PrivFormer [2], and PUMA [18].

THE-X [12] evaluates BERT-tiny [70] (a distilled student model of BERT with 4.4 million parameters) under HE using HE-friendly replacements for GELU, Softmax, and LayerNorm as well as dropping pooling layers. Thereby, it, however, inherently reduces accuracy and leaks the comparison result of each ReLU evaluation — which is part of the GELU and Softmax approximation — to the client. Iron [28] builds up on Cheetah [32], a HE-based private transformer inference system, and improves its matrix-matrix multiplication with a more efficient packing that encodes multiple matrix rows in one ciphertext. Both Cheetah [32] and Iron [28] do not optimally leverage all ciphertext slots in the matrix-matrix multiplications as a large number of polynomial coefficients are set to zero. MPCFormer [44] proposes a private inference system for the BERT-base model using arithmetic secret-sharing. It employs knowledge distillation to cure the accuracy drop caused by MPC-friendly approximations (inspired by previous works [14], [50]) for the non-linear functions. Moreover, MPCFormer requires significant effort to conduct additional knowledge distillation on the large models that have been well-trained, which has a significant accuracy drop (greater than 5%) on smaller datasets like RTE. Two recent works, PrivFormer [2], and PUMA [18] proposed secure inference for transformers in 3PC settings instead of 2PC. They rely on the honest-majority assumption and the setting is different from ours 2PC.

9. Discussion & Conclusion

Generalizability of BOLT. As stated in § 3, the cryptographic techniques of BOLT are applicable to all transformer encoders, and we use BERT as an example to demonstrate the efficiency of our system. Transformer decoders like GPT [57] share the same architecture as encoder models. Therefore, BOLT without word elimination is directly applicable to transformer decoders. Algorithms from Flash Attention [15] are also compatible with BOLT but do not significantly speed up operations since they optimize GPU memory access during attention computation, whereas BOLT runs on the CPU and its bottlenecks are communication and HE computations rather than memory access. However, BOLT should still outperform Iron for the variants of transformers with a significant margin similar to BERT. Further optimizing BOLT's computation from a hardware perspective is an interesting direction for future work.

Security Argument & Inference with Malicious Security. As mentioned in § 3, our threat model assumes semi-honest client and server. Our methods described in § 4 and § 5 are based on established cryptographic protocol building blocks (using combinations of secret sharing-based MPC and HE from the EzPC framework [11]) that were already proven secure. As we do not propose new protocols, but combine them in novel ways, the security of BOLT follows naturally from the security of these building blocks.

Supporting malicious security (either for both parties or for clients only) is a very interesting direction for future work. It is challenging to adapt techniques from current state-of-the-art malicious secure protocols. For instance, adapting linear layer protocols' techniques from MUSE [42] may be expensive as they use zero-knowledge proofs for HE. Prior works like MUSE [42] and SIMC [10] support the ReLU and ReLU6 non-linear functions that are very simple and involve only comparisons. The non-linear functions in transformers are much more complex and could be costly to compute in protocols with malicious security.

Performance in Extremely Fast Networks. The run-time savings of BOLT compared to Iron become less significant in extremely low-latency network settings. We tested the end-to-end run-time of BOLT and Iron in an extremely fast network setting, with bandwidth 25 Gbps and latency 0.3 ms. BOLT without word elimination is 1.50× faster and BOLT with word elimination is 2.63× faster than Iron. In this network setting, our linear layers' run-time is 0.98-1.44× faster than Iron for BOLT without word elimination, and 1.98-2.70× faster for BOLT with word elimination. An interesting future direction would be a hybrid protocol that can automatically choose linear protocols based on the network setting to leverage our efficient communication and Iron's fast computation.

Conclusion. We propose BOLT, a privacy-preserving, accurate, and efficient inference protocol for transformers. BOLT incorporates optimizations from both cryptographic primitives and machine learning. These optimizations significantly enhance BOLT's inference performance, advancing towards practical secure inference for transformers.

Acknowledgements

We thank the anonymous shepherd and reviewers for their valuable feedback. We appreciate Iron's authors' help in explaining their implementation and confirming our results. This project received funding from Google Research Scholar 2023 and the ERC under the European Union's Horizon 2020 research and innovation program (grant agreement No. 850990 PSOTI). It was co-funded by the DFG within SFB 1119 CROSSING/236615297 and GRK 2050 Privacy & Trust/251805230. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors, and do not necessarily reflect the views of the sponsors.

References

[1] Openai chatgpt. https://openai.com/blog/chatgpt.

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

[3] Martin Albrecht, Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Shai Halevi, Jeffrey Hoffstein, Kim Laine, Kristin Lauter, et al. Homomorphic encryption standard. *Protecting privacy through homomorphic encryption*, 2021.

[4] Mauro Barni, Pierluigi Failla, Riccardo Lazzeretti, Ahmad-Reza Sadeghi, and Thomas Schneider. Privacy-preserving ECG classification with branching programs and neural networks. *IEEE Transactions on Information Forensics and Security (TIFS)*, 2011.

[5] Som S Biswas. Role of chat GPT in public health. *Annals of Biomedical Engineering*, 2023.

[6] Fabian Boemer, Rosario Cammarota, Daniel Demmler, Thomas Schneider, and Hossein Yalame. MP2ML: A mixed-protocol machine learning framework for private inference. In *ARES*, 2020.

[7] Jean-Philippe Bossuat, Christian Mouchet, Juan Troncoso-Pastoriza, and Jean-Pierre Hubaux. Efficient bootstrapping for approximate homomorphic encryption with non-sparse keys. In *EUROCRYPT*, 2021.

[8] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical GapSVP. In *CRYPTO*, 2012.

[9] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory*, 2014.

[10] Nishanth Chandran, Divya Gupta, Sai Lakshmi Bhavana Obbattu, and Akash Shah. {SIMC}:{ML} inference secure against malicious clients at {Semi-Honest} cost. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 1361–1378, 2022.

[11] Nishanth Chandran, Divya Gupta, Aseem Rastogi, Rahul Sharma, and Shardul Tripathi. EzPC: programmable, efficient, and scalable secure two-party computation for machine learning. Cryptology ePrint Archive, Paper 2012/144, 2017. https://eprint.iacr.org/2017/1109.

[12] Tianyu Chen, Hangbo Bao, Shaohan Huang, Li Dong, Binxing Jiao, Daxin Jiang, Haoyi Zhou, and Jianxin Li. THE-X: Privacy-preserving transformer inference with homomorphic encryption. *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2022.

[13] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In *ASIACRYPT*, 2017.

[14] Edward Chou, Josh Beal, Daniel Levy, Serena Yeung, Albert Haque, and Li Fei-Fei. Faster cryptonets: Leveraging sparsity for real-world encrypted inference. *arXiv preprint arXiv:1811.09953*, 2018.

[15] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022.

[16] Daniel Demmler, Ghada Dessouky, Farinaz Koushanfar, Ahmad-Reza Sadeghi, Thomas Schneider, and Shaza Zeitouni. Automated synthesis of optimized circuits for secure computation. In *CCS*, 2015.

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

[18] Ye Dong, Wen-jie Lu, Yancheng Zheng, Haoqi Wu, Derun Zhao, Jin Tan, Zhicong Huang, Cheng Hong, Tao Wei, and Wenguang Cheng. Puma: Secure inference of llama-7b in five minutes. *arXiv preprint arXiv:2307.12533*, 2023.

[19] William S Dorn. Generalizations of horner's rule for polynomial evaluation. *IBM Journal of Research and Development*, 1962.

[20] Cynthia Dwork, Aaron Roth, et al. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 2014.

[21] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Paper 2012/144, 2012. https://eprint.iacr.org/2012/144.

[22] Karthik Garimella, Zahra Ghodsi, Nandan Kumar Jha, Siddharth Garg, and Brandon Reagen. Characterizing and optimizing end-to-end systems for private inference. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, pages 89–104, 2023.

[23] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, 2009.

[24] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *CRYPTO*, 2013.

[25] 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 *ICML*, 2016.

[26] Saurabh Goyal, Anamitra Roy Choudhury, Saurabh Raje, Venkatesan Chakaravarthy, Yogish Sabharwal, and Ashish Verma. Power-bert: Accelerating bert inference via progressive word-vector elimination. In *International Conference on Machine Learning*, pages 3690–3699. PMLR, 2020.

[27] Shai Halevi and Victor Shoup. Bootstrapping for Helib. *Journal of Cryptology*, 2021.

[28] Meng Hao, Hongwei Li, Hanxiao Chen, Pengzhi Xing, Guowen Xu, and Tianwei Zhang. Iron: Private inference on transformers. In *NeurIPS*, 2022.

[29] Aditya Hegde, Helen Möllering, Thomas Schneider, and Hossein Yalame. SoK: Efficient privacy-preserving clustering. *PoPETs*, 2021.

[30] Dan Hendrycks and Kevin Gimpel. Bridging nonlinearities and stochastic regularizers with gaussian error linear units. *arXiv preprint arXiv:1606.08415*, 2016.

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

[32] Zhicong Huang, Wen jie Lu, Cheng Hong, and Jiansheng Ding. Cheetah: Lean and fast secure Two-Party deep neural network inference. In *USENIX Security*, 2022.

[33] Mihai F Ionescu and Klaus E Schauser. Optimizing parallel bitonic sort. In *Proceedings 11th International Parallel Processing Symposium*, pages 303–309. IEEE, 1997.

[34] Xiaoqian Jiang, Miran Kim, Kristin Lauter, and Yongsoo Song. Secure outsourced matrix computation and application to neural networks. In *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, pages 1209–1222, 2018.

[35] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. GAZELLE: A low latency framework for secure neural network inference. In *USENIX Security*, 2018.

[36] Jongmin Kim, Gwangho Lee, Sangpyo Kim, Gina Sohn, Minsoo Rhu, John Kim, and Jung Ho Ahn. Ark: Fully homomorphic encryption accelerator with runtime data generation and inter-operation key reuse. In *IEEE/ACM International Symposium on Microarchitecture*, 2022.

[37] Sangpyo Kim, Jongmin Kim, Michael Jaemin Kim, Wonkyung Jung, John Kim, Minsoo Rhu, and Jung Ho Ahn. Bts: An accelerator for bootstrappable fully homomorphic encryption. In *Annual International Symposium on Computer Architecture*, 2022.

[38] Sehoon Kim, Amir Gholami, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. I-bert: Integer-only bert quantization. In *International conference on machine learning*. PMLR, 2021.

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

[40] Donald E Knuth. Evaluation of polynomials by computer. *Communications of the ACM*, 1962.

[41] Nishant Kumar, Mayank Rathee, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. Cryptflow: Secure tensorflow inference. In *S&P*, 2020.

[42] Ryan Lehmkuhl, Pratyush Mishra, Akshayaram Srinivasan, and Raluca Ada Popa. Muse: Secure inference resilient to malicious clients. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 2201–2218, 2021.

[43] Baiyu Li, Daniele Micciancio, Mark Schultz, and Jessica Sorrell. Securing approximate homomorphic encryption using differential privacy. In *Annual International Cryptology Conference*, pages 560–589. Springer, 2022.

[44] Dacheng Li, Rulin Shao, Hongyi Wang, Han Guo, Eric P Xing, and Hao Zhang. MPCFormer: fast, performant and private transformer inference with MPC. In *International Conference on Learning Representations (ICLR)*, 2023.

[45] Natasha Lomas. Italy orders ChatGPT blocked citing data protection concerns. https://techcrunch.com/2023/03/31/chatgpt-blocked-italy/. Last accessed: 05/28/2023.

[46] Brady D Lund and Ting Wang. Chatting about ChatGPT: how may AI and GPT impact academia and libraries? *Library Hi Tech News*, 2023.

[47] Cecily Mauran. Whoops, Samsung workers accidentally leaked trade secrets via ChatGPT. https://mashable.com/article/samsung-chatgpt-leak-details. Last accessed: 05/28/2023.

[48] Daniele Micciancio and Michael Walter. On the bit security of cryptographic primitives. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 3–28. Springer, 2018.

[49] Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. Delphi: a cryptographic inference service for neural networks. In *USENIX Security*, 2020.

[50] Payman Mohassel and Yupeng Zhang. SecureML: a system for scalable privacy-preserving machine learning. In *S&P*, 2017.

[51] Theodore Samuel Motzkin. Evaluation of polynomials and evaluation of rational functions. *Bulletin of the American Mathematical Society*, 1955.

[52] OpenAI. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[53] Alexander M Ostrowski. On two problems in abstract algebra connected with horner's rule. In *Studies in Mathematics and Mechanics presented to Richard von Mises*, 1954.

[54] V Ya Pan. Methods of computing values of polynomials. *Russian Mathematical Surveys*, 1966.

[55] Kate Park. Samsung bans use of generative AI tools like ChatGPT after April internal data leak. https://techcrunch.com/2023/05/02/samsung-bans-use-of-generative-ai-tools-like-chatgpt-after-april-internal-data-leak. Last accessed: 05/28/2023.

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

[57] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[58] Deevashwer Rathee, Anwesh Bhattacharya, Rahul Sharma, Divya Gupta, Nishanth Chandran, and Aseem Rastogi. Secfloat: Accurate floating-point meets secure 2-party computation. In *S&P*, 2022.

[59] 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 *S&P*, 2021.

[60] Deevashwer Rathee, Mayank Rathee, Nishant Kumar, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. Cryptflow2: Practical 2-party secure inference. In *CCS*, 2020.

[61] Evgeny Yakovlevich Remez. Sur une classe de développement des fonctions en séries trigonométriques. *Comptes Rendus (Doklady) de l'Académie des Sciences de l'URSS*, 1934.

[62] Malik Sallam. ChatGPT utility in healthcare education, research, and practice: systematic review on the promising perspectives and valid concerns. In *Healthcare*, 2023.

[63] Nikola Samardzic, Axel Feldmann, Aleksandar Krastev, Srinivas Devadas, Ronald Dreslinski, Christopher Peikert, and Daniel Sanchez. F1: A fast and programmable accelerator for fully homomorphic encryption. In *Annual IEEE/ACM International Symposium on Microarchitecture*, 2021.

[64] Nikola Samardzic, Axel Feldmann, Aleksandar Krastev, Nathan Manohar, Nicholas Genise, Srinivas Devadas, Karim Eldefrawy, Chris Peikert, and Daniel Sanchez. Craterlake: a hardware accelerator for efficient unbounded computation on encrypted data. In *Annual International Symposium on Computer Architecture*, 2022.

[65] Microsoft SEAL (release 4.1). https://github.com/Microsoft/SEAL, January 2023. Microsoft Research, Redmond, WA.

[66] Seyedmostafa Sheikhalishahi, Riccardo Miotto, Joel T Dudley, Alberto Lavelli, Fabio Rinaldi, Venet Osmani, et al. Natural language processing of clinical notes on chronic diseases: systematic review. *JMIR medical informatics*, 2019.

[67] Gilbert W Stewart. *Afternotes on numerical analysis*. SIAM, 1996.

[68] Sijun Tan, Brian Knott, Yuan Tian, and David J Wu. CryptGPU: Fast privacy-preserving machine learning on the GPU. In *S&P*, 2021.

[69] Ahmed Tlili, Boulus Shehata, Michael Agyemang Adarkwah, Aras Bozkurt, Daniel T Hickey, Ronghuai Huang, and Brighter Agyemang. What if the devil is my guardian angel: ChatGPT as a case study of using chatbots in education. *Smart Learning Environments*, 2023.

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

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

[72] 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 *International Conference on Learning Representations*, 2018.

[73] Ling Wang, Joseph Needham, and Wang Ling. Horner's method in chinese mathematics: Its origins in the root-extraction procedures of the han dynasty. *T'oung Pao*, 1954.

[74] Andrew C Yao. Protocols for secure computations. In *Annual Symposium on Foundations of Computer Science (SFCS)*, 1982.

[75] Zhewei Yao, Zhen Dong, Zhangcheng Zheng, Amir Gholami, Jiali Yu, Eric Tan, Leyuan Wang, Qijing Huang, Yida Wang, Michael Mahoney, et al. Hawq-v3: Dyadic neural network quantization. In *International Conference on Machine Learning*, pages 11875–11886. PMLR, 2021.

[76] Ofir Zafrir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat. Q8bert: Quantized 8bit bert. In *2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing-NeurIPS Edition (EMC2-NIPS)*, pages 36–39. IEEE, 2019.

[77] Wenting Zheng, Ryan Deng, Weikeng Chen, Raluca Ada Popa, Aurojit Panda, and Ion Stoica. Cerebro: A platform for multi-party cryptographic collaborative learning. In *USENIX Security*, 2021.

[78] Wenting Zheng, Raluca Ada Popa, Joseph E Gonzalez, and Ion Stoica. Helen: Maliciously secure coopetitive learning for linear models. In *S&P*, 2019.

Appendix A. Gazelle & Iron Methods for MatMul

In this section, we present the methods of Gazelle and Iron for computing ciphertext-plaintext matrix multiplications.

Gazelle [35] packs multiple copies of the same encrypted input vector in a ciphertext, sends this to the server, and arranges the plain weight matrix in a diagonal format in order to compute dot products efficiently. Unfortunately, extending this to the transformer architecture will produce inefficient packing for three reasons. First, the multiplication input (i.e., $\boxed{\mathbf{A}}$ in our example) is a matrix instead of a vector, which could be realized with Gazelle's technique by encoding each matrix row as one vector-ciphertext. However, this increases the number of expensive HE multiplications to $m \times d_1$. This is significantly more than actually needed. Second, Gazelle has to rotate (a) the encrypted rows of $\boxed{\mathbf{A}}$, or (b) the multiplication products before summing them up to match corresponding ciphertext slots. Third, Gazelle pads the vector's size to the next power of 2 to enable correct summations after rotations. Taking BERT-base's parameters as an example, this essentially blows up the matrix size to $2^{10} = 1024$ to accommodate $d = 768$ model dimensions, resulting in 25% "wasted" ciphertext slots.

Iron transfers the idea of Cheetah [32] to matrix-matrix multiplications. They leverage polynomial coefficient packing and encode the values in the polynomial coefficients. Iron arranges packing indices of the two matrices in a smart way, such that the multiplication result polynomial's coefficients contain the inner product results of the subvectors of the matrices. In this way, Iron gets rid of the expensive rotations in HE. However, the result packings are very sparse, and most of the coefficients are useless, which causes great communication overhead.

Appendix B. Motzkin's Polynomial Pre-Processing

In this section, we present the detailed Motzkin's polynomial pre-processing procedure.

Theorem 1 (Motzkin's polynomial pre-processing). Let f be a polynomial of degree n:

$$f(x) = x^n + a_{n-1}x^{n-1} + \cdots + a_0. \tag{10}$$

$t, \alpha_i, \beta_i, i \in [n]$ are constant parameters. Then f can be transformed into a polynomial P. The evaluation of this new polynomial P requires evaluating the following sequence of polynomials [40]:

$$\begin{aligned} &y = x + t,\ w = y^2, \\ &P_0 = \begin{cases} w + y + \beta_0 & \text{if } n \mod 2 = 0 \\ y + \beta_0 & \text{if } n \mod 2 = 1, \end{cases} \\ &P_1 = P_0 \cdot [(w - \alpha_1) + \beta_1], \\ &P_2 = P_1 \cdot [(w - \alpha_2) + \beta_2], \\ &\text{etc.} \end{aligned} \tag{11}$$

The choices in the later steps depend on whether the reduction equation is solvable or not. This gives us $n-r-1$ multiplications, where r is the number of reduction equations we are able to solve. Approximately, we need $\lceil n/2 \rceil$ multiplications and $n+1$ additions. Using the above procedure, degree-4 polynomials need 2 multiplications, while degree-5 only need 3 [40].

Appendix C. Approximation Details

In this section, we give the exact parameters for our approximations presented in §5.2 and §5.3.

GELU:

$a = 0.020848611754127593$,
$b = -0.18352506127082727$,
$c = 0.5410550166368381$,
$d = -0.03798164612714154$, and
$e = 0.001620808531841547$.

Pre-processed GELU:

$g_0 = 0.14439048359960427$,
$g_1 = -0.7077117131613893$,
$g_2 = 4.5702822654246535$,
$g_3 = -8.15444702051307$.
$g_4 = 16.382265425072532$.

Tanh:

$a = -0.013232131886235352$,
$b = 0.09948747962825866$,
$c = -0.20093640347818847$,
$d = -0.17616532856475706$,
$e = 1.0542492677156243$,
$f = -0.0024920889620412097$.

Pre-processed Tanh:

$t_0 = -4.259314087994767$,
$t_1 = 18.86353816972803$,
$t_2 = -36.42402897526823$,
$t_3 = -0.013232131886235352$,
$t_4 = -3.3289339650097993$,
$t_5 = -0.0024920889620412097$.

Appendix D. Detailed Algorithm of Word Elimination

In this section, we present our detailed algorithm of word elimination in Alg. 2.

Recall that in §6.1, we sort the score vector $\mathsf{s} \in \mathsf{R}^m$, which forms key-value pairs with the input token sequence. This allows us to rank tokens by their scores and discard those contributing minimally.

However, we need to maintain the relative order of remaining tokens while eliminating those with smaller contributions. To address this, a comparison with the median will yield either m or 0. We then add token indices to these comparison results and get another vector $\boxed{\mathsf{C}}$ as indicated in line 6 of Alg. 2. In the second sorting round, we sort $\boxed{\mathsf{C}}$, ensuring that tokens scoring higher than the median are

Algorithm 2 Oblivious Word Elimination

function WORDELIMINATE($\boxed{\mathsf{s}}$, $\boxed{\mathbf{X}}$, $\boxed{\mathbf{Att}}$)
  ▷ contribution scores, input matrix, attention result
  $\boxed{\mathsf{s}'} = \mathsf{BitonicSort}(\boxed{\mathsf{s}})$
  $\boxed{\mathsf{median}} = \boxed{\mathsf{s}'}_{m/2}$
  Create $\mathsf{v} \in \mathsf{Z}^m$ with $\mathsf{v}_i = i$
  $\boxed{\mathsf{c}} = \mathsf{Compare}(\boxed{\mathsf{s}}, \boxed{\mathsf{median}}) \times m + \mathsf{v}$
  $\boxed{\mathbf{X}'}$, $\boxed{\mathbf{Att}'}$ = $\mathsf{BitonicSortSwap}(\boxed{\mathsf{c}}, \boxed{\mathbf{X}}, \boxed{\mathbf{Att}})$
  return $\boxed{\mathbf{X}'}_{1:m/2}$, $\boxed{\mathbf{Att}'}_{1:m/2}$

always chosen and their relative order is preserved by their index sequence.

Given that bitonic sorting is a classic sorting algorithm, we omit the details of its implementation. Note that the function BitonicSort in Alg. 2 returns the sorted vector $\boxed{\mathsf{s}'}$, and the function BitonicSortSwap will sort $\boxed{c}$ and obviously swap the rows of $\boxed{\mathbf{X}}$ and $\boxed{\mathbf{Att}}$ together with the elements in $\boxed{c}$ during sorting.

Appendix E. Scales of Linear Layers

In this section, we present the scales of the different linear layers in secure computation-aware fine-tuning. The detailed scales are shown in Tab. 4.

Layer	Linear #	Input	Weights
0,1,3-11	1	5	6
2	1	5	5
0-11	2	6	6
0-11	3	5	6
0-8, 11	4	4	5
9,10	4	4	4

TABLE 4: Input and weights scale of each linear layer in each attention layer.

Appendix F. BSGS Rotation Savings for Different Matrix Dimensions in BERT

In this section, we present the detailed savings on rotations of our BSGS technique applied for ciphertext-plaintext matrix multiplications §4.1.2. The concrete savings are shown in Tab. 5

	Linear 1	Linear 2	Linear 3	Linear 4
BOLT w/ BSGS	288	168	324	324
BOLT w/o BSGS	756	756	756	3024

TABLE 5: BSGS rotation savings for different linear layers of BERT.

Consistent with our findings in §4.1.2, we save more rotations when the RHS plaintext matrix is tall and skinny as shown in Linear 4 of Tab. 5, where the size of the RHS matrix is 3072×768. Overall, BSGS can significantly save rotations numbers, with a factor of $2.33\times$ to $9.33\times$.

Appendix G. Clarification on Iron's LayerNorm Optimization

Iron's LayerNorm optimization is inaccurate and will introduce large errors to the inference and make the model's performance close to random guessing. LayerNorm is defined as

$$\mathsf{LayerNorm}(\mathsf{X})_{i,j} = \frac{{}_j(\mathsf{X}_{i,j} - \quad_i)}{\quad_i} + \quad_j.$$

Iron proposes to combine the weights of the LayerNorm ($\frac{\gamma_j}{\sigma_i}$ and β_j) with the next linear layer's weight to save one matrix multiplication. Now the output of the LayerNorm is

$$\mathsf{LayerNorm}'(\mathsf{X})_{i,j} = \mathsf{X}_{i,j} - \quad_i.$$

However, Iron does not account for the residual connections in transformers – the output of LayerNorm is added to subsequent layers. Given that the optimized LayerNorm does not directly produce accurate outputs, the residual connections will lead to significant errors, making the model's performance close to random guessing. We have confirmed this flawed optimization with Iron's authors.

Appendix H.
Meta-Review

The following meta-review was prepared by the program committee for the 2024 IEEE Symposium on Security and Privacy (S&P) as part of the review process as detailed in the call for papers.

H.1. Summary

The reviewers appreciated the various cryptographic improvements and system optimizations along with the efficient approximations proposed by the authors. Overall, the paper focuses on a timely research topic and the evaluation results demonstrate significant improvement over the prior state of the art.

H.2. Scientific Contributions

- Provides a valuable step forward in an established field.

H.3. Reasons for Acceptance

- The paper focuses on an important problem of improving the efficiency of secure inference.

H.4. Noteworthy Concerns

- The paper does not include a formal security proof of the BOLT protocol, besides the fact that it consists of secure building blocks.