Characterization of MPC-based Private Inference for Transformer-based Models

Yongqin Wang*
University of Southern California
yongqin@usc.edu

G. Edward Suh
Meta AI
edsuh@fb.com

Wenjie Xiong
Meta AI
wenjiex@fb.com

Benjamin Lefaudeux
Meta AI
lefaudeux@fb.com

Brian Knott
Meta AI
brianknott@fb.com

Murali Annavaram
University of Southern California
annavara@usc.edu

Hsien-Hsin S. Lee
Meta AI
leehs@fb.com

***Abstract*—In this work, we provide an in-depth characterization study of the performance overhead for running Transformer models with secure multi-party computation (MPC). MPC is a cryptographic framework for protecting both the model and input data privacy in the presence of untrusted compute nodes. Our characterization study shows that Transformers introduce several performance challenges for MPC-based private machine learning inference. First, Transformers rely extensively on "softmax" functions. While softmax functions are relatively cheap in a non-private execution, softmax dominates the MPC inference runtime, consuming up to 50% of the total inference runtime. Further investigation shows that computing the maximum, needed for providing numerical stability to softmax, is a key culprit for the increase in latency. Second, MPC relies on approximating non-linear functions that are part of the softmax computations, and the narrow dynamic ranges make optimizing softmax while maintaining accuracy quite difficult. Finally, unlike CNNs, Transformer-based NLP models use large embedding tables to convert input words into embedding vectors. Accesses to these embedding tables can disclose inputs; hence, additional obfuscation for embedding access patterns is required for guaranteeing the input privacy. One approach to hide address accesses is to convert an embedding table lookup into a matrix multiplication. However, this naive approach increases MPC inference runtime significantly. We then apply tensor-train (TT) decomposition, a lossy compression technique for representing embedding tables, and evaluate its performance on embedding lookups. We show the trade-off between performance improvements and the corresponding impact on model accuracy using detailed experiments.**

***Index Terms*—MPC, Transformer, BERT, XLM, secret sharing, binary share, arithmetic share**

I. Introduction

Given the resource management benefits such as elasticity, availability, and cost-effectiveness offered by cloud service providers, a growing number of machine learning workloads are migrated to the cloud for operations. Under this modern compute paradigm, confidential data and models can be leaked to unwanted parties if the service providers are curious, malicious, or compromised. Transformer-based models are commonly used for natural language processing (NLP) today. But their usage has also broadened to other application domains such as computer vision [6]. These models take natural languages, images, or videos as inputs for inference. However, these inputs may contain sensitive private information about the users and require rigorous protection. In other words, data privacy has become a pressing concern for transformer-based machine learning models deployed in the cloud.

To address the security and privacy challenge, secure computation techniques such as homomorphic encryption (HE) [24], trusted execution environments (TEE) [9], [13], [18], [21], [27], [28], and secure multi-party computation (MPC) [8] have been applied to support privacy-preserving machine learning (PPML). TEEs use secure hardware to protect against malicious OS and physical attacks. However, TEEs assume the hardware vendor is trusted and the hardware implementation is bug-free, whereas MPC and HE are based on modern cryptography with strong mathematical security guarantees. In addition, MPC protocols can still provide strong security even when a subset of parties are compromised [8]. In this paper, we focus on using MPC to protect transformer-based models. HE, which also provides defense against a strong threat model, incurs significantly higher performance overhead, and is outside the scope of this paper's characterization focus.

This paper provides the first in-depth study on MPC-based private inference of Transformer-based models. Although there are previous studies on MPC-based private inference, they focused primarily on convolution neural networks (CNNs) [3], [7], [12], [15], [30]. In our characterization study, we found that Transformer-based models introduce new performance challenges to MPC:

1) Overhead of softmax.
2) Overhead of securing embedding table accesses.
3) Limited dynamic ranges of approximated complex non-linear functions.

A. Softmax Runtime Overhead

While non-linear activation functions such as ReLU are known to be a major source of performance overhead for running CNNs in MPC, softmax accounts for an even larger

*This work was performed during the period when Yongqin Wang was employed as a research intern at Meta AI.

portion of the MPC execution time for Transformers. Transformers use softmax in each layer. Hence, the performance penalty in executing softmax function in MPC gets amplified due to the repeated use of this function.

Further investigation showed that the softmax function relies on computing the max for achieving numerical stability. However, computing the max in MPC setting is the primary source for this overhead.

B. Secure Accesses to Embedding Tables

Unlike CNNs whose main computation components are activation functions, convolutions and fully-connected operations, Transformed-based NLP models use embedding tables to convert word IDs (a type of categorical data) into a more meaningful representation format. Categorical data are typically sparse and are difficult for machine learning systems to handle because semantically similar items, in most cases, are not close in the form of sparse vectors. During training, embedding tables will learn to map semantically similar inputs closer in the embedding space. An embedding table lookup is defined as

$$R = A \times W \tag{1}$$

where W is the embedding table and $A = [e_1, e_2, ..., e_t]$. Each e_i is a one-hot vector corresponding to the embedding table entry being accessed. Embedding tables can be as large as 1GB for NLP models, and embedding table look-ups are commonly implemented as row selection operations. However, accesses to embedding tables leak private information of the user, even if the embedding table data is encrypted. Hence, embedding table accesses must be made oblivious to input values in private inference. To implement embedding tables securely and privately in the MPC setting, one can obfuscate embedding table lookups by turning them into matrix multiplications. Our characterization study shows that replacing an embedding table lookup with a matrix multiplication significantly increases the execution time of embedding table operations.

One approach to reduce this cost is to decompose a large matrix into multiple smaller matrices, using a technique known as tensor train (TT) decomposition [11], [33]. However, TT decomposition is a lossy compression and hence using compressed tables to perform inference lookups may impact accuracy. We experimentally demonstrate the trade-off between inference runtime and model accuracy.

C. MPC dynamic range issue

Besides the model inference runtime, we also identified a key challenge to MPC inference of Transformer-based models, that is, the approximated complex functions lead to narrow dynamic ranges, which can affect model accuracy. While ReLU, commonly used in CNNs, can be precisely evaluated in the standard MPC protocols with reasonable overhead, more complex non-linear functions are often approximated as a polynomial function. The accuracy of this approximation is acceptable when the inputs fall within a specified range, often referred to as "dynamic range". Inputs outside the dynamic range cause substantial numerical error which in turn hurt model accuracy. Our study sheds light on this hurdle and informs the practitioners to pay close attention to the dynamic range of inputs to the approximated polynomial which is used as a substitute for the non-linear operation in MPC. When the dynamic range of the approximated non-linear functions is expected to be narrow, using MPC may be impractical since approximating such functions leads to substantial accuracy loss. Besides impacts on the model accuracy, narrow dynamic ranges also make optimizations to softmax very challenging due to the small dynamic ranges of reciprocal and exponential functions.

The rest of the paper is organized as follows: Section II provides necessary background information about both MPC and Transformer-based models. Those are key bedrocks of this paper. Without understanding those concepts, one will find this paper difficult to understand. Section III presents our characterization study results for private inference of Transformer-based models and discusses the key findings we listed above.

II. Backgrounds

A. Secure Multi-Party Computation

This section provides a brief overview of how MPC protocols compute some of the common operations in Transformer models. The MPC protocol allows a client to distribute its operands (a vector, a matrix, etc.) as secret shares among "multiple untrusted parties" (MPC servers) such that each share does not leak any information about the original operands. After receiving their respective secret share, each MPC server can only see and manipulate its own secret share. When local MPC computations are completed, the MPC servers will return the results to the client. The client then combines the results from multiple MPC servers to decode the final plaintext result (using operations such as summation or XORing). Figure 1 shows an example of a 2-PC, where a client wishes to compute $Y = W \times X$, where $W \in \mathbb{R}^{2\times 2}$ and $X \in \mathbb{R}^2$. The client distributes the respective secret share of operands W and X to the two MPC servers, and retrieves Y after the MPC servers finish the operations of their local shares. We adopt semi-honest **threat model** for MPC in our work, where the MPC servers will exactly follow the predetermined protocols, but curious about the data MPC clients provided to them. For simplicity of explanation, assume MPC parties to be non-colluding, although all MPC protocols can tolerate a predefined amount of collusion.

1) Secret sharing formats: There are two major secret sharing formats to share an operand x in the MPC: **1) additive sharing** $[x_i]$**, 2) binary sharing** $\langle x_i \rangle$. For example, $[x_1] = r$ and $[x_2] = x - r$ can be x's two additive secret shares where $x = [x_1] + [x_2]$, and r is sampled from a uniform random variable. Practically, x and r are commonly implemented using **fixed point** because x and r can be in the same integer ring achieving perfect secrecy. On the other hand, $\langle x_1 \rangle = r$ and $\langle x_2 \rangle = x \oplus r$ can be x's two binary secret shares such that $x = \langle x_1 \rangle \oplus \langle x_2 \rangle$, and r is also sampled from a uniform

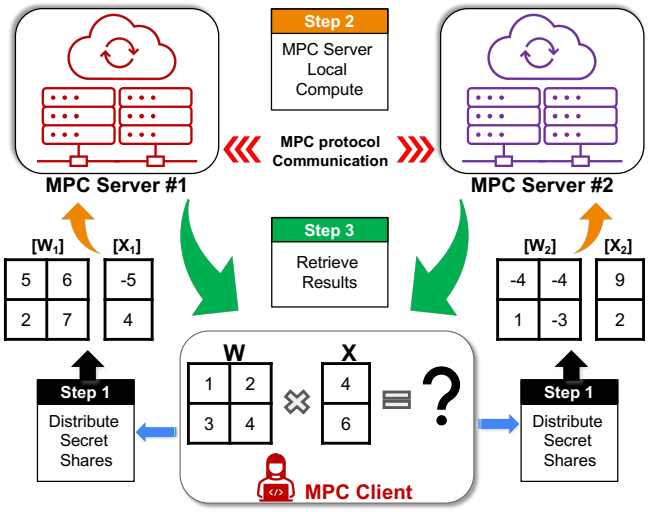


Fig. 1: 2-PC system example.

random variable. Generally speaking, additive shares are used for additions and multiplications; binary shares are used for bit-wise operations. In Figure 1, both the matrix W and the vector X are in additive sharing format.

2) MPC Multiplications: For MPC servers to compute their share of z ($[z]$ s.t. $z = x \times y$), a Beaver triple $([a], [b], [c])$ s.t. $c = a \times b$ can facilitate the multiplication. The Beaver triples are distributed among MPC servers during an off-line phase. Algorithm 1 shows the detailed algorithm to compute MPC multiplications.

Algorithm 1 Beaver Triple Assisted MPC Multiplication

Input: n MPC servers;
each party has $[x_i]$, $[y_i]$, $[a_i]$, $[b_i]$ and $[c_i]$ s.t. $c = a \cdot b$
for $i = 1$ **to** n **do**
  computes $[x_i] - [a_i]$ and $[y_i] - [b_i]$
  broadcast local $[x_i] - [a_i]$ and $[y_i] - [b_i]$
  wait until other $[x_i] - [a_i]$ and $[y_i] - [b_i]$ has been received
  computes $x - a = \sum_{i=1}^{n} [x_i] - [a_i]$
  computes $y - b = \sum_{i=1}^{n} [y_i] - [b_i]$
end for
for $i = 2$ **to** n **do**
  computes $[z_i] = [c_i] + (x - a)[b_i] + (y - b)[a_i]$
end for
Party # 1 computes $[z_1] = [c_1] + (x - a)[b_1] + (y - b)[a_1] + (x - a)(y - b)$

MPC client can retrieve $z = xy$ by summing the $[z_i]$ from all MPC servers.

$$\begin{aligned}\sum_{i=1}^{n}[z_i] =& \sum_{i=1}^{n}[c_i] + (x-a)[b_i] + (y-b)[a_i] \\ &+ (x-a)(y-b) \\ =& c + xb - ab + ya - ba + xy \\ &- xb - ya + ab = xy = z \end{aligned} \tag{2}$$

3) MPC comparisons: To produce $[x_i < y_i]$ in MPC settings when x and y are in additive sharing format, MPC parties will first obtain $[d_i] = [x_i] - [y_i]$, and convert additive shares $[d_i]$ to binary share $\langle d_i \rangle$. Then, obtain the sign bit (shifting the sign bit to the LSB): $\langle s_i \rangle = sgn(\langle d_i \rangle)$. Finally, convert the sign bit $\langle s_i \rangle$ to an additive sharing format will produce $[x_i < y_i]$. Other comparison results can be derived from $x < y$. Comparisons are seemingly efficient. However, conversions between additive and binary shares require a round of communications which will add significant performance overhead. The detailed conversion algorithms are presented below.

Algorithm 2 Additive Share to Binary Share Conversion

Input: n MPC servers; each party has $[x_i]$
for $i = 1$ **to** n **do**
  generate binary shares $\langle [x_i] \rangle_j : [x_i] = \bigoplus_{j=1}^{n} \langle [x_i] \rangle_j$
  send $\langle [x_i] \rangle_j$ to party j
  wait until all $\langle [x_j] \rangle_i$ are received
  compute $\langle x_i \rangle = \langle \sum_{j=1}^{n} [x_j] \rangle_i$
  Note: that one can only use bit-wise operations to compute summation of binary shares (eg: using Ripple-carry adder logics).
end for

Algorithm 3 Binary Share to Additive Share Conversion

Input: n MPC servers;
each party has $\langle x_i \rangle$, $[r_i], \langle r_i \rangle$, where r is a random mask; $r^{(b)}$ and $x^{(b)}$ are the b_{th} bit of r and x
for $i = 1$ **to** n **do**
  compute $\langle z_i \rangle = \langle x_i \rangle \oplus \langle r_i \rangle$
  broadcast $\langle z_i \rangle$ to retrieve $z = x \oplus r$
  compute $[\langle x_i \rangle^{(b)}] = [r_i^{(b)}] + z^{(b)} - 2[r_i^{(b)}] z^{(b)}$
  compute $[x_i] = \sum_b 2^b [\langle x_i \rangle^{(b)}]$
end for

4) MPC ReLU: ReLU in plaintext is defined as $ReLU(x) = x \times (x > 0)$. However, in MPC protocols, the comparison operations use the algorithm defined in Section II-A3 requiring two rounds of communication. Thus, ReLU, an inexpensive operation in the plaintext, can be very expensive in the MPC protocols due to communication latency.

5) Approximated MPC operations: MPC protocols allow us to retrieve exact additions, multiplication, comparisons and ReLU results. However, some non-linear operations need to be approximated using additions and multiplications.

MPC Exponential: Exponential functions are commonly used for softmax. In MPC settings, the exponential function is generally implemented as

$$e^x = \lim_{n \to \infty} (1 + \frac{x}{2^n})^{2^n} \tag{3}$$

where n is the total number of approximation iterations. In our implementation, we choose n to be 8. Polynomial approximations such as Taylor Series are not used because exponential functions grow much faster than polynomials [14].

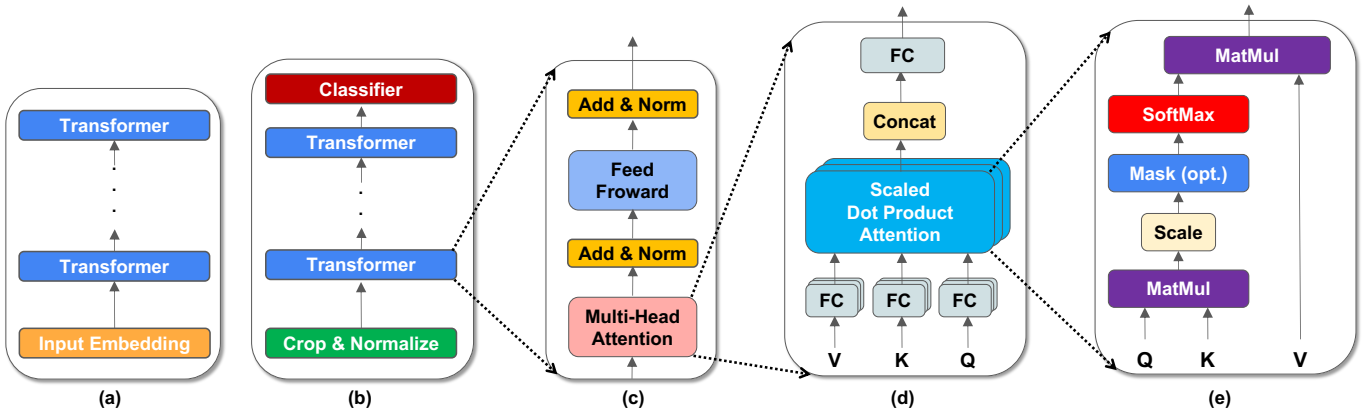


Fig. 2: Transformer-based model computation graph decomposition: (a) General structure of XLM; (b) General structure of a ViT; (c) A Transformer; (d) Multi-headed attention; (e) Scaled dot-product attention.

MPC Reciprocal: Finding the reciprocal is important for division in the MPC protocols. One can use reciprocal+multiplication to compute division operations. Like exponential functions, reciprocals are also approximated. Generally, in MPC protocols, the Newton-Raphson iteration is used to approximate reciprocal:

$$\frac{1}{x} = \lim_{n\to\infty} y_n = y_{n-1}(2 - xy_{n-1}) \tag{4}$$

where $y_0(x) = 3e^{0.5-x} + 0.003$ which makes the approximation work for a large input domain [14].

MPC GELU: GELU [10] is a novel activation function based on Gaussian error functions. GELU in plaintext is defined as:

$$GELU(x) = x \cdot \frac{1}{2}[1 + erf(\frac{x}{\sqrt{2}})] \tag{5}$$

where x is standard Gaussian Distribution, and erf is the Gaussian error function. The error function is defined as:

$$erf(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt \tag{6}$$

In MPC, erf can be approximated using Taylor approximation or $tanh$ functions.

B. Transformer-based Models

Transformers can be used to encode dependence of input tokens [32]. Transformers are gaining popularity among NLP and image processing. Modern language pre-processors such as BERT [5], Roberta [19], XLM [16], and XLM-R [4] are Transformer-based. These models consist of an input embedding layer followed by multiple layers of Transformers [29]. ViT [6], an emerging vision model, consists of multiple layers of transformers and a fully connected classifier. Different models' input processing layers are different, but their Transformer layers have the same structure. Figure 2 shows the computation decomposition of two Transformer-based models (XLM and ViT). Figure 2(c) shows a computation graph of a Transformer encoder. There are two major computation blocks: multi-head attention and feed forward. Figure 2(d) shows the detailed computation breakdown of a multi-head attention layer, and the feed forward layer in Figure 2(c) is defined as:

$$FFN(x) = Linear\{Act[Linear(x)]\} \tag{7}$$

where "Act" can be ReLU or GeLU. Our characterization study found that the softmax function inside the scaled dot-product attention is the main performance bottleneck of MPC-based private inference.

C. Other Transformer Architectures

Figure 2(e) shows that the complexity of self-attention is quadratic with the input size due to the matrix multiplications among Q, K, and V. The complexity of the input dimension to the softmax function is also quadratic. As we show later in our characterization data, the cost of softmax is significant in MPC setting. Hence, Transformers' quadratic complexity restricts applications using long sentences, even for plaintext. And in an MPC setting, these costs will only get worse. Several proposals try to reduce the Transformer complexity. In this paper, we mainly look into Linformer [31] and Nystromformer [32] as two alternative Transformer implementations that are likely to be more amenable in an MPC setting. These novel transformers modify structures of multi-head attention layers to reduce softmax complexity.

Linformer introduces extra projection operations prior to the scaled dot-product attention for V and K inputs. A projection operator is essentially a rectangular linear layer that projects long V and K into shorter sequences, thus reducing the cost of $Softmax(QK^T)$.

Nystromformer uses the well-known Nystrom method [1] to approximate the Transformer attention using $\mathcal{O}(n)$ complexity algorithm. The scaled dot-product layer in Figure 2(d) can be written as

$$Attention(V, K, Q) = Softmax(\frac{QK^T}{s})V \tag{8}$$

where s is a constant scale factor. If the sequence length of Q and K is N, the sequence length of QK^T will be N^2. Nystromformer uses Nystrom algorithm to approximate the original attention layer:

$$NystromAttention(V,K,Q) = Softmax(\frac{Q\tilde{K}^T}{s})\ Z^* Softmax(\frac{\tilde{Q}K^T}{s}) \quad (9)$$

where $Z^* = Softmax(\frac{\tilde{Q}\tilde{K}^T}{s})^+$, and $\tilde{K} = AvgPooling(K)$. Nystromformer approximates the full attention matrix computation by focusing on a selection of landmarks within the inputs. These landmarks first lead to $\tilde{K}$ and $\tilde{Q}$ approximations of K and Q, which are then used to compute $Softmax(Q\tilde{K}^T)$, $Softmax(\tilde{Q}K^T)$ and $Softmax(\tilde{Q}\tilde{K}^T)$, whose dimensions are greatly reduced when compared to the original scaled dot product attention. The matrix product of these intermediates provides the final attention matrix, with a complexity of $\mathcal{O}(n)$ while maintaining a good approximation of the original softmax function.

D. Tensor-Train (TT) Decomposition.

The basic idea of TT decomposition is to represent a big matrix using tensor products of several smaller matrices.

$$\mathbb{R}^{M_1\times M_2\ldots\times M_k} \bigotimes \mathbb{R}^{N_1\times N_2\ldots\times N_k} \rightarrow \mathbb{R}^{M_1\cdot N_1\times M_2\cdot N_2\ldots\times M_k\cdot N_k} \quad (10)$$

Generalizing TT decomposition to an embedding matrix $W \in \mathbb{R}^{M\times N}$, W can be decomposed into d smaller matrices $w_k \in \mathbb{R}^{R_{k-1}\times m_k\times n_k\times R_k}$, where $M = \prod_{k=1}^{d} m_k$, $N = \prod_{k=1}^{d} n_k$, and $R_0 = R_d = 1$. We refer R_k as the ranks of decomposed matrices. For example, if d is 2, an embedding size of $[250002\times 1024]$ can be decomposed into two smaller matrices of $[1\times 500\times 32\times rank]$ and $[rank\times 502\times 32\times 1]$. Note that $502\times 500 > 250002$ and $32\times 32 = 1024$. If the rank of the all decomposed matrices is 4, the original matrix with 24M parameters can be decomposed into two smaller 64K-element matrices. When accessing certain locations in the original embedding table, an entry in every decomposed matrix is fetched (this fetching is implemented as dense one-hot matrix multiplications). Algorithm 4 shows the algorithm that reconstructs an embedding table entry from decomposed matrices. One_hot function in Algorithm 4 converts an index into a one-hot vector. Results from one-hot index can be supplied by an MPC client. TT decomposition can compress embedding tables significantly. However, the TT decomposition introduces additional computations to reconstruct the embedding (the second for loop in Algorithm 4). Later sections demonstrate the trade-off between the model accuracy and the inference runtime.

Algorithm 4 TT Reconstruction

Input: an index e,
d decomposed matrices $w_k \in \mathbb{R}^{R_{k-1}\times m_k\times n_k\times R_k}$
fetchedLines = [ ]
for $i = 1$ **to** d **do**
  line = one_hot($e \bmod m_k$)
  fetchedLines.append(line$\times w_k$)
end for
res =fetchedLines[0]
for $i = 1$ **to** $d-1$ **do**
  res = res $\times$fetchedLines[i]
end for
Return res

III. Transformer-based Model MPC Inference

In this section, we present the MPC inference characterization results and analyses. In Section III-B, we first present the runtime analysis for XLM [16] (a state-of-the-art NLP model) and ViT (vision model using layers of Transformer) [6]. The runtime analysis identities the key components of MPC inference. In Section III-C, we discuss a numerical challenge when using Transformer-based models with MPC protocols. Finally, in Section III-D and Section III-E, we present an in-depth analysis of softmax and embedding table accesses and key challenges in optimizing their runtime.

A. Experimental Setup

We implemented MPC-based execution of several Transformer models using CrypTen MPC framework [14]. Each MPC server has an NVIDIA Tesla V100 Volta GPU, and they are connected by a 100Gb/s Ethernet cable. CrypTen employs 64-bit fixed-point numbers and uses 16 bits as the precision bits [14]. CrypTen supports additive and binary secret sharing formats to enable a wide range of operators implemented in MPC. CrypTen also implements more functions using generic MPC protocols than other known frameworks [15], [30]. Given the optimized implementation and at-scale usage of CrypTen in the MPC systems, we believe that the data we collected and characterized in this paper are a reflection of MPC's algorithmic limitations, not CrypTen-specific observations.

B. Performance Breakdowns

Table I breaks down the MPC inference runtime for XLM (left) and ViT (right) into multiple components. "ReLU" is the activation function inside the feed forward layer. The execution times of all fully-connected layers and matrix multiplications inside the Transformer are listed under "MatMul". "Embedding" is the embedding table access runtime. The first column shows the model inference runtime in plaintext 32-bit floating point divided into the core operations performed in each model; the second column shows the time spent for additional communications introduced by the MPC protocols; the third column shows the total MPC inference runtime for each operation type; the fourth column shows each operation's runtime percentage. For both XLM and ViT, softmax in the MPC setting makes up about 50% of the total inference runtime, while in plaintext the execution time of softmax is very small. Note that the XLM model uses an embedding table lookup to convert words into a dense representation.

TABLE I: Transformer-based model MPC runtime decomposition in seconds.

	12-layer XLM				12-layer ViT			
	Plaintext	MPC Comm	MPC Total	MPC breakdown %	Plaintext	MPC Comm	MPC Total	MPC breakdown %
Embedding	1.75E-5	3.72	11.81	20.37%	N/A	N/A	N/A	N/A
Norm	1.30E-3	0.3	0.43	0.74%	8.03E-4	0.22	0.35	1.23%
MatMul	4.24	1.88	8.22	14.17%	1.08	1.36	5.06	17.99%
ReLU	6.15E-4	8.28	8.59	14.82%	3.76E-4	5.76	5.99	21.29%
Softmax	6.61E-4	25.46	28.93	49.90%	3.77E-4	13.69	16.74	59.49%
Total	4.25	39.63	57.98	100%	1.09	21.03	28.14	100%

The embedding table size is $[250002 \times 1024]$. MPC secure embedding table lookups also make up a significant portion of MPC total inference runtime. However, ViT which consists of 12 Transformer-only layers does not use embedding table and hence there is no runtime associated with the embedding lookup in ViT.

1) MPC vs. Plaintext Inference: Comparing with their plaintext FP32 counterparts, all operations become slower with MPC. For the plaintext inference, majority of the inference runtime is spent on computing linear operations. ReLU, embedding table accesses, and softmax become extremely expensive in MPC. Softmax and ReLU become more time-consuming due to the cost of communication imposed by MPC protocols. In the background section, we described that to perform comparisons, two rounds of communications are needed. These additional communication rounds made ReLU and softmax quite expensive. Embedding table accesses are obfuscated using matrix multiplication operations, and hence embedding table lookups become time-consuming, and this aspect will be elaborated in Section III-E.

2) Sequence Length on Inference Runtime: Figure 3 shows the inference runtime of a single Transformer layer in the XLM model as a function of the input sequence length. We show how the runtime scales with different input sequence lengths using a single Transformer layer for clarity; similar trends can be observed for the entire XLM model's inference runtime as well. We show the runtime growth of MatMul (labeled as **Linear**), **ReLU**, and **Softmax** w.r.t the input length. We ignore the "Norm" computations because they account for a tiny fraction of the total computation time. Softmax runtime grows quadratically and dominates inference runtime since input dimensions to softmax function grow quadratically with input length. ReLU runtime grows only linearly because the input dimension to the ReLU function grow linearly.

3) Speedups when Model Weights are Public: When model parameters are non-sensitive or publicly available, model parameters can be distributed to MPC servers in plaintext while still allowing user input data to be private with MPC. In this setting, multiplications and additions will be the same as their plaintext counterparts. To compute $z = xy$, where y is public, each MPC party i only needs to compute one multiplication on its own secret share $[z_i] = [x_i]y$, and it is easy to see

$$\sum_i z_i = \sum_i [x_i]y = xy = z. \tag{11}$$

Multiplications with one public operand and one secret operand only involve computations on MPC servers' local shares and avoid broadcasting communications.

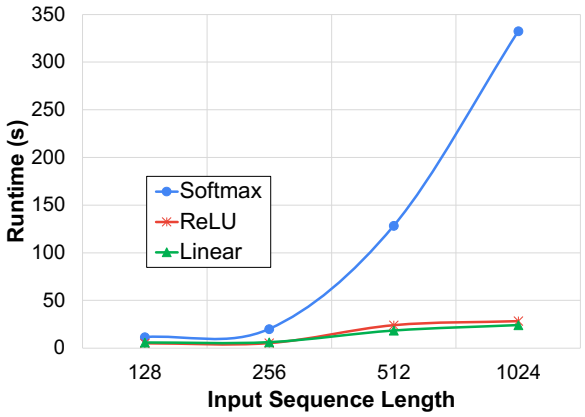


Fig. 3: GPU 2-PC Transformer inference runtime w/ different sequence lengths.

To understand how these observations impact inference runtime, we experimented with two setups: (1) a public model setting where the model is in plaintext but the user input is private, and (2) a private model setting where both the model and the input are private.

Figure 4 compares the MPC runtimes for a public model and a private model, assuming the model input is private. The models shown are VGG19 and a 12-layer Transformer from XLM. VGG19's major computation components are categorized into two classes: linear operations (convolution layers and dense layers), and non-linear operations (ReLU and Maxpooling). The Transformer model's major component can also be categorized as linear (matrix multiplications) and non-linear (Softmax, Activation, and Normalization). For VGG19, the runtime of non-linear operations (ReLU and Maxpooling) remain unchanged whether using a plaintext or private model for inference. However, the public model can cut the cost of linear operations significantly. Thus for CNN models such as VGG19, using a public model can reduce the overall inference runtime while still protecting the user input.

When using MPC on Transformer-based models, however, there is only a small performance improvement in linear operations even when a model is public. The reason is that transformers' linear operations are mostly matrix multiplications between user inputs (see Figure 2(d)), and softmax in the multi-head attention directly operates on a function of user inputs. All of their operands are still in the additive sharing

format, and making the model weights public does not reduce their inference runtime. With the major operator runtime unaffected by public weights, Transformer-based models show smaller speedups (6%) compared to CNN models when using public model as opposed to a private model.

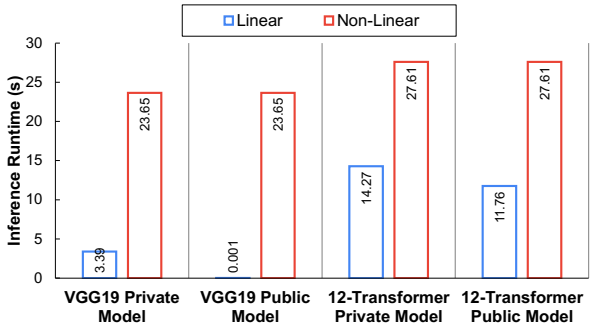


Fig. 4: MPC public model vs. private model.

4) Alternative Transformer Architectures for Reducing Non-linear MPC Overheads: The versatility of the original Transformer architecture [29], combined with a wide variety of use cases that exposed some limitations, gave birth over the last couple of years to a vibrant ecosystem. A taxonomy of Transformer architectures proposed by Tay et al. consists of 5 overlapping categories: low rank approximations, memory, non-learnable pattern, learnable pattern and recurrence [26]. Not all of these alterations are applicable to all problems, and choosing them will often imply some knowledge over the input distribution, or some prior training with a Transformer which exposes specific attention patterns.

In this study, we focus on Transformer models that are more suited for the MPC setting. Given that Softmax is a significant bottleneck, we selected two Transformer models, Linformer [31] and Nystromformer [32] from a modelling library [17]. Linformer and Nystromformer are part of the low rank approximations, and as such are best suited to cases where the input elements have a distribution which is inherently of a lower dimension than what their raw sequence would suggest. The Linformer puts projection operators prior to scaled dot-product attention to project K and V into shorter sequences, effectively reducing the input dimension to the softmax function. In our experiments, we choose 4 as the Linformer projection ratio. Nystromformer uses the well know Nystrom algorithm to construct a lower dimension approximation of the K and Q matrices, and decomposes the canonical attention matrix computation (which includes a softmax) into three smaller operations.

We can see from Figure 5, Linformer and Nystromformer's softmax runtime in 2-PC setting is reduced by 3.8x and 7x compared to the basic Transformer. For this experiment we used 4 times longer input sequences than what we used in evaluating the baseline Transformer model results shown in Table I. Hence, even with much longer inputs the reduction of softmax usage through reduced tensor dimensionality helps Linformer and Nystromformer. Although these new Transformers improve runtime, we observe that non-linear operations and softmax are still persistent bottlenecks (see Figure 5).

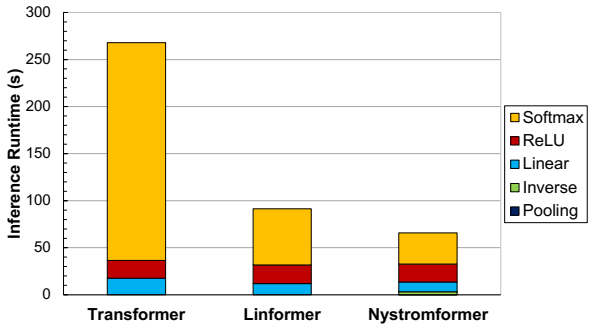
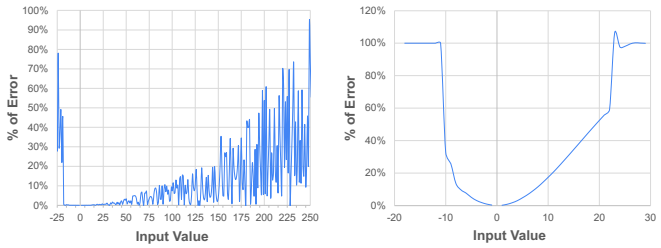


Fig. 5: 2-Party inference runtime comparison among Transformer, Linformer, and Nystromformer.

C. Dynamic Range Issues from Approximations

In Section II-A5, we discussed how certain operations in MPC protocols are approximated. However, those approximations will only produce a negligible error for inputs within a certain range. We refer to this range as a dynamic range. If an input to an approximated operation falls outside its dynamic range, the model output can become useless, resulting in low accuracy, comparable to random guesses. Figure 6a and 6b show the percentage of error $(ABS[(MPC - Actual)/Actual]\%)$ for MPC reciprocal and exponential functions, which are used within softmax. For the reciprocal, the approximated error is less than 10% when inputs are between -18 and 85. We can see that the error for the approximated reciprocal increases when the input gets bigger. For the exponential function, the approximation for inputs less than -10 result in 100% error which in turn will eventually cause significant accuracy degradation. MPC exponential functions introduce less than 10% of error for inputs between -8 and 8. Thus depending on the input value, the approximation of non-linear operations necessitated by MPC cause significant losses in accuracy.

For CNNs, the dynamic range is not a major problem because operations for CNNs can be exactly computed using MPC protocols (multiplications, ReLUs, comparisons, etc.). On the other hand, Transformer-based models require more complex non-linear operations that need to be approximated (GELU and Softmax). As a result, we found that a designer needs to pay far more attention to the dynamic range in order to get accurate prediction results for Transformer-based models in MPC. For example, the GELU function can be approximated using either Taylor approximation or $tanh$ functions. However, those approximations are unstable when input values are large. Falling outside GELU's dynamic range will destroy model accuracy leading to a random guess.



(a) Reciprocal (b) Exponential

Fig. 6: MPC approximated function percentage of error.

D. Analysis of Softmax in MPC

In this section, we perform a deeper analysis of why softmax operations are exceedingly slow in MPC. This slowdown is mainly due to the maximum function used in softmax for numerical stability. Softmax functions for the i_{th} element in a vector size of n is defined as

$$Softmax(x_i) = \frac{e^{x_i}}{\sum_{k=1}^{n} e^{x_k}} \tag{12}$$

The exponential function is approximated using the limit approximation (Section II-A5). The exponential function can explode quickly even in plaintext, causing numerical overflow when some input values are large. To achieve numerical stability, softmax is practically implemented as

$$Softmax(x_i) = \frac{e^{x_i - x_{max}}}{\sum_{k=1}^{n} e^{x_k - x_{max}}} \tag{13}$$

where x_{max} is the maximum value in the given vector. The subtraction of the maximum value does not change the final value of the softmax function, but it greatly improves the numerical stability since the largest input to the exponential function becomes 0. The "maximum" function is typically cheap in plaintext. However, in the context of MPC, the maximum function can account for the majority of inference runtime.

1) Maximum in MPC: When using MPC, inputs are additively shared among multiple parties. To obtain the maximum value from a vector with N elements, we need to perform $\mathcal{O}(log(N))$ comparison/maximum operations. The maximum between two numbers x and y can be calculated as

$$Max(x, y) = sign(y - x) \cdot x + sign(y - x) \cdot y \tag{14}$$

where $sign$ function extracts the sign bit. We describe how the sign bit can be computed with the MPC comparison algorithm in Section II-A5. To extract the sign bit of an additive share, two rounds of communication between MPC servers are required, which makes the maximum function quite expensive. Our experiments show the maximum function can make up to 88% of the total softmax runtime. The communication in the maximum function accounts for 80% of the softmax communication time.

TABLE II: Densified embedding table access runtime.

# of accesses	Communication Time	Total Runtime
32	3.53	6.42
64	3.59	6.38
128	3.35	6.47
512	3.46	9.08
1024	3.72	11.81

2) Difficulties in Replacing the Maximum: The maximum function in softmax acts as a stabilizer for the exponential function. A reciprocal function follows exponential functions to compute the softmax. As we mentioned earlier, the reciprocal function in MPC has a small dynamic range, and input values outside this range can break the model accuracy. The subtraction of the maximum makes the output of an exponential function at most 1, helping ensure the inputs to the exponential and the reciprocal can fall into their dynamic range. One may want to replace the maximum function with other stabilizers that are faster to compute in MPC. However, such optimizations are non-trivial due to the challenges related to the dynamic range: **1) exponential grows rapidly, and 2) narrow exponential & reciprocal dynamic range**. If the stabilizer is too large, the exponential functions are approximated to be zeros, resulting in model accuracy loss. On the other hand, if the stabilizer is not big enough, the output of the exponential function will fall outside the dynamic range of the approximated reciprocal function. Thus, finding a faster yet effective stabilizer for exponential functions is challenging.

E. Analysis of the Embedding Table Lookups

Finally, we turn our attention to embedding table lookups. Transformer-based NLP models use embedding tables to process categorical language tokens. As discussed earlier, embedding table lookups are implemented as row selection operations in plaintext. Thus, even when inputs to embedding tables are encrypted, memory accesses to embedding tables are correlated with inputs themselves. Consequently, failing to protect embedding table memory access patterns from adversaries (MPC servers) will compromise input confidentiality. One way to protect memory accesses to embedding tables is to implement embedding table lookups as one-hot matrix multiplications. However, our study shows that densified embedding table lookups impose serious overhead in an MPC setting. One way to reduce this overhead is to use matrix decomposition techniques to speedup embedding table accesses, with the potential for some negative impact on model inference accuracy.

1) Densified Operations: We firstly show densified embedding table inference runtime (one-hot vector & matrix multiplications) in Table II.

We measure the average time to compute a $A^T \times W$ in MPC, where A^T is $[batch \times 250002]$ and W is $[250002 \times 1024]$. Batch for A^T is equal to the total number of accesses. With more batched accesses, the communication does not change when the number of accesses is below 512. Communication costs consist of two parts: transmission delays (data size /

TABLE III: Matrix dimensions using TT decomposition.

$3/ranks$	$4/ranks$	$5/ranks$
$1 \times 50 \times 8 \times ranks$	$1 \times 20 \times 4 \times ranks$	$1 \times 10 \times 4 \times ranks$
$ranks \times 65 \times 8 \times ranks$	$ranks \times 24 \times 4 \times ranks$	$ranks \times 10 \times 4 \times ranks$
$ranks \times 80 \times 16 \times 1$	$ranks \times 25 \times 8 \times ranks$	$ranks \times 10 \times 4 \times ranks$
-	$ranks \times 25 \times 8 \times 1$	$ranks \times 13 \times 4 \times ranks$
-	-	$ranks \times 20 \times 4 \times 1$

TABLE IV: TT decomposition inference runtime improvement & MaskedLM Perplexity (PPL); Baseline PPL without TT decomposition is 12.8.

Number of Matrices	3			4			5		
Ranks	64	128	196	64	128	196	64	128	196
MaskedLM PPL	17.0	15.17	**14.04**	18.25	16.47	14.41	17.82	16.09	14.83
Batch=32 Speedup	110.02	46.70	19.58	**112.12**	38.19	22.51	120.20	41.04	19.56
Batch=64 Speedup	96.07	30.96	16.80	74.70	25.38	10.78	64.48	27.35	11.61
Batch=128 Speedup	64.45	21.63	8.82	47.21	14.80	6.58	51.99	13.27	6.57
Batch=256 Speedup	43.42	11.83	5.94	30.93	9.34	4.23	26.24	9.18	4.33
Batch=512 Speedup	31.47	7.57	3.56	18.93	5.88	2.54	15.88	6.05	2.57

bandwidth) and propagation delay (network propagation time). Though there are more bytes communicated with bigger batch sizes, the number of communication rounds for each batch size is the same. For a high-bandwidth interconnect (100Gbs), the number of rounds of communications has more impact on communication runtime. Typically, in MPC, larger batches can benefit from amortized rounds of communication, resulting better throughput.

Although communication costs are similar, our results show that computation costs grow from 6.47s to 11.81s when the number of accesses increases. The computation runtime increase comes from more multiplications when more accesses are needed.

2) TT Decomposition on Embedding Tables: Besides naive densified one-hot vector multiplications, one can use matrix decomposition to help reduce embedding table lookups. TT decomposition is one such method [11], [33]. TT decomposition can be applied to the embedding table to reduce the overhead, enabling a trade-off between performance and model accuracy. Here, we present the embedding table query runtime and model accuracy using different configurations of TT decomposition. We use $d/ranks$ to represent TT decomposition configurations. d represents the number of smaller decomposed matrices, and $ranks$ represents the rank of each decomposed matrix. Configuration $3/64$ means that the original embedding table is decomposed into 3 smaller 64-rank matrices. Throughout our experiments, the original embedding table size is still $[250002 \times 1024]$. The dimensions of smaller matrices using different configurations of TT decomposition are shown in Table III. In Table III, we can see that with more decomposed tables and fewer ranks, the compression ratio will be higher. $3/128$'s table compression ratio is 30x whereas the compression ratio of $3/64$ and $4/128$ is 115x and 208x respectively. However, this compression ratio is not free, as we show later in our accuracy study.

Table IV demonstrates embedding table inference speedups using different TT decomposition configurations with batch sizes. The baseline is the densified matrix multiplication we mentioned in the earlier section. The batch size represents the number of embedding table accesses that are performed together. When the number of batches is small, most configurations demonstrate speedups more than 15x. However, with more batched accesses, the speedups provided by TT decomposition decreases. The reason is that when there are more batched accesses, computations needed to reconstruct the original embedding indices increase with batch size (see Algorithm 4). On the contrary, the baseline's one-hot matrix multiplication's runtime does not grow linearly as we can see in Table II. The communication cost for the densified baseline does not increase linearly when the number of accesses are small, and its computation runtime increase is not apparent. Thus, the linearly increasing runtime of TT decomposition embedding accesses shows less speedups compared with the densified matrix multiplications.

The speedups and compression come with a cost. In addition to inference runtime, we also measured the TT decomposition's impact on model accuracy. We ran a masked language model on WikiText-103 [20]. The resulting perplexity scores are present in the third row of Table IV. The Transformer configuration we used is ($L = 24,\ H = 1024,\ A = 16$), and the token embedding table size is $[250002 \times 1024]$. We trained the model for ten epochs and reported the best perplexity score. When not using TT decomposition, the perplexity score is 12.8 (the lower the better). The higher the compression ratio is, the more perplexity loss there will be since more compression ratio may result in a loss in information. Thus, in our results, perplexity scores are better for configurations with more ranks. Configuration $3/196$ achieves a runtime speedup of 19.58x, while incurring a 1.19 loss in perplexity score. If applications tolerate higher loss in perplexity scores, using TT decomposition configurations such as ($3/64, 3/128, 4/64$) can achieve even more aggressive speedups.

IV. Related Work

There is a large body of previous work on MPC-based privacy preserving machine learning (PPML). For example, Astra

[2], Blaze [22], Falcon [30], CrypTFlow [15] and CrypTen [14] provide frameworks for MPC-based machine learning. In this study, we leverage CrypTen as the MPC framework. For performance, most of MPC-based PPML frameworks use the arithmetic secret sharing used in CrypTen. In that sense, we believe that the findings in this study applies to a broad class of MPC frameworks in general. Previous work including Sphynx [3], DeepReduce [12], AriaNN [23], and Circa [7] also proposed optimizations to improve the performance of MPC-based inference.

However, the previous studies focus on CNN vision models and optimizations on ReLU (bottleneck for CNNs). On the contrary, this paper characterizes Transformed-based models in the MPC setting.

CryptGPU [25] is another MPC framework based on CrypTen. Its main focus is a special family of 3-PC using 2-out-of-3 sharing, that does not scale to more parties. Our work primarily focuses on N-out-of-N sharing that can scale to more parties providing stronger security.

TT-Rec [33] uses Tensor Train decomposition to support large embedding tables size and uses a decomposed embedding cache to speed up DLRM training. Another work [11] applies Tensor Train decomposition to token embedding tables. Both of them focus on training efficiencies in plaintext, while this paper focuses on Transformer-based models in the MPC settings. Linformer [31] and Nystromformer [32] propose novel Transformer architectures that reduce Transformer computation complexity, whereas this paper studies their potential in privacy-preserving MPC protocols. This paper also demonstrates that they can achieve much higher speedups in MPC than in plaintext.

V. Conclusion

Transformer-based models are gaining popularity across a wide range of tasks from natural language processing to vision classification tasks. Given the nature of inputs processed, these models will likely represent an important privacy preserving machine learning workloads of the future. This paper presents a detailed study of the challenges in supporting Transformer-based models with MPC protocols. The study shows three new research challenges that do not exist in the private inference of CNNs: 1) significant softmax overhead, 2) secure accesses to embedding tables, and 3) limited dynamic ranges of approximated non-linear operations.

Our characterization study shows that the maximum function used for numerical stability is the main runtime bottleneck in softmax. Computing maximum in MPC requires comparisons of secret shares, which in turn require data sharing format conversion. These conversions induce a large amount of communications among MPC parties. Our studies also show that secure accesses to embedding tables using densified matrix multiplications add significant runtime overhead. However, one can use tensor-train decomposition to significantly reduce the runtime with a certain loss in the model accuracy. Finally, our paper also shows that approximated complex non-linear functions have a dynamic range. Inputs outside that range can significantly degrade model accuracy if model designers are not careful when applying MPC protocols to their models.

References

[1] D. W. Arthur, "C. T. H. Baker, The Numerical Treatment of Integral Equations (Clarendon Press; Oxford University Press, 1978), xiv 1034 pp., £22–50." *Proceedings of the Edinburgh Mathematical Society*, vol. 22, no. 1, p. 67–67, 1979.

[2] H. Chaudhari, A. Choudhury, A. Patra, and A. Suresh, "ASTRA: high throughput 3pc over rings with application to secure prediction," in *Proceedings of the 2019 ACM SIGSAC Conference on Cloud Computing Security Workshop, CCSW@CCS 2019, London, UK, November 11, 2019*, R. Sion and C. Papamanthou, Eds. ACM, 2019, pp. 81–92. [Online]. Available: https://doi.org/10.1145/3338466.3358922

[3] M. Cho, Z. Ghodsi, B. Reagen, S. Garg, and C. Hegde, "SPHYNX: Relu-efficient network design for private inference," 2021.

[4] A. Conneau and K. Khandelwal, "Unsupervised cross-lingual representation learning at scale," *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 2020.

[5] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," *Proceedings of NAACL-HLT 2019*, 2018.

[6] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. G. andJakob Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," *Proceedings of Ninth ICLR*, 2021.

[7] Z. Ghodsi, N. K. Jha, B. Reagen, and S. Garg, "Circa: Stochastic relus for private deep learning," *arXiv preprint arXiv:2106.08475*, 2021.

[8] O. Goldreich, "Secure multi-party computation," 1998.

[9] H. Hashemi, Y. Wang, and M. Annavaram, "Darknight: A data privacy scheme for training and inference of deep neural networks," *Proceedings on the 54th International Symposium on Microarchitecture*, 2021.

[10] D. Hendrycks and K. Gimpel, "Gaussian error linear units (gelus)," 2020.

[11] O. Hrinchuk, V. Khrulkov, L. Mirvakhabova, E. Orlova, and I. Oseledets, "Tensorized embedding layers for efficient model compression," *Proceedings of Ninth ICLR*, 2021.

[12] N. K. Jha, Z. Ghodsi, S. Garg, and B. Reagen, "DeepReDuce: Relu reduction for fast private inference," *Proceedings of the 38 th International Conference on Machine Learning*, 2021.

[13] S. Johnson, V. Scarlata, C. Rozas, E. Brickell, and F. Mckeen, "Intel software guard extensions: Epid provisioning and attestation services," 2016.

[14] B. Knott, S. Venkataraman, A. Hannun, S. Sengupta, M. Ibrahim, and L. van der Maaten, "Crypten: Secure multi-party computation meets machine learning," in *arXiv 2109.00984*, 2021.

[15] N. Kumar, M. Rathee, N. Chandran, D. Gupta, A. Rastogi, and R. Sharma, "CrypTFlow: Secure tensorflow inference," in *IEEE Symposium on Security and Privacy*. IEEE, May 2020. [Online]. Available: https://www.microsoft.com/en-us/research/publication/cryptflow-secure-tensorflow-inference/

[16] G. Lample and A. Conneau, "Cross-lingual language model pretraining," *33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.*, 2019.

[17] B. Lefaudeux, F. Massa, D. Liskovich, W. Xiong, V. Caggiano, S. Naren, M. Xu, J. Hu, M. Tintore, and S. Zhang, "xFormers: A modular and hackable Transformer modelling library," https://github.com/facebookresearch/xformers, 2021.

[18] A. Limted, "Arm security technology building a secure system using trustzone technology," 2016.

[19] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "RoBERTa: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[20] S. Merity, C. Xiong, J. Bradbury, and R. Socher, "Pointer sentinel mixture models," *arXiv preprint arXiv:1609.07843*, 2016.

[21] K. G. Narra, Z. Lin, Y. Wang, K. Balasubramaniam, and M. Annavaram, "Privacy-preserving inference in machine learning services using trusted execution environments," *IEEE International Conference on Cloud Computing*, 2021.

[22] A. Patra and A. Suresh, "BLAZE: blazing fast privacy-preserving machine learning," *CoRR*, vol. abs/2005.09042, 2020. [Online]. Available: https://arxiv.org/abs/2005.09042

[23] T. Ryffel, P. Tholoniat, D. Pointcheval, and F. Bach, "Ariann: Low-interaction privacy-preserving deep learning via function secret sharing," 2020. [Online]. Available: https://arxiv.org/abs/2006.04593

[24] X. Sun, P. Zhang, J. K. Liu, J. Yu, and W. Xie, "Private machine learning classification based on fully homomorphic encryption," *IEEE Transactions on Emerging Topics in Computing*, vol. 8, no. 2, pp. 352–364, 2020.

[25] S. Tan, B. Knott, Y. Tian, and D. J. Wu, "Cryptgpu: Fast privacy-preserving machine learning on the gpu," 2021. [Online]. Available: https://arxiv.org/abs/2104.10949

[26] Y. Tay, M. Dehghani, D. Bahri, and D. Metzler, "Efficient transformers: A survey," 2020, cite arxiv:2009.06732. [Online]. Available: http://arxiv.org/abs/2009.06732

[27] D. L. C. Thekkath, M. Mitchell, P. Lincoln, D. Boneh, J. Mitchell, and M. Horowitz, "Architectural support for copy and tamper resistant software," *ACM SIGARCH Computer Architecture News*, 2000.

[28] F. Tramèr and D. Boneh, "Slalom: Fast, verifiable and private execution of neural networks in trusted hardware," *arXiv preprint arXiv:1806.03287*, 2019.

[29] A. Vaswani and N. Shazeer, "Attention is all you need," *31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.*, 2017.

[30] S. Wagh, S. Tople, F. Benhamouda, E. Kushilevitz, P. Mittal, and T. Rabin, "FALCON: Honest-majority maliciously secure framework for private deep learning," *Proceedings on Privacy Enhancing Technologies*, 2020.

[31] S. Wang, B. Z. Li, M. Khabsa, H. Fang, and H. Ma, "Linformer: Self-attention with linear complexity," *arXiv preprint arXiv:2006.04768*, 2020.

[32] Y. Xiong, Z. Zeng, R. Chakraborty, M. Tan, G. Fung, Y. Li, and V. Singh, "Nyströmformer: A nyström-based algorithm for approximating self-attention," *Association for the Advancement of Artificial Intelligence*, 2021.

[33] C. Yin, B. Acun, X. Liu, and C.-J. Wu, "TT-REC: Tensor train compression for deep learning recommendation model embeddings," *arXiv preprint arXiv:2101.11714*, 2021.