Characterizing and Improving MPC-based Private Inference for Transformer-based Models

Anonymous Author(s)
Affiliation
Address
email

Abstract

Secure multi-party computation (MPC) is gaining popularity with the growing demand for privacy-preserving cloud services. While there has been plenty of attention paid to MPCs for convolution neural networks (CNNs) [1, 2, 3, 4, 5], MPC-based private inference for Transformer models has not been studied in detail. This paper provides a characterization study of the performance overhead for running Transformer models with secure MPC, and proposes an optimization for embedding tables. Our study shows that Transformers introduce a couple of new challenges for MPC-based private inference: softmax and embedded tables. To address the overhead of embedding table accesses under MPC, we propose to use tensor-train (TT) decomposition, a mechanism that splits a large embedding tables into multiple smaller embedding tables. For the NLP workloads, the experiments show that the TT decomposition can speed up embedding table accesses by **2x** with only a 1.19 drop in the masked-language model perplexity score.

1 Introduction

Data privacy is a pressing concern for privacy-preserving machine learning (PPML) in the cloud. To address this challenge, secure computation techniques such as homomorphic encryption (HE) [6], trusted execution environments (TEE) [7, 8, 9, 10, 11, 12], and secure multi-party computation (MPC) [13] have been applied to PPML. The previous studies, however, focused primarily on convolutional neural networks (CNNs) [1, 2, 3, 4, 5]. This paper studies MPC-based private inference for Transformer-based models, which are commonly used for natural language processing (NLP) and have been applied to computer vision [14] more recently.

As the first step to enable efficient MPC-based private inference for Transformer models, we performed a detailed study of the performance overhead. The study uncovers two challenges, which are new to Transformers and have not been studied in the context of CNNs: softmax and embedding tables. 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 portion of the MPC execution time for Transformers. Unlike CNNs where softmax only needs to be performed at the end, Transformers use softmax in each layer. An in-depth investigation also shows that the max function that is used for numerical stability is the main source of softmax overhead.

Another new challenge for Transforms comes from embedding tables, which convert categorical data into continuous data. Embedding tables can be as large as 1GB for NLP models, and embedding table look-ups are commonly implemented as row selection operations. However, because embedding table indices are from secret input data, embedded table accesses must be oblivious to input values in private inference. To implement embedding tables securely in the MPC setting, one can "densify" embedding table lookups by turning them into matrix multiplications. Unfortunately, our characterization study

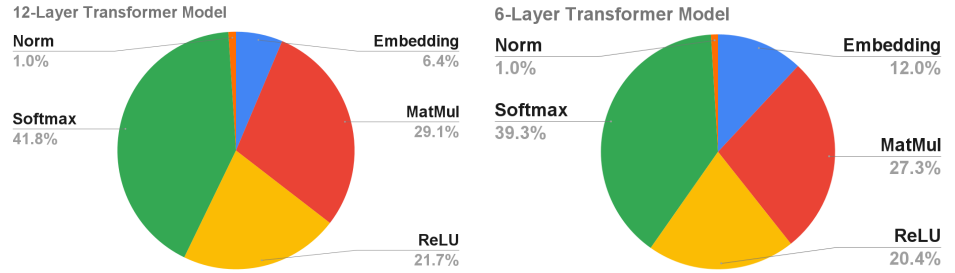
Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

shows that replacing an embedding table look-up with a matrix multiplication significantly increases the execution time of embedding table operations.

For more efficient embedding table operations in MPC, we propose to utilize embedding tables' compressibility. In particular, we use tensor train (TT) decomposition. The TT decomposition has been applied to plaintext machine learning models [15, 16] to reduce the memory footprint and speed up training processes. We exploit TT decomposition to reduce the size of matrices that encode embedding tables and reduce the overhead for embedding table accesses. The experiments show that the TT decomposition can speed up embedding table accesses by **2x** with only a 1.19 drop in the masked-language model perplexity score.

2 Transformer-based Model MPC Inference Characterization

Secure Multi-Party Computation. In the MPC setting, an MPC client wishes to have remote servers (non-colluding MPC servers) perform a sensitive task, such as translating classified sentences, without revealing secret data and/or model parameters. The secret data is encrypted into two secret shares, in a way that each share does not leak any information about the secret. There are two main secret sharing formats that we use: additive and binary format. The binary secret sharing format is suitable for bit-wise operations, whereas additive shares are more suitable for addition and multiplication. Details about those formats can be found in [17]. After MPC servers receive their secret share, each cloud server only sees and manipulates its own secret share, and returns the results to the client. The client can combine the results from both servers to obtain the plaintext result. While MPC requires that multiple parties do not collude and incur non-trivial communication overhead, MPC represents one of the most promising techniques for PPML as its overhead is often much lower compared to HE.



(a) A 12-layer Transformer-based Model Operator 2-PC Inference Runtime Decomposition

(b) A 6-layer Transformer-based Model Operator 2-PC Inference Runtime Decomposition

Figure 1: Transformer-based Model Operator 2-PC Inference Runtime Decomposition

Experimental Setup. In our characterization study, we obtained a secure 2-PC inference runtime. We implemented MPC models using the CrypTen MPC framework [17]. The token embedding table size is $[250002 \times 1024]$, and there are 6/12 layers of Transformers in the model ($L = 6/12,\ H = 1024,\ A = 16$). Embedding tables are implemented as "densified" matrix multiplications. Runtimes are obtained from using two nodes on the same server rack, and each node has an NVIDIA Tesla V100 Volta GPU.

MPC vs. Plaintext Inference. Comparing with the plaintext single GPU inference, total inference runtime is 12x slower with MPC. Embedding table accesses are $2,523$x slower; Matrix multiplications are 3.5x slower; Softmax functions are 465x slower; and ReLU functions are $1,372$x slower. Among all the operators, embedding tables and softmax sees the most overhead when MPC protocols are implemented.

MPC Execution Time Breakdown. Figure 1a and Figure 1b show the breakdown of inference runtime. Embedding table accesses make up 12% of total inference runtime. Softmax functions and activation functions make up 40% and 28% of total inference runtime, respectively. Lightweight non-linear functions such as ReLUs and Softmax dominate the inference runtime when MPC is used because they need to be approximated with arithmetic or binary functions. Note that in the above experiments the length of each input sentence is 128. If we increase the number of tokens in each sentence, MPC softmax runtime grows more rapidly than all other operators because the input size

of softmax increases quadratically with sentence length. When token length of input sentences is 1024, softmax can account for 85% of the total inference runtime, and the total 2-party Transformer inference is 316x slower than the computation in plaintext.

Analysis of Softmax Overhead. Among all the operations, softmax shows the largest showdown when MPC is applied. This slowdown is mainly due to a maximum function used in softmax for numerical stability. Softmax functions for the i^{th} element in a vector size of n is defined as

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

The exponential function is approximated using limit approximation [17]. The exponential function can explode quickly even in plaintext, causing overflow when some input values are big. To achieve numerical stability, softmax is practically implemented as

$$Softmax(x_i) = \frac{e^{x_i - x_{max}}}{\sum_{k=1}^{n} e^{x_k - x_{max}}} \tag{2}$$

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. The "maximum" function is typically cheap. However, in the context of MPC, the maximum function turns into an extremely expensive operation. When using MPC, inputs are additively shared among multiple parties. The maximum function requires a large number of comparisons, which are expensive in MPC because comparisons require two secret sharing format conversions. This communication during secret sharing format conversions makes maximum functions expensive, and there are $\mathcal{O}(log(N))$ comparisons. Our experiments show that if the maximum function is removed (for timing purpose only) from the softmax operation, the softmax function becomes 7x and 9x faster on CPU and GPU, respectively.

3 Tensor-Train Decomposition for Efficient Embedding Tables

Tensor-Train (TT) Decomposition. The basic idea of TT decomposition is to represent a big matrix using tensor products of several smaller matrices. A tensor product is a function:

$$\mathbb{R}^{M_1 \times M_2 \ldots \times M_k} \bigotimes \mathbb{R}^{N_1 \times N_2 \ldots \times N_k} \rightarrow \mathbb{R}^{M_1 \cdot N_1 \times M_2 \cdot N_2 \ldots \times M_k \cdot N_k} \tag{3}$$

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 are 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, a entry in every decomposed matrices is fetched (this fetching is implemented as dense one-hot matrix multiplications). To reconstruct the original entry, apply dot products at the dimension of the rank among fetched entries.

TT Decomposition for Embedding Tables in MPC. TT decomposition can be applied to the matrix multiplications that are used for embedded tables 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 $[250002 \times 1024]$. With fixed configurations, the dimensions of smaller matrices using different configurations of TT decomposition are shown in Table 1. The performance results are obtained by running 2-party inference using CrypTen [17] on two nodes on the same server rack, each node with an NVIDIA Tesla V100 Volta GPU. To evaluate the impact of model accuracy, the experiments on a masked language model are performed without MPC on a single node with 8 NVIDIA Tesla V100 Volta GPUs.

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

Table 1: Matrix Dimensions using TT Decomposition

Number of Matrices	3			4			5		
Ranks	64	128	196	64	128	196	64	128	196
Batch=32 Speedup	16.37	6.95	2.91	16.68	5.68	3.35	17.88	6.11	2.91
Batch=64 Speedup	11.93	3.84	2.09	9.28	3.15	1.34	8.01	3.40	1.44
Batch=128 Speedup	8.65	2.90	1.18	6.34	1.99	0.88	6.98	1.78	0.88
Batch=256 Speedup	6.56	1.79	0.90	4.67	1.41	0.64	3.97	1.39	0.65
Batch=512 Speedup	4.95	1.19	0.56	2.98	0.93	0.40	2.50	0.95	0.40

Table 2: TT Decomposition Inference Runtime Improvement

Table 2 demonstrates embedding table inference speedups using different TT decomposition configurations with batch sizes. The batch size represents the number of embedded table accesses that are performed together. When batched accesses are small, most configurations demonstrate speedups. However, with more batched accesses, the configurations with more decomposed matrices and more ranks begin to show negative speedups. Computations needed and the number of bytes of data communicated for reconstructing embedding entries from decomposed matrices increases linearly with the numbers of accesses. On the contrary, the baseline's one-hot matrix multiplication's runtime does not grow linearly because the number of communication rounds is amortized as the number of batched accesses grows (the size of the embedding table does not depend on batch sizes). The results suggest that TT decomposition can significantly improve performance for cases with small batch sizes as in real-time inference settings. For example, users' inputs to real-time translation software are generally no longer than 50 tokens where one token corresponds to one embedding table access; length of 90% data points in MNLI[18], XNLI[19] and CoNLL-2003 [20] is smaller than 75.

The speedups and compression come with a cost. Besides inference runtime reduction, we also measured the TT decomposition's impact on model accuracy. We have run a masked language model on WikiText-103 [21]. The Transformer configuration we used is ($L = 24,\ H = 1024,\ A = 16$), and the token embedding table size is $[250002 \times 1024]$. We have trained the model for ten epochs and reported their best perplexity score. Table 3 presents perplexity of various TT decomposition configurations. Configuration $3/64$ achieves a speedup of 2.09x, while incurring a 1.19 loss in perplexity score. If applications can tolerate higher loss in perplexity scores, using TT decomposition configurations such as ($3/64, 3/128,\ 4/64$) can achieve even more speedups.

4 Conclusions

Transformers are widely used in NLP tasks and will likely represent an important workload for PPML in the future. Interestingly, this paper shows that Transformers introduce new research challenges that do not exist for private inference of CNNs. While this paper shows that TT decomposition can be used to speed up embedding table accesses, further optimizations, especially for non-linear operations such as softmax, are needed to enable private real-time NLP inference.

Configs	Plaintext	$3/64$	$3/128$	$3/196$	$4/64$	$4/128$	$3/196$	$5/64$	$5/128$	$5/196$
PPL	12.8	17.0	15.17	14.04	18.25	16.47	14.41	17.82	16.09	14.83

Table 3: Masked Language Model Perplexity Score

References

[1] Nishant Kumar, Mayank Rathee, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. Cryptflow: Secure tensorflow inference. In *IEEE Symposium on Security and Privacy*. IEEE, May 2020.

[2] Sameer Wagh, Shruti Tople, Fabrice Benhamouda, Eyal Kushilevitz, Prateek Mittal, and Tal Rabin. Falcon: Honest-majority maliciously secure framework for private deep learning. *Proceedings on Privacy Enhancing Technologies*, 2020.

[3] Minsu Cho, Zahra Ghodsi, Brandon Reagen, Siddharth Garg, and Chinmay Hegde. Sphynx: Relu-efficient network design for private inference. 2021.

[4] Nandan Kumar Jha, Zahra Ghodsi, Siddharth Garg, and Brandon Reagen. Deepreduce: Relu reduction for fast private inference. *Proceedings of the 38 th International Conference on Machine Learning*, 2021.

[5] Zahra Ghodsi, Nandan Kumar Jha, Brandon Reagen, and Siddharth Garg. Circa: Stochastic relus for private deep learning. *arXiv preprint arXiv:2106.08475*, 2021.

[6] Xiaoqiang Sun, Peng Zhang, Joseph K. Liu, Jianping Yu, and Weixin Xie. Private machine learning classification based on fully homomorphic encryption. *IEEE Transactions on Emerging Topics in Computing*, 8(2):352–364, 2020.

[7] Simon Johnson, Vinnie Scarlata, Carlos Rozas, Ernie Brickell, and Frank Mckeen. Intel software guard extensions: Epid provisioning and attestation services. 2016.

[8] David Lie Chandramohan Thekkath, Mark Mitchell, Patrick Lincoln, Dan Boneh, John Mitchell, and Mark Horowitz. Architectural support for copy and tamper resistant software. *ACM SIGARCH Computer Architecture News*, 2000.

[9] ARM Limted. Arm security technology building a secure system using trustzone technology. 2016.

[10] Florian Tramèr and Dan Boneh. Slalom: Fast, verifiable and private execution of neural networks in trusted hardware. *arXiv preprint arXiv:1806.03287*, 2019.

[11] Krishna Giri Narra, Zhifeng Lin, Yongqin Wang, Keshav Balasubramaniam, and Murali Annavaram. Privacy-preserving inference in machine learning services using trusted execution environments. *IEEE International Conference on Cloud Computing*, 2021.

[12] Hanieh Hashemi, Yongqin Wang, and Murali Annavaram. Darknight: A data privacy scheme for training and inference of deep neural networks. *Proceedings on the 54th International Symposium on Microarchitecture*, 2021.

[13] Oded Goldreich. Secure multi-party computation. 1998.

[14] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly andJakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *Proceedings of Ninth ICLR*, 2021.

[15] Oleksii Hrinchuk, Valentin Khrulkov, Leyla Mirvakhabova, Elena Orlova, and Ivan Oseledets. Tensorized embedding layers for efficient model compression. *Proceedings of Ninth ICLR*, 2021.

[16] Chunxing Yin, Bilge Acun, Xing Liu, and Carole-Jean Wu. Tt-rec: Tensor train compression for deep learning recommendation model embeddings. *arXiv preprint arXiv:2101.11714*, 2021.

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

[18] Adina Williams, Nikita Nangia, and Samuel R. Bowman. A broad-coverage challenge corpus for sentence understanding through inference. *arXiv preprint arXiv:1704.05426*, 2018.

[19] Alexis Conneau, Guillaume Lample, Ruty Rinott, Adina Williams, Samuel R. Bowman, Holger Schwenk, and Veselin Stoyanov. Xnli: Evaluating cross-lingual sentence representations. *arXiv preprint arXiv:1809.05053*, 2018.

[20] Erik F., Tjong Kim Sang, and Fien De Meulder. Introduction to the conll-2003 shared task: Language-independent named entity recognition. *arXiv preprint arXiv:0306050*, 2003.

[21] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.