MPCFORMER: FAST, PERFORMANT AND PRIVATE TRANSFORMER INFERENCE WITH MPC

Anonymous authors
Paper under double-blind review

ABSTRACT

Enabling private inference is crucial for many cloud inference services that are based on Transformer models. However, existing private inference solutions for Transformers can increase the inference latency by more than 60× or significantly compromise the quality of inference results. In this paper, we design the framework MPCFORMER using secure multi-party computation (MPC) and Knowledge Distillation (KD). It can be used in tandem with many specifically designed MPC-friendly approximations and trained Transformer models. We use MPCFORMER to significantly speed up Transformer model inference in MPC settings while achieving similar ML performance to the original model. We evaluate MPCFORMER with various settings in MPC. On the IMDb dataset, we achieve similar performance to $\text{BERT}_{\text{BASE}}$, while being 5.3× faster. On the GLUE benchmark, we achieve 97% performance of $\text{BERT}_{\text{BASE}}$ with a 2.2× speedup. We show that MPCFORMER remains effective with different trained Transformer weights such as $\text{ROBERTA}_{\text{BASE}}$ and larger models including $\text{BERT}_{\text{Large}}$. In particular, we achieve similar performance to $\text{BERT}_{\text{LARGE}}$, while being 5.93× faster on the IMDb dataset.

1 INTRODUCTION

Pre-trained Transformer models can easily adapt to various downstream tasks with high performance, then be deployed as model inference services (Bommasani et al., 2021; Feng et al., 2020; Yang et al., 2019b; Clark et al., 2020). However, such model inference services suffer from privacy concerns. For instance, GitHub Copilot, a code-generating engine adapted from pre-trained GPT weights (Chen et al., 2021; Brown et al., 2020), requires either users to reveal their code prompts to the service provider for code generation, or the service provider to release the Copilot's trained weights, which are business proprietary, to users.

Secure Multi-Party Computation (MPC) offers a promising solution by keeping data and models private during the inference process (Evans et al., 2018). However, the vanilla Transformer inference computation in MPC is unacceptably slow. For instance, $\text{BERT}_{\text{BASE}}$ takes < 1 second without MPC, but ~ 60 seconds with MPC (Figure 2). Prior works on convolutional neural networks (CNNs) speed up the inference in MPC by replacing computational operations with faster approximations (we call these *MPC-friendly approximations*) (Mohassel & Zhang, 2017). However, such a simple replacement strategy drastically decreases the models' quality (Chou et al., 2018).

In this paper, we take the first step to tackle the research problem: *How to perform privacy-preserving Transformer model inference in MPC, while remaining fast and performant?* In particular, we seek a simple and effective solution that allows various MPC-friendly approximations and Transformer weights to be plugged in.

In this paper, we explore a new two-stage approach – we first make transformer inference fast in MPC, then make fast Transformers performant. Drawing insights from existing secure inference methods for CNNs, we first validate that using MPC-friendly approximations can also bring speedups to Transformer models. More concretely, we benchmark the transformer inference process with an MPC system (Knott et al., 2021) and find the major bottlenecks are GeLU and Softmax functions (Wang et al., 2022). Replacing them with off-the-shelf MPC-friendly approximations significantly speeds up the whole process (See § 4.2.1).

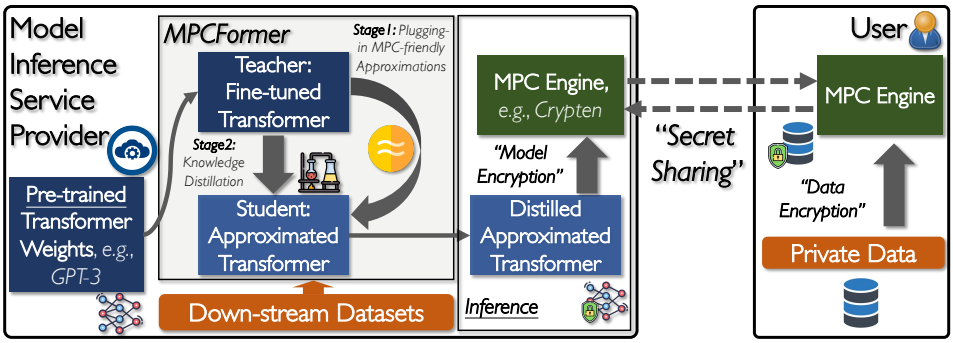

Figure 1: An illustration of our proposed MPCFORMER framework. MPCFORMER takes a trained (or finetuned) Transformer model and adopts given MPC-friendly approximations, then uses KD on the downstream datasets to construct high-quality models. During inference time, MPCFORMER leverages an MPC engine to attain private model inference.

Unlike existing methods, in the second stage, we seek to make the fast approximated Transformer performant. We validate that simply training the fast approximated architecture is not performant (§ 5). There are two possible causes: (1) many MPC-friendly approximations are not compatible with common optimization algorithms, *e.g.,* Gradient Descent(GD) (Ruder, 2016). For instance, quadratic functions are fast in MPC but their large gradients cause the GD to diverge (Mishra et al., 2020). (2) The task-specific objective is not efficient enough to train a good approximated Transformer model on small downstream datasets. We resort to the *knowledge distillation* (KD) framework to address both causes. First, KD is effective in guiding the learning of an otherwise hard-to-train model (Hinton et al., 2015; Sanh et al., 2019; Jiao et al., 2019). In our formulation, we use the original Transformer model to guide the new model with approximations, which is faster in inference but less accurate. In addition, prior work has shown that the KD objective can guide Transformer models to learn well on small downstream datasets (Touvron et al., 2021). We empirically validate this in our setting (§ 5.3).

Our approach. In this paper, we build MPCFORMER, an *easy-to-adopt* framework for fast, performant, and private Transformer inference. MPCFORMER is compatible with many MPC-friendly approximations and trained Transformer models. It first replaces bottleneck functions in the original Transformer model with the chosen MPC-friendly approximations. The resulting approximate Transformer model has a faster inference speed in the MPC setting. MPCFORMER then applies knowledge distillation to the approximated Transformer model, by using the original performant Transformer model as the teacher. The strong teacher guidance and the data efficiency of KD train a performant approximated Transformer model with downstream datasets. Finally, the model provider can use the distilled approximated Transformer on top of an MPC engine, *e.g.,* Crypten for private model inference service, achieving high inference speed and high ML performance simultaneously. The overall workflow of MPCFORMER is shown in Figure 1.

Our contributions. Our contributions are three-fold.

1. We propose MPCFORMER, a two-stage framework that allows many MPC-friendly approximations and Transformer models to be plugged in to achieve fast and accurate private Transformer model inference with MPC.
2. We leverage our framework with an MPC system to speed up Transformer inference, by plugging in MPC-friendly approximations and trained Transformer models. In the process, we design a new and faster MPC-friendly approximation to the Softmax function.
3. We extensively evaluate the framework with plugged-in approximations and trained Transformers in the MPC setting. On the IMDb benchmark, we achieve similar ML performance to $\text{BERT}_{\text{BASE}}$ with 5.3× speedup. We achieve similar ML performance to $\text{BERT}_{\text{LARGE}}$ with 5.9× speedup. On the GLUE benchmark, we achieve 97% performance of $\text{BERT}_{\text{BASE}}$ with 2.2× speedup. MPCFORMER is also effective when plugged in with different trained Transformer models, *e.g.,* $\text{RoBERTa}_{\text{BASE}}$.

2 BACKGROUND

In this section, we first describe the Transformer model. Then we describe how functions in Transformer models can be implemented in MPC, and point out performance bottlenecks.

2.1 TRANSFORMER MODELS

An n-layer Transformer model consists of three components: (1) The embedding layer. (2) A stack of n Transformer layers. (3) The prediction layer. The embedding layer maps a token (e.g. a word or an image patch) to a hidden representation (Devlin et al., 2018; Dosovitskiy et al., 2020). One Transformer layer consists of an attention module and several matrix multiplication modules (Bahdanau et al., 2014). The prediction layer maps the output of the last Transformer layer to a task output (*e.g.,* a probability distribution for a classification task). A partial illustration of a Transformer model can be found in Figure 3.

2.2 TRANSFORMER MODELS IN MPC

The Transformer model inference process can be formulated as a 2-Parties Computation (2PC). In 2PC, the user party inputs the data, and the model provider party inputs the Transformer model. They jointly compute an inference result. Throughout the entire inference process, 2PC guarantees both parties only know information about their own inputs and the result (Yang et al., 2019a).

We describe the **secret sharing** scheme as a mean to preserve privacy during the inference process (Damgård et al., 2012; Goldreich et al., 2019). Assuming that the user provides a number x as its input, the secret sharing scheme splits x into two numbers, x_1 and x_2. It then lets the user party hold x_1 and distributes x_2 to the model provider party. There are two properties of x_1 and x_2. First, either x_1 or x_2 alone contains no information about x. This property allows the user to hide the actual value x from the model provider. Second, together they reconstruct x. For instance, x_1 and x_2 add up to x: $x = x_1 + x_2$. The second property allows joint computation.

We take multiplication via Beaver triple as a joint computation example (Beaver, 1991). In multiplication, the user party provides x and the model provider provides y, and they secret share x and y. Thus, the user gets x_1 and y_1; the model provider gets x_2, and y_2. Beaver triple assumes a triple $c = ab$ has been generated [1]. The triple is also secret shared so that the user party gets c_1, a_1, b_1, and the model provider gets c_2, a_2, b_2. The user first computes $\epsilon_1 = x_1 - a_1$, $\delta_1 = y_1 - b_1$ locally. The model provider similarly computes $\epsilon_2 = x_2 - a_2$, $\delta_2 = y_2 - b_2$ locally. They communicate these four numbers and reconstruct $\epsilon = \epsilon_1 + \epsilon_2$, $\delta = \delta_1 + \delta_2$. the user then use these two values to compute $r_1 = c_1 + \epsilon b_1 + \delta a_1 + \delta\epsilon$. The model provider computes $r_2 = c_2 + \epsilon b_2 + \delta a_2$. At this point, the multiplication result xy can be reconstructed by $xy = r_1 + r_2$.

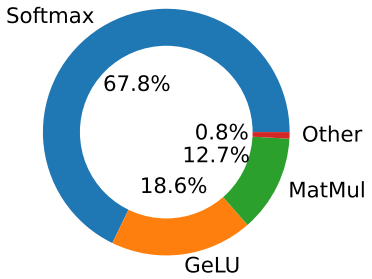

Figure 2: $\text{BERT}_{\text{BASE}}$ model (12-layers, 512 tokens) run-time breakdown evaluated on an MPC system. The overall runtime with MPC is 59.0 seconds, but < 1 second without MPC.

Table 1: The communication statistics explain the left figure. Assuming a 10GbE bandwidth, 50.3GB of softmax takes 40.3 seconds to transfer.

Primitives	Comm. (rounds)
Addition	0
Multiplication	1
Comparison	7
functions	Comm. (volume)
MatMul	3.5GB
GeLU	14.8GB
Softmax	50.3GB

There are two important observations in the multiplication example: (1) it does not leak information to the other party. For instance, the user does not send x_1 to the model party. Instead, it sends $\epsilon_1 = x_1 - a_1$ where a_1 is a random mask; (2) it requires one extra round of communication compared to the multiplication without MPC. This partially explains why vanilla Transformer models are slow in MPC. In particular, functions in Transformers (e.g., nonlinear activation) can

[1] For example, through oblivious transfer (Keller et al., 2016) or homomorphic encryption (Paillier, 1999).

be mainly implemented by three primitives [2], i.e., addition, multiplication, and comparison. Any computational operations composed by these primitives would result in extra complexity in MPC [3].

Empirically, we show these complexities by running $\text{BERT}_{\text{BASE}}$ (Figure 2) and reporting communication statistics in Table 1 with a secret-sharing-based MPC system (Knott et al., 2021). We observe that GeLU functions and Softmax functions in Transformer layers are the major sources of bottlenecks. $\text{GeLU}(x) = x \times \frac{1}{2}\left[1 + \text{erf}\left(\frac{x}{\sqrt{2}}\right)\right]$ is slow because the Gaussian Error function $\text{erf}(\cdot)$ is evaluated by a high order Taylor expansion, which requires many multiplication primitives. $\text{Softmax}(\mathbf{x}_i) = \frac{\exp(\mathbf{x}_i)}{\sum_j \exp(\mathbf{x}_j)}$ is slow because (1) The exponential function is evaluated by several iterations of squaring, which requires many multiplication primitives; (2) the maximum operation over $\mathbf{x}$ is required for numerical stability (Paszke et al., 2019), which requires comparison primitives.

3 Related work

MPC. Secure Multi-party Computation(MPC) enables joint computation between parties while keeping inputs private. The privacy feature and rich support of systems have made it suitable for Transformer inference (Mohassel & Zhang, 2017; Liu et al., 2017; Mohassel & Rindal, 2018; Riazi et al., 2018; Juvekar et al., 2018; Wagh et al., 2019; Mishra et al., 2020; Knott et al., 2021). In this paper, we do not aim to implement a new MPC system. Rather, we aim to develop an algorithmic solution to speed up Transformer inference that can be portable across many MPC systems.

Transformer models. Transformer models have achieved great success in language understanding Yang et al. (2019b); Lan et al. (2019); Raffel et al. (2020); Clark et al. (2020), vision understanding Dosovitskiy et al. (2020); Liu et al. (2021); Radford et al. (2021), and beyond (Sharir et al., 2021). In particular, the two-stage training strategy for Transformer models has been shown to be effective in extensive settings and has become the domincated paradigm (Liu et al., 2019; Radford et al., 2018; Turc et al., 2019). In this training strategy, Transformer models are first pre-trained on a large dataset for general understanding, and then fine-tuned on a small downstream dataset to learn task-specific features. In this work, we consider this paradigm as the default setting, where we assume that model providers use pre-trained Transformer weights from elsewhere, and only have downstream data.

MPC-friendly approximations. Existing research has developed MPC-friendly approximations to speed up CNN computation in MPC. Chou et al. (2018) develops an optimization framework that minimizes the approximation error of order 2 polynomial to ReLU: $\text{ReLU}(x) = 0.125 \times x^2 + 0.25 \times x + 0.5$. This introduces a significant accuracy drop because the quadratic activation causes the Gradient Descent(GD) algorithm to diverge. Mishra et al. (2020) alleviates this problem by using a set of carefully designed heuristics along with Neural Architecture Search (NAS). Mohassel & Zhang (2017) proposes an approximation to softmax by replacing exponential with ReLU functions. We do not focus on developing heuristics for a single pair of bottleneck functions and approximations. Rather, we focus on developing a general framework that can consistently output a performant Transformer model with various approximations.

Knowledge Distillation(KD). KD transfers knowledge from the teacher model to the student model by matching their hidden representations (Hinton et al., 2006). Several research has designed effective objectives for Transformer models (Sanh et al., 2019; Jiao et al., 2019; Dosovitskiy et al., 2020) such as matching the attention matrices. In particular, Sanh et al. (2019) and Jiao et al. (2019) have a different goal than us and train on the pre-training dataset as well. However, we share the same assumption on the model providers' side — they only have downstream datasets.

4 Method

In this section, we present the MPCFormer framework. MPCFormer allows the model provider to convert its Transformer model to a faster and performant one for private inference service. In 4.1, we introduce the workflow of MPCFormer (Figure 1), followed by the details of each step in 4.2.

[2] We only consider primitives that take two secret share numbers for the ease of illustration.

[3] We provide more details on implementing primitives and functions in MPC at A.1.

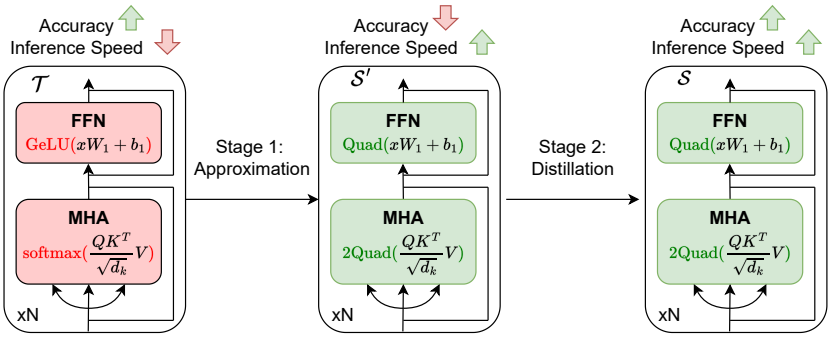

Figure 3: The overview of the MPCFORMER framework. The first stage uses MPC-friendly approximations to construct a faster Transformer architecture $\mathcal{S}'$ from $\mathcal{T}$. The second stage uses Knowledge Distillation to learn a performant and fast Transformer model $\mathcal{S}$ based on $\mathcal{S}'$.

4.1 HIGH-LEVEL WORKFLOW

In the inference service, a model provider holds a Transformer model $\mathcal{T}$, and the user holds data X. They reach to an agreement on an MPC system to perform private inference.

In §2, we illustrate that using $\mathcal{T}$ to perform the inference in the MPC system is slow. Instead of using $\mathcal{T}$, the model provider can use MPCFORMER to generate a more suited one $\mathcal{S}$. $\mathcal{S}$ runs much faster than $\mathcal{T}$ in the MPC setting while having similar ML performance compared to $\mathcal{T}$.

To use MPCFORMER, the model provider needs to provide the original model $\mathcal{T}$, the downstream dataset $\mathcal{D}$, and MPC-friendly approximations $\mathcal{A}$. These MPC-friendly approximations $\mathcal{A}$ need to be fast in MPC and will be used to replace bottleneck functions in $\mathcal{T}$.

The process can be concisely described as:

$$\begin{aligned} \text{Convert:} \quad & \mathcal{S} = \text{MPCFORMER}(\mathcal{T}, \mathcal{D}, \mathcal{A}) \\ \text{Inference:} \quad & y = \text{MPC}_{\mathcal{S}}(X) \end{aligned} \tag{1}$$

4.2 MPCFORMER

MPCFORMER is a two-stage framework as shown in Figure 3. The first stage leverages $\mathcal{A}$ to construct an MPC-friendly Transformer architecture $\mathcal{S}'$ from $\mathcal{T}$. $\mathcal{S}'$ achieves *fast* inference in the MPC system. The second stage applies knowledge distillation(KD) to $\mathcal{S}'$ to learn the output model $\mathcal{S}$. The use of KD let $\mathcal{S}$ preserve the *high performance* of $\mathcal{T}$.

4.2.1 STAGE 1: APPROXIMATION

In the first stage, MPCFORMER replaces bottleneck functions in $\mathcal{T}$ with given $\mathcal{A}$ to construct a fast Transformer architecture $\mathcal{S}'$ (Figure 3). Below we show how we construct $\mathcal{A}$ for our experiments *i.e.,* using the MPC system Knott et al. (2021). In particular, we highlight that our framework is easy-to-adopt by showing how $\mathcal{A}$ can be easily constructed. In §2, we identify the bottleneck to be GeLU and Softmax functions. We thus construct $\mathcal{A}$ for these two functions.

Approximating GeLU. Analysis in Section § 2 shows that multiplication in MPC requires extra communication. Thus, quadratics are the fastest nonlinear activation in MPC. Since GeLU and ReLU functions share similar function values, we simply take the quadratics designed for the ReLU function (Section §3) to approximate the GeLU function: $GeLU(x) \approx 0.125x^2 + 0.25x + 0.5$. We denote this as "Quad".

Approximating Softmax. Prior works in CNNs have developed an MPC-friendly approximation to Softmax functions (Mohassel & Zhang, 2017):

$$\text{softmax}(x) \approx \frac{\text{ReLU}(x)}{\sum \text{ReLU}(x)} \tag{2}$$

We validate that this has a faster inference speed than the Softmax function in our setting (Figure 4). We denote this approximation as "2ReLU". However, this is not yet satisfactory. Analysis in §2

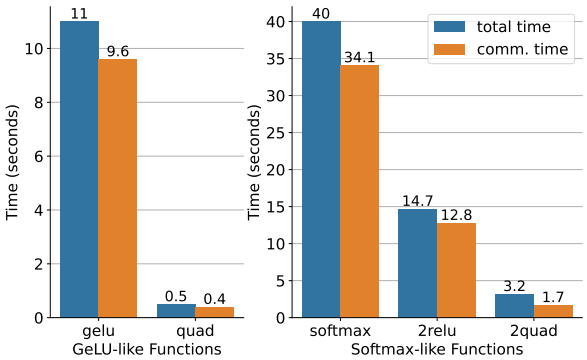

Figure 4: Running time comparison of different approximations in §4.2.1. Blue areas represent the total running time. Orange areas represent the communication time. MPC-friendly approximations greatly reduce both the communication time and the total time. In particular, our proposed "2Quad" is much faster than the original Softmax function, and the previous "2ReLU" approximation.

shows that evaluating the ReLU function requires heavy use of comparison primitives, which is very expensive. Thus, we propose a more aggressive approximation for the softmax by replacing ReLU in Eq. 2 with quadratics:

$$\text{softmax}(x) \approx \frac{(x+c)^2}{\sum(x+c)^2} \tag{3}$$

We denote this as "2Quad". Importantly, "2Quad" and Softmax functions differ a lot by numerical values, while prior works argue that similarity in numerical values is crucial to the model's performance (Chou et al., 2018). We are able to use this aggressive approximation because our next distillation stage is effective enough to bridge the performance gap.

Figure 4 shows the comparison between the running time of the original GeLU and softmax function against their approximations. In particular, 2Quad has a much faster inference speed than 2ReLU.

4.2.2 Stage 2: Distillation

In the second stage, we use KD to make the fast approximated Transformer model $\mathcal{S}'$ performant. The benefits of KD are two-fold. First, it allows us to use more aggressive approximations such as "2Quad", which leads to higher speedups. Second, its data efficiency allows us to effectively learn a good $\mathcal{S}$ with the small downstream datasets. Concretely, we conduct layer-wise distillation to transfer knowledge from the original model $\mathcal{T}$ to $\mathcal{S}'$ by matching representation at the following four positions: (1) the embedding layer, (2) the attention matrix in each Transformer layer, (3) the hidden states after each Transformer layer, and (4) the final prediction layer. These four positions have been shown to store meaningful information in previous works (Hinton et al., 2015; Jiao et al., 2019; Clark et al., 2019). We use the Mean Square Error (MSE) loss to match the representations between $\mathcal{T}$ and $\mathcal{S}$ for all positions. We follow the learning procedure of Jiao et al. (2019) to first distill the embedding and Transformer layers (including the attention matrix and the hidden states) and then distill the prediction layer.

Student initialization. An important component of knowledge distillation is the initialization of the student model (Sanh et al., 2019). Taking the advantage that $\mathcal{S}'$ and $\mathcal{T}$ share the same architecture, we initialize $\mathcal{S}'$ using weights in $\mathcal{T}$. We find that this outperforms random weight initialization, especially on smaller datasets (§5.3).

5 Experiments

We design the MPCFormer framework to be compatible with many MPC-friendly approximations and trained Transformer models, so that model providers can conveniently plug in MPC-friendly approximations based on their MPC systems. Thus, we evaluate MPCFormer with different MPC-friendly approximations under (1) Different datasets (§ 5.1), and (2) Different models (especially larger models) (§5.2). In the ablation study, we study (1) the effect of student initialization and, (2) the effect of the number of training examples in the distillation stage.

Experimental setup. We use two P3.2x AWS instances to simulate the inference service scenarios (one P3.2x for the model provider, and one for the user). Each instance is equipped with one Tesla V100 GPU, and 10GbE Ethernet bandwidth. We place instances in the same placement group to guarantee a 10GbE bandwidth in AWS. Time breakdown is measured with Crypten, which implements secret sharing with semi-honest adversaries assumption (Section §2) (Knott et al., 2021). We train and evaluate models based on HuggingFace(Wolf et al., 2019). In particular, we find that the implementation of 2Quad requires careful use of HuggingFace source code (§A.2).

Baselines. We identify three important properties during Transformer inference in § 1: speed, performance, and privacy. In our workflow, privacy has been guaranteed by using MPC systems. Thus, we evaluate MPCFORMER by the other two aspects: speed and performance. Concretely, we require $\mathcal{S}$ to run faster than $\mathcal{T}$ while matching the performance of $\mathcal{T}$. To the best of our knowledge, we are among the first to tackle Transformer inference in MPC. Thus, we seek a baseline from CNN literature, which is strong and general enough so we can easily adapt it to the Transformer case. In particular, we choose the training strategy in (Chou et al., 2018) and denote it as MPCFORMER$_{\text{w/o}\{\text{d}\}}$. MPCFORMER$_{\text{w/o}\{\text{d}\}}$ also constructs the approximated model $\mathcal{S}'$ but trains $\mathcal{S}'$ on $\mathcal{D}$ with the task-specific objective, *i.e.,* without distillation. We note that $\mathcal{S}'$ is initialized with weights in $\mathcal{T}$, *i.e.,* with different functions, whose effect has not been studied. We thus propose a second baseline MPCFORMER$_{\text{w/o}\{\text{p,d}\}}$, which trains $\mathcal{S}'$ on $\mathcal{D}$ without distillation, and random weight initialization. Below we compare the performance of MPCFORMER with MPCFORMER$_{\text{w/o}\{\text{p,d}\}}$ and MPCFORMER$_{\text{w/o}\{\text{d}\}}$ with the same speedups.

We denote the output model of our framework with BERT$_{\text{BASE}}$, Roberta-base, and BERT$_{\text{LARGE}}$ as MPCBert-B, MPCRoberta-B, and MPCBert-L for short.

5.1 COMPARISON WITH BASELINES ON DIFFERENT BENCHMARKS

Settings. In this section, we evaluate our MPCFormer framework with different approximations and compare it with baselines on the IMDb dataset and the GLUE benchmark (Maas et al., 2011; Wang et al., 2018). For all experiments in this section, we use BERT$_{\text{BASE}}$ as the base model. According to the dataset statistics, we use a sequence length of 512 for the IMDb dataset and a sequence length of 128 for GLUE datasets. We note that a longer sequence length generally reflects a higher speedup because the Softmax functions can be sped up more. Baselines are trained with learning rates tuned from 1e-6, 5e-6, 1e-5, and 1e-4, the number of epochs from 10, 30, and 100, the batch size 32 for IMDB, batch sizes 64 and 256 for GLUE. MPCBert-B is trained with learning rate 5e-5 for embedding and Transformer layer distillation, and 1e-5 for prediction layer distillation. Further details on hyper-parameters tuning can be found in A.3.

We show the accuracy and speedup on the IMDb dataset in Table 2. MPCBert-B achieves 5.26× speedup with "Quad+2Quad" approximation with almost no accuracy drop. In addition, we note that this holds for not only the fastest "Quad+2Quad" approximation but other approximations ranging from 1.24× to 2.65 × speedups. Baselines have inferior performance for all approximations. For example, both baselines have at least an accuracy drop of 6.8% with 5.26 × speedup. Interestingly, MPCBert-B$_{\text{w/o}\{\text{d}\}}$ has moderate accuracy drop for "GeLU+2ReLU" approximation with 1.76× speedup. However, it does not preserve accuracy with other approximations. In contrast, our framework consistently preserves accuracy with various kinds of approximation.

To validate the observation with more datasets, We evaluate MPCBert-B on the Glue benchmark (8 datasets) (Wang et al., 2018). As shown in Table 3, MPCBert-B achieves 1.93× speedup with 98% performance of BERT$_{\text{BASE}}$ on the GLUE benchmark, and 2.2× speedup with 97% performance of BERT$_{\text{BASE}}$. Both baselines introduce severe performance drop, i.e., 19.5 average score drop for MPCBert-B$_{\text{w/o}\{\text{d}\}}$ and 26.2% average score drop for MPCBert-B$_{\text{w/o}\{\text{p,d}\}}$ for 1.93× speedup. Interestingly, we observe that the baseline MPCBert-B$_{\text{w/o}\{\text{d}\}}$ consistently outperforms MPCBert-B$_{\text{w/o}\{\text{p,d}\}}$. This indicates that using weights in $\mathcal{T}$ as initialization benefits the training of $\mathcal{S}'$ with a task-specific objective, using a 12-layer Transformer backbone. Additionally, we evaluate MPCFORMER with more approximations using a subset of the GLUE benchmark. Results are shown in the right part of Table 2. We observe similar patterns as in the IMDb dataset. The baseline MPCBert-B$_{\text{w/o}\{\text{d}\}}$ performs well in "GeLU+2Quad' and "GeLU+2ReLU" approximations, but MPCFORMER achieves high ML performance consistently under all approximations.

Table 2: Performance and speedup on the IMDB dataset and a part of the GLUE benchmark (QNLI, CoLA and RTE) with different approximations and $\text{BERT}_{\text{BASE}}$ as the backbone. The original model $\mathcal{T}$ is denoted with "*". "p" stands for using weights in $\mathcal{T}$ as initialization, "d" stands for applying knowledge distillation with $\mathcal{T}$ as the teacher.

Method	**Approximation**	**IMDb**		**GLUE**	
		Speedup	**Accuracy**	**Speedup**	**Avg. Score**
Bert-B*	GeLU+Softmax	1×	94.1	1×	73.1
MPCBert-$\text{B}_{\text{w/o}\{\text{p,d}\}}$	Quad+Softmax	1.24×	87.5	1.13×	40.8
MPCBert-$\text{B}_{\text{w/o}\{\text{d}\}}$			87.5		43.0
MPCBert-B			**94.0**		**72.6**
MPCBert-$\text{B}_{\text{w/o}\{\text{p,d}\}}$	GeLU+2ReLU	1.76×	86.1	1.40×	39.6
MPCBert-$\text{B}_{\text{w/o}\{\text{d}\}}$			93.8		71.8
MPCBert-B			**94.0**		**72.0**
MPCBert-$\text{B}_{\text{w/o}\{\text{p,d}\}}$	Quad+2ReLU	2.61×	85.8	1.93×	43.5
MPCBert-$\text{B}_{\text{w/o}\{\text{d}\}}$			86.8		48.2
MPCBert-B			**94.0**		**69.8**
MPCBert-$\text{B}_{\text{w/o}\{\text{p,d}\}}$	GeLU+2Quad	2.65×	87.3	1.55 ×	39.6
MPCBert-$\text{B}_{\text{w/o}\{\text{d}\}}$			90.6		69.7
MPCBert-B			**94.0**		**71.0**
MPCBert-$\text{B}_{\text{w/o}\{\text{p,d}\}}$	**Quad+2Quad**	**5.26**×	87.8	**2.20**×	40.7
MPCBert-$\text{B}_{\text{w/o}\{\text{d}\}}$			87.3		40.8
MPCBert-B			**93.9**		**68.4**

Table 3: Performance on Glue benchmark with $\text{BERT}_{\text{BASE}}$ as the backbone. F1 score is reported for QQP and MRPC. Average Pearson and Spearman correlation is reported for STS-B. Matthews correlation is reported for CoLA. Accuracy is reported for other datasets.

Method	**Approx.**	MNLI 393k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	**Avg.**	**Speed -up**
Bert-B*(GeLU+Softmax)		84.7/85.0	88.1	91.7	93.1	57.8	89.1	90.3	69.7	82.8	1×
MPCBert-$\text{B}_{\text{w/o }\{\text{p,d}\}}$	**Quad**	62.1/61.3	74.6	61.8	80.7	13.4	23.1	81.2	55.2	56.6	
MPCBert-$\text{B}_{\text{w/o }\{\text{d}\}}$	**+**	73.1/72.5	82.9	75.5	83.4	16.4	41.3	81.2	52.7	63.3	**1.93**×
MPCBert-B	**2ReLU**	**85.0/85.3**	**87.8**	**91.2**	**92.0**	**54.0**	**85.7**	**88.9**	**64.3**	**81.1**	
MPCBert-$\text{B}_{\text{w/o }\{\text{p,d}\}}$	**Quad**	63.5/62.4	78.6	59.8	81.1	9.9	19.5	81.4	52.7	55.7	
MPCBert-$\text{B}_{\text{w/o }\{\text{d}\}}$	**+**	70.6/70.5	83.4	69.8	83.3	0	36.1	81.2	52.7	60.9	**2.2**×
MPCBert-B	**2Quad**	**84.9/85.1**	**88.1**	**90.6**	**92.0**	**52.6**	**80.3**	**88.7**	**64.9**	**80.3**	

5.2 More comparisons with different models

We evaluate MPCFormer with trained Transformer models other than $\text{BERT}_{\text{BASE}}$, *i.e.,* $\text{ROBERTA}_{\text{BASE}}$ model (12 layers), and in particular a larger $\text{BERT}_{\text{LARGE}}$ model (24 layers) (Liu et al., 2019). Results are shown in Table 4. MPCRoberta-B preserves 98% average score of the original model $\text{ROBERTA}_{\text{BASE}}$, and outperforms baselines by large margins. Comparing the performance of baselines, we again observe that initialization with weights in $\mathcal{T}$ helps training with $\mathcal{S}'$, *i.e.,* MPCRoberta-$\text{B}_{\text{w/o }\{\text{d}\}}$ performs better than MPCRoberta-$\text{B}_{\text{w/o }\{\text{p,d}\}}$.

To show our method can scale to different model sizes, we evaluate MPCFormer with a larger model $\text{BERT}_{\text{LARGE}}$ (24-layer). On the IMDb dataset, $\text{BERT}_{\text{BASE}}$ achieves 95.0 % accuracy. MPCBert-L achieves 5.9× speedup with 94.5% accuracy, while baselines MPCBert-$\text{L}_{\text{w/o }\{\text{p,d}\}}$ achieves 87.1% accuracy, and MPCBert-$\text{L}_{\text{w/o }\{\text{d}\}}$ achieves 50.0% accuracy. We further select four datasets from the GLUE benchmark, where Bert-L* noticeably outperforms Bert-B*(MNLI, QNLI, CoLA, and RTE). Compared with Bert-B* in table 3, Bert-L* increases the average score from 82.8 to 85.3. MPCFormer increases the average score from 81.5 to 84.6. This indicates that MPCFormer can scale with the size of $\mathcal{T}$. On the other hand, baselines do not scale with the original model: MPCBert-$\text{L}_{\text{w/o }\{\text{p,d}\}}$ decreases the average score from 60.1 to 59.3; MPCBert-$\text{L}_{\text{w/o }\{\text{d}\}}$ decreases the average score from 68.5 to 43.5. In particular, we observe that initializing $\mathcal{S}'$ with weights in $\mathcal{T}$ without distillation harms performance when the model is larger.

Table 4: The performance on a subset of Glue benchmark with Roberta-base backbone (denoted as "MPCRoberta-B"). MPCRoberta-B and baselines use "Quad+2Quad" approximations with 2.1 × speedup. MPCBert-L and baselines use "Quad+2ReLU" approximations with 2.0 × speedup.

	MNLI-m	**MNLI-mm**	**QNLI**	**RTE**	**Avg**	**Speedup**
Roberta-B*	87.4	87.0	92.6	76.5	85.8	1×
MPCRoberta-$B_{w/o\ \{p,d\}}$	58.0	58.1	69.0	52.7	59.5	
MPCRoberta-$B_{w/o\ \{d\}}$	73.1	72.7	81.6	52.7	70.0	**2.1**×
MPCRoberta-B	**86.5**	**86.9**	**92.2**	**72.2**	**84.5**	
Bert-L*	86.7	86.6	92.7	75.1	85.3	1×
MPCBert-$L_{w/o\ \{p,d\}}$	62.3	62.3	60.0	52.7	59.3	
MPCBert-$L_{w/o\ \{d\}}$	35.4	35.2	50.5	52.7	43.5	**2.0**×
MPCBert-L	**86.5**	**86.7**	**92.8**	**72.2**	**84.6**	

5.3 Ablation Study

The first question we study is the effect of different student initialization. This is of interest as we do not have a general knowledge of whether initializing with weights in $\mathcal{T}$ will still benefit the training of $\mathcal{S}'$ after aggressive approximations. We design an experiment with random initialization (denoted as MPCBert-B_r in the table). We train MPCBert-B_r with 10× more epochs than MPCBert-B_r and confirm that its distillation objective has converged. We observe that on larger datasets(QNLI and SST-2), the gap between using different initialization is small, but on smaller datasets(STS-B, MRPC, and RTE), initializing with the weights in $\mathcal{T}$ is better.

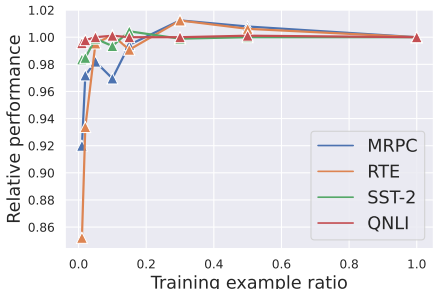

Figure 5: Ratio of training examples versus performance (normalized by performance with ratio = 1.0) on MRPC, RTE, QNLI, and SST-2.

The second question we study is the effect of the number of training examples in the distillation stage. We perform studies on two small (RTE, MRPC) and two medium (SST-2, QNLI) datasets in the GLUE benchmark (Figure 5). We find that roughly 5% of the small datasets and 2% of the medium datasets provides enough data to learn a good $\mathcal{S}$. This shows that KD in our setting is efficient enough to learn a good $\mathcal{S}$ in downstream tasks (using GLUE datasets as representatives).

Table 5: Student model initialized with weights in $\mathcal{T}$ versus with random weights. Result for MPCBert-B_r is tuned with embedding and Transformer layer distillation learning rates from 5e-5 and 3e-5. Results for both are obtained with the "Quad+2Quad" approximation.

	QNLI	SST-2	STS-B	MRPC	RTE	Avg
MPCBert-B	**90.6**	**92.0**	**80.8**	**88.7**	**64.9**	**84.0**
MPCBert-B_r	**90.6**	90.0	60.0	81.2	58.5	76.1

5.4 Limitation and Future Direction

We recognize two limitations in our paper. First, our speedups and performance are tested on a single MPC system. We leave theoretical analysis or empirical study on more MPC systems as future work. Second, in our design, $\mathcal{T}$ and $\mathcal{S}$ only differ by functions, *i.e.,* they have the same model size. We leave extension to our framework to a smaller student model as future work.

6 Conclusion

In this paper, we propose a framework to achieve fast and performant private Transformer model inference with MPC. Evaluations show that it is compatible with various MPC-friendly approximations and trained Transformer models. We suggest two directions of interest: (1) Theoretical or empirical analysis on more MPC systems, and (2) extension on the problem formulation to allow a smaller size of $\mathcal{S}$.

REFERENCES

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Donald Beaver. Efficient multiparty protocols using circuit randomization. In *Annual International Cryptology Conference*, pp. 420–432. Springer, 1991.

Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

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.

Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D Manning. What does bert look at? an analysis of bert's attention. *arXiv preprint arXiv:1906.04341*, 2019.

Kevin Clark, Minh-Thang Luong, Quoc V Le, and Christopher D Manning. Electra: Pre-training text encoders as discriminators rather than generators. *arXiv preprint arXiv:2003.10555*, 2020.

Ronald Cramer, Ivan Damgård, and Yuval Ishai. Share conversion, pseudorandom secret-sharing and applications to secure computation. In *Theory of Cryptography Conference*, pp. 342–362. Springer, 2005.

Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Annual Cryptology Conference*, pp. 643–662. Springer, 2012.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

David Evans, Vladimir Kolesnikov, Mike Rosulek, et al. A pragmatic introduction to secure multi-party computation. *Foundations and Trends® in Privacy and Security*, 2(2-3):70–246, 2018.

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. Codebert: A pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155*, 2020.

Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game, or a completeness theorem for protocols with honest majority. In *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*, pp. 307–328. 2019.

Geoffrey Hinton, Oriol Vinyals, Jeff Dean, et al. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2(7), 2015.

Geoffrey E. Hinton, Simon Osindero, and Yee Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18:1527–1554, 2006.

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. Tinybert: Distilling bert for natural language understanding. *arXiv preprint arXiv:1909.10351*, 2019.

Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. {GAZELLE}: A low latency framework for secure neural network inference. In *27th USENIX Security Symposium (USENIX Security 18)*, pp. 1651–1669, 2018.

Marcel Keller, Emmanuela Orsini, and Peter Scholl. Mascot: faster malicious arithmetic secure computation with oblivious transfer. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pp. 830–842, 2016.

Brian Knott, Shobha Venkataraman, Awni Hannun, Shubho Sengupta, Mark Ibrahim, and Laurens van der Maaten. Crypten: Secure multi-party computation meets machine learning. *Advances in Neural Information Processing Systems*, 34:4961–4973, 2021.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019.

Jian Liu, Mika Juuti, Yao Lu, and Nadarajah Asokan. Oblivious neural network predictions via minionn transformations. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, pp. 619–631, 2017.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 10012–10022, 2021.

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pp. 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/P11-1015`.

Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. Delphi: A cryptographic inference service for neural networks. In *29th USENIX Security Symposium (USENIX Security 20)*, pp. 2505–2522, 2020.

Payman Mohassel and Peter Rindal. Aby3: A mixed protocol framework for machine learning. In *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, pp. 35–52, 2018.

Payman Mohassel and Yupeng Zhang. Secureml: A system for scalable privacy-preserving machine learning. In *2017 IEEE symposium on security and privacy (SP)*, pp. 19–38. IEEE, 2017.

Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *International conference on the theory and applications of cryptographic techniques*, pp. 223–238. Springer, 1999.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pp. 8748–8763. PMLR, 2021.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, Peter J Liu, et al. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(140):1–67, 2020.

M Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M Songhori, Thomas Schneider, and Farinaz Koushanfar. Chameleon: A hybrid secure computation framework for machine learning applications. In *Proceedings of the 2018 on Asia conference on computer and communications security*, pp. 707–721, 2018.

Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

Gilad Sharir, Asaf Noy, and Lihi Zelnik-Manor. An image is worth 16x16 words, what is a video worth? *arXiv preprint arXiv:2103.13915*, 2021.

Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *International Conference on Machine Learning*, pp. 10347–10357. PMLR, 2021.

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.

Sameer Wagh, Divya Gupta, and Nishanth Chandran. Securenn: 3-party secure computation for neural network training. *Proc. Priv. Enhancing Technol.*, 2019(3):26–49, 2019.

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. *arXiv preprint arXiv:1804.07461*, 2018.

Yongqin Wang, G Edward Suh, Wenjie Xiong, Benjamin Lefaudeux, Brian Knott, Murali Annavaram, and Hsien-Hsin S Lee. Characterization of mpc-based private inference for transformer-based models. In *2022 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 187–197. IEEE, 2022.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*, 2019.

Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10(2):1–19, 2019a.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in neural information processing systems*, 32, 2019b.

A APPENDIX

A.1 A CONCRETE SYSTEM IMPLEMENTATION OF MPC: CRYPTEN

In this section, we provide how a concrete MPC system (Crypten) implements primitives and functions for Transformer models in detail (Knott et al., 2021). We provide a portion of details here to help describe the complexity of Transformer inference in MPC. A more complete system overview and privacy proof are available in the Crypten paper.

Threat model. Crypten follows Evans et al. (2018) to assume that parties are semi-honest. Under this assumption, parties are *honest* that they will follow the system protocols. However, each party is also *curious* (i.e., semi-honest), meaning it will try to infer the information about others' data based on the values it receives.

Secret shares Crypten uses secret shares to implement private computation. A floating point value x_f is first scaled to an integer x[4], then secretly shared with both parties. Secret shares are of type *arithmetic* or *binary*. The arithmetic secret shares $[x] = \{[x]_1, [x]_2\}$ is a set of two numbers, where the first party holds $[x]_1$, and the second holds $[x]_2$. They are constructed with a pair of zero-sum random maskings (Cramer et al., 2005) so that $[x]_1 + [x]_2 = x$. Binary shares $\langle x \rangle$ are formed by arithmetic secret shares of bits in x, so that the bitwise xor $\langle x \rangle_1 \oplus \langle x \rangle_1 = x$.

Primitives and functions In arithmetic secret shares, to privately evaluate addition $[x] + [y]$, both parties simply compute: $[x]_i + [y]_i$ individually. Multiplication $[x][y]$ is evaluated by using a Beaver triple generated off-line $([c], [a], [b])$ where $c = ab$ (Beaver, 1991). Both parties compute and reveal intermediate values $[\epsilon] = [x] - [a]$ and $[\delta] = [y] - [b]$. The final result is computed by $[x][y] = [c] + \epsilon[b] + [a]\delta + \epsilon\delta$. Linear functions such as matrix multiplication can be evaluated by using additions and multiplications. Non-linear functions are evaluated by numerical approximations using additions and multiplications, such as Taylor expansion.

Comparison requires conversions between arithmetic and binary secret shares. Conversion from $[x]$ to $\langle x \rangle$ first creates binary secret shares $\langle [x]_i \rangle$ from arithmetic secret share $[x]_i$ ($i = 0, 1$), then computes $\langle x \rangle = \langle [x]_1 \rangle + \langle [x]_2 \rangle$ using an adder circuit. Conversion from $\langle x \rangle$ to $[x]$ is done by: $[x] = \sum_{b=1}^{B} 2^b [\langle x \rangle^{(b)}]$, where $\langle x \rangle^{(b)}$ is the bth bit of $\langle x \rangle$. The comparison function $[z < 0]$ is then evaluated by: (1) convert $[z]$ to $\langle z \rangle$ (2) compute the sign bit $\langle b \rangle = \langle z \rangle >> (L - 1)$. (3) Convert $\langle b \rangle$ to $[b]$.

We study on the standard setting where each tensor is represented in 64 bits (i.e. L=64). Each multiplication requires one round of communication for revealing the intermediate values ϵ, δ. Each conversion from $[x]$ to $\langle x \rangle$ requires $\log_2 L = 6$ rounds of communications for the adder circuit; each conversion from $\langle x \rangle$ to $[x]$ requires one round for generating $[\langle x \rangle^{(b)}]$. Thus, each comparison requires 7 rounds of communication. Each $\max(\cdot)$ between N elements requires $\mathcal{O}(\log_2(N))$ rounds of communications, assuming a tree-reduction algorithm.

We provide a simple addition example here. The scaling factor and ring size $\mathcal{Q}$ are set to small for ease of understanding.

Table 6: Example of secure addition computation. Suppose m is the actual message, then each party holds a share of m such that: $[m]_1 + [m]_2 = m$.

Action	Party 1	Party 2	Note
Declare x	$x = 1$	$x =$ random	x provided by party 1
Generate a secret-sharing mask for x	$[z_x]_1 = -4$	$[z_x]_2 = 4$	sum to 0
secret share x	$[x]_1 = x + [z_x]_1 = -3$	$[x]_2 = [z_x]_2 = 4$	sum to x_1
Declare y	$y =$ random	$y = 2$	y provided by party 2
Generate a secret-sharing mask for y	$[z_y]_1 = 50$	$[z_y]_2 = -50$	sum to 0
secret share y	$[y]_1 = [z_y]_1 = 50$	$[y]_2 = y + [z_y]_2 = -48$	sum to y_2
Compute $x + y$	$[x + y]_i = [x]_1 + [y]_1 = 47$	$[x + y]_2 = [x]_2 + [y]_2 = -44$	
Reveal $x + y$	$x + y = [x + y]_1 + [x + y]_2 = 3$	$x + y = [x + y]_1 + [x + y]_2 = 3$	both get correct results

[4] $x \in \mathcal{Z}/\mathcal{Q}\mathcal{Z}$ is required for privacy protocols, where $\mathcal{Z}/\mathcal{Q}\mathcal{Z}$ is a ring with $\mathcal{Q}$ elements.

A.2 2QUAD IMPLEMENTATION DETAILS

We note that, implementing "2Quad" to replace the softmax requires attention to the effect brought by the masking operation. For example, the default implementation by Huggingface Wolf et al. (2019) would result in an exploding problem due to masking. Therefore, we would need to do a different version of the implementation of masking. We describe it in detail below.

The default attention implementation by Huggingface is

$$\begin{aligned} Attention(Q,K,V) &= softmax(\frac{QK^T}{\sqrt{d_k}} + M_{\{0,-inf\}})V \\ &= \frac{e^{\left(\frac{QK^T}{\sqrt{d_k}} + M_{\{0,-inf\}}\right)}}{\sum_{j=1}^{K} e^{\left(\frac{QK^T}{\sqrt{d_k}} + M_{\{0,-inf\}}\right)_j}} V. \end{aligned}$$

If we directly replace the e^x with $(x+c)^2$ as in 2Quad approximation, where $x = \frac{QK^T}{\sqrt{d_k}} + M_{\{0,-inf\}}$ will explode when being masked, causing a problem in the forward pass. To solve this problem, we could simply change the implementation of masking from "adding a zero or negative infinite number in the exponent" to "multiplying one or zero to the exponential function". That is,

$$\begin{aligned} Attention(Q,K,V) &= \frac{e^{\left(\frac{QK^T}{\sqrt{d_k}}\right)} \odot M_{\{1,0\}}}{\sum_{j=1}^{K} e^{\left(\frac{QK^T}{\sqrt{d_k}}\right)_j} \odot M_{\{1,0\}}} V \\ &\rightarrow \frac{\left(\frac{QK^T}{\sqrt{d_k}} + c\right)^2 \odot M_{\{1,0\}}}{\sum_{j=1}^{K} \left(\frac{QK^T}{\sqrt{d_k}} + c\right)_j^2 \odot M_{\{1,0\}}} V. \end{aligned}$$

It's just a different implementation of the same masking purpose but avoids exploding at the masking positions.

In our experiments, we empirically tried $c = 5$ and it worked pretty well, indicating the choice of the constant c could be flexible.

A.3 HYPER-PARAMETER CHOICE

For baselines, We study the effect of hyper-parameters by running a grid search over the STS-B dataset [5], with learning rate from [1e-6, 5e-6, 1e-5, 5e-5, 1e-4, 5e-4], batch size from [256, 128, 64, 32, 16], epoch from [3, 10, 30, 50, 80, 100, 200]. We show the grid search results with $\text{BERT}_{\text{BASE}}$ in figure 6, 7, and a smaller grid search for $\text{BERT}_{\text{Large}}$ and $\text{ROBERTA}_{\text{BASE}}$ in Figure 8, 9. We empirically discover that the learning rates from 1e-6, 5e-6, 1e-5, batch size from 64 and 256, epoch from 10, 100 give good performance. To let baselines explore more hyper-parameters, we use learning rate from [1e-6, 5e-6, 1e-5, 1e-4], batch size from [64, 256], epochs from [10, 30, 100] for all Glue datasets. Since we use sequence length 512 for IMDB dataset, we use batch size 32 to fit into our 16GB Tesla V100 GPU. We also empirically discover that (1) $\text{MPCBert-B}_{\text{w/o\{d\}}}$ (best 0.43) can not scale up when the base model scales to $\text{BERT}_{\text{Large}}$ *i.e.,* $\text{MPCBert-L}_{\text{w/o\{d\}}}$ (best 0.08). (2) baseline benefits from using the pre-trained weights, *i.e.,* $\text{MPCBert-B}_{\text{w/o\{d\}}}$ (best 0.42) performs better than $\text{MPCBert-B}_{\text{w/o\{p, d\}}}$ (best 0.23). (3) $\text{MPCFormer}_{\text{w/o\{d\}}}$ benefits when the base model becomes better, *i.e.,* $\text{MPCRoberta-B}_{\text{w/o\{d\}}}$ (best 0.62) performs better than $\text{MPCBert-B}_{\text{w/o\{d\}}}$ (best 0.42).

For MPCFORMER, we decide the number of epochs according to the MSE loss for embedding and Transformer layer distillation, 5 epochs for prediction layer distillation, and batch size 8 for small datasets (CoLA, MRPC, RTE) and 32 for larger ones (MNLI, QQP, SST2, STS-B). We minimize the hyper-parameter tuning for MPCFORMER, since we would like the performance to be an expectation for future researchers using MPCFORMER, who prefer not to tune hyper-parameters. Specifically,

[5] We select STS-B because it is a regression task, where performance varies in a large range.

we use 5 epochs for MNLI, 5 epochs for QQP, 10 epochs for QNLI, 10 epochs for SST-2, 20 epochs for MRPC, 30 epochs for IMDB 50 epochs for STS-B, 50 epochs for CoLA, 50 epoches for RTE, for the embedding and Transformer layer distillation stage.

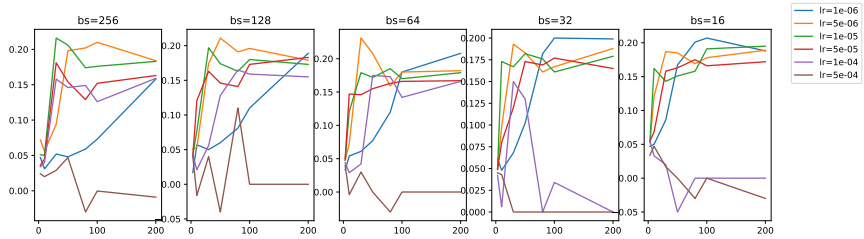

Figure 6: Grid search results for MPCBert-$B_{w/o\{p,d\}}$ on STS-B dataset. X-axis is number of epochs, and Y-axis is correlation (unscaled).

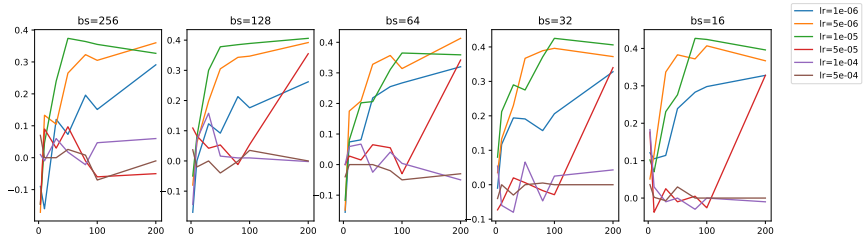

Figure 7: Grid search results for MPCBert-$B_{w/o\{d\}}$ on STS-B dataset. X-axis is number of epochs, and Y-axis is correlation (unscaled).

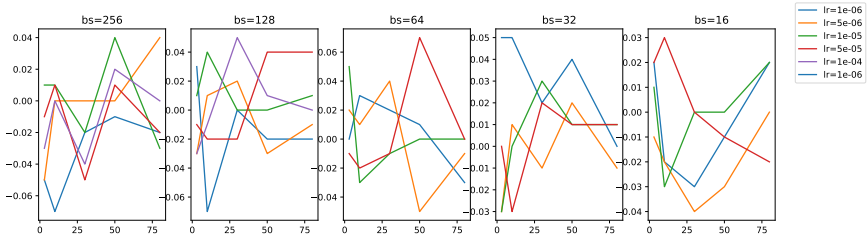

Figure 8: Grid search results for MPCBert-$L_{w/o\{d\}}$ on STS-B dataset. X-axis is number of epochs, and Y-axis is correlation (unscaled).

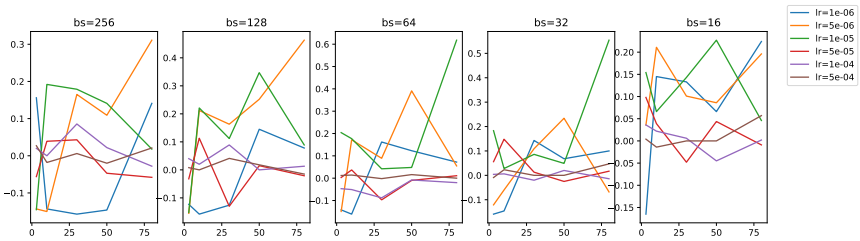

Figure 9: Grid search results for MPCRoberta-$B_{w/o\{d\}}$ on STS-B dataset. X-axis is number of epochs, and Y-axis is correlation (unscaled).