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 b^{th} 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	$x = 1$	$x = \text{random}$	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	$y = \text{random}$	$y = 2$	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.

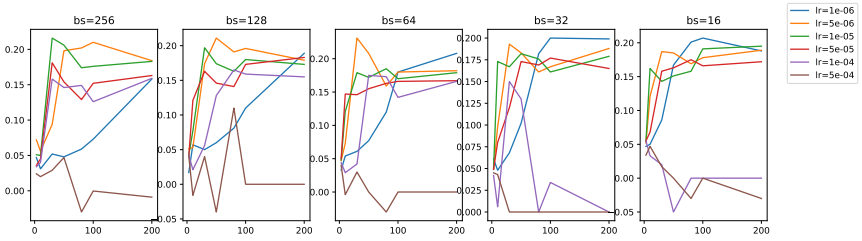

Figure 6: Grid search results for MPCBert-$B_{w/o\{p,d\}}$ on STS-B dataset. X-axis is number of epochs, and Y-axis is correlation (unscaled).

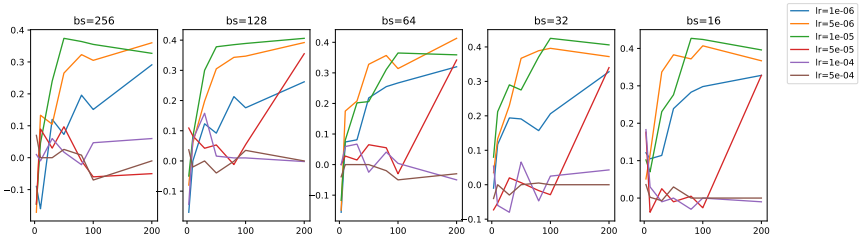

Figure 7: Grid search results for MPCBert-$B_{w/o\{d\}}$ on STS-B dataset. X-axis is number of epochs, and Y-axis is correlation (unscaled).

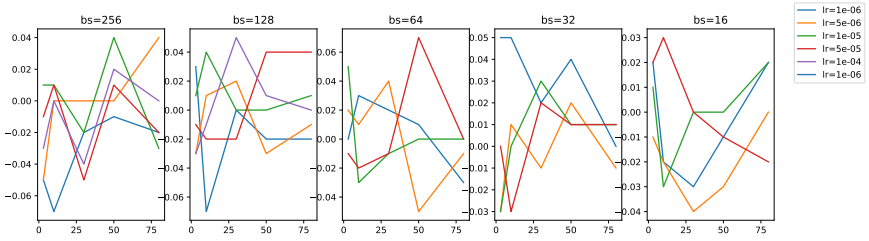

Figure 8: Grid search results for MPCBert-$L_{w/o\{d\}}$ on STS-B dataset. X-axis is number of epochs, and Y-axis is correlation (unscaled).

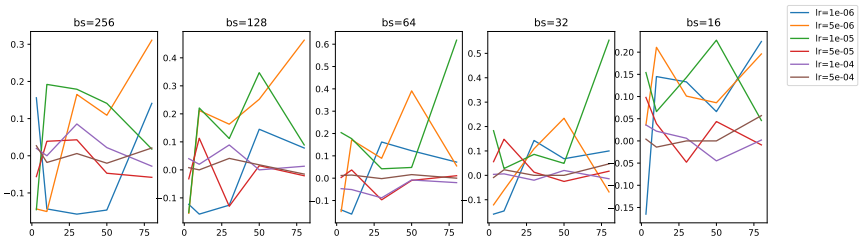

Figure 9: Grid search results for MPCRoberta-$B_{w/o\{d\}}$ on STS-B dataset. X-axis is number of epochs, and Y-axis is correlation (unscaled).