NETADAPT: Network-Adaptive Hybrid Protocol Assignment for PPML

Yuntian Chen, Tianpei Lu, Zhanyong Tang, Bingsheng Zhang, Zhiyuan Ning, Zhiying Shi and Kui Ren

***Abstract*—The widespread use of machine learning on sensitive data makes Privacy-Preserving Machine Learning (PPML) essential for data confidentiality. Current state-of-the-art PPML systems employ hybrid protocol designs to evaluate various operators to achieve better performance; yet, existing hybrid approaches adopt fixed protocol assignments without considering the deployment setting, resulting in inefficiencies across diverse network environments, such as LANs and WANs. To address this, we introduce NETADAPT, a cost-model-driven framework that automatically assigns FHE and MPC protocols to optimize inference efficiency under varying network conditions, by designing a predictive cost model based on the dialect of the Multi-Level Intermediate Representation (MLIR)'s Tensor Operator Set Architecture (TOSA), and an ILP-based solver for managing the cross-protocol conversion while dynamically updating FHE costs based on homomorphic multiplication depth. Experimental results demonstrate that NETADAPT delivers $6.68\times$ to $12.92\times$ improvements in inference running time compared to state-of-the-art solutions, allowing network-agnostic PPML.**

***Index Terms*—Privacy-Preserving Machine Learning, Secure Multi-Party Computation, Fully Homomorphic Encryption, Cost Modeling.**

I. INTRODUCTION

THE widespread adoption of machine learning across domains such as healthcare, finance, and recommendation systems [1], [2], [3] has intensified concerns about data privacy, particularly when handling sensitive personal information [4]. To address these concerns, Privacy-Preserving Machine Learning (PPML) [5], [6], [7], [8], [9] has emerged as a critical field, leveraging techniques like Secure Multi-Party Computation (MPC) [10] and Fully Homomorphic Encryption (FHE) [11], [12], [13]. In a nutshell, MPC and FHE enable the evaluation of a function while keeping the inputs private, without data leakage.

In PPML applications, particularly for complex models like Transformers [14], computations involve both large-scale linear operations (e.g., matrix multiplications) and high-complexity non-linear operations (e.g., SoftMax, Tanh, ReLU) [15]. To handle the wide variety of operations in modern PPML workloads, prior studies—such as Cheetah [16] have shown that using a hybrid FHE–MPC paradigm achieves lower latency than using a single cryptographic primitive. By taking advantage of the complementary strengths of FHE and MPC for different types of operations, these hybrid designs achieve out-performance. Following this idea, recent work has introduced manually designed protocol combinations to improve the efficiency of PPML systems further. Examples include ABY [17], ABY3 [18], CostCo [19], Silph [20], FIT [21], [22], THEX [23], Nimbus [24], and PrivCirNet [25], which offer improvements when dealing with large computation workloads.

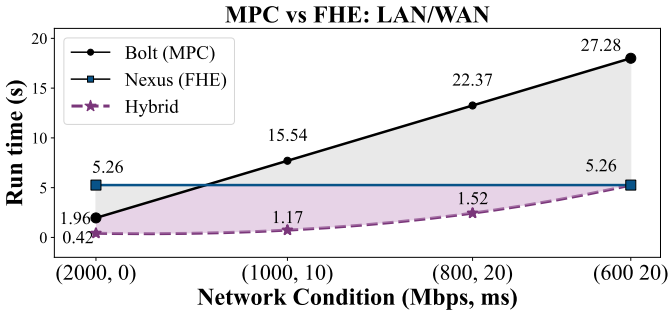


Fig. 1: Comparison of the performance of a single technology in different deployment environments for SoftMax (from 2000 Mbps bandwidth, 0 ms latency to 600 Mbps bandwidth, 20 ms latency), the unit for runtime is seconds. The " —●— " indicates the use of MPC technology, while the " —■— " indicates the use of FHE technology.

However, current hybrid approaches are still limited by a major drawback [26]. Most approaches rely on manually defined rules, treating complex composite operators as indivisible units and assigning a fixed protocol to each, which limits the ability to optimize protocol selection across the entire model. Specifically, BOLT [27] and BumbleBee [28] simply classify operators as follows: linear operations (such as matrix multiplication) are always handled by FHE, while nonlinear operations (such as activation functions) are always handled by MPC. Yet FHE and MPC have very different communication requirements, and these rigid, operator-specific protocol assignments are difficult to adapt to network conditions (e.g., LAN vs. WAN), limiting the overall performance of PPML systems.

These limitations become evident when we examine the performance of specific operators under different network conditions. For instance, the matrix multiplication, CryptFlow2 [29], which leverages MPC-based techniques, achieves 2.56 s in (2 Gbps bandwidth, 0 ms latency) environments and

Yuntian Chen, Zhiying Shi, Zhiyuan Ning, and Zhanyong Tang are with the Xi'an Key Laboratory of Advanced Computing and Software Security, with the College of Computer Science, Northwest University, Xi'an, China.

Tianpei Lu, Bingsheng Zhang and Kui Ren are with the State Key Laboratory of Blockchain and Data Security, Zhejiang University, Hangzhou, China.

0000–0000/00$00.00 © 2021 IEEE

significantly outperforms the FHE-based Nexus [30] (13.10 s), as MPC's lower computational complexity is advantageous when network latency is minimal. Nevertheless, CryptFlow2's performance degrades substantially in (400 Mbps bandwidth, 80 ms latency) settings (25.68 s), where increased latency and limited bandwidth amplify communication overhead, making it less efficient than Nexus. Similarly, for non-linear operations such as SoftMax, Nexus approximates the function using FHE, while BOLT [27] relies on MPC. As illustrated in Figure 1, BOLT outperforms Nexus in (2 Gbps bandwidth, 0 ms latency) environments. However, under reduced bandwidth and higher latency, its frequent MPC communications significantly increase overhead, resulting in notable performance degradation.

The performance differences above can be traced back to the fact that matrix multiplication and SoftMax are composite operators that can be implemented using hybrid protocols, i.e., each consists of multiple basic operators (such as exp and sum) that can be assigned to different types of protocols, such as MPC or FHE. As shown in Figures 1, the performance variations in the transitional ***Gray Areas*** are due to a poor protocol assignment strategy without considering network settings, thereby, it is even worse then a single cryptographic protocol. The performance of hybrid protocols (advantages of near-optimal protocol assignments), represented by the ***Light Purple Area*** in the figures, remains unattainable with current approaches, as they do not automatically adapt protocol assignments to different network settings. This lack of flexibility underscores the necessity of a critical goal: achieving network-agnostic privacy-preserving inference requires a more integrated and optimal protocol assignment strategy that minimizing end-to-end runtime across diverse network environments. In conclusion, this requires a hybrid protocol assignment methodology that automatically assigns the FHE and MPC protocols to each operator, intending to achieve optimal inference performance, as depicted by ***Dark Purple Line*** in Figure 1.

To realize such an automatic hybrid protocol assignment, it is essential to accurately estimate the execution time of each protocol across different operators, parameter settings, and network conditions. Given the large number of operators in complex Transformer-based models, exhaustively testing each operator under all possible configurations is computationally impossible. To address this, a cost model is first used to predict the execution time of each operator under its two possible protocol implementations (FHE and MPC) across varying configurations,i.e., network conditions. Based on these predicted costs, the protocol assignment for the entire model can then be formulated as an optimization problem, which is solved using an ILP solver to determine the optimal assignment for each basic operator. To the best of our knowledge, existing protocol assignment strategies suffer from several critical limitations.

Heavy Customization Effort and Limited Generality. Current cost models, such as those in Pattuk et.al [31] and Schroepfer et.al [32], depend on manually defining the costs of basic operators (e.g., $+$, $-$, $\times$, $\div$, $\geq$, $\gg$, and $\ll$) to estimate the costs of composite operators. This labor-intensive process is not only time-consuming, but also lacks extensibility, making it difficult to adapt these models to the diverse and complex operator sets found in large-scale neural networks. Consequently, their flexibility and applicability are significantly constrained.

Limitations of Static Protocol Assignment Solvers. Existing solvers [20], [19] are primarily designed for single-technology MPC and only account for intra-MPC conversion costs. They are not designed for hybrid FHE-MPC scenarios, where the inclusion of FHE introduces unique challenges. Specifically, changes in the multiplicative depth require updates to encryption parameters to accommodate the new depth, and these adjustments subsequently impact the overall execution cost of FHE. Moreover, sequential homomorphic multiplications that exceed the multiplicative depth limit necessitate bootstrapping to refresh ciphertext noise. These characteristics make it impossible to directly apply SOTA solvers in hybrid MPC-FHE protocol assignments.

To overcome these limitations, we present NETADAPT, a novel framework for the assignment of hybrid protocols that integrates a cost-model-driven approach with an optimization solver. By integrating this cost model with an optimization solver, we can systematically determine the optimal protocol allocation for diverse network environments, thereby enhancing the overall performance. Specifically, our approach leverages Multi-Level Intermediate Representation (MLIR)'s Tensor Operator Set Architecture (TOSA)-level dialect for a unified operator representation, coupled with a Multi-Output training method to generate independent, high-precision cost models for each operator. These models, tailored for FHE and MPC, eliminate manual cost estimation. For protocol assignment, we design an Integer Linear Programming (ILP)-based solver that dynamically updates FHE costs based on homomorphic multiplication depth, incorporating a penalty strategy to account for increasing costs and a constraint to manage FHE-MPC transition overhead. Additionally, homomorphic bootstrapping costs are integrated as a constraint when multiplication depth reaches a preset threshold, further optimizing the assignment process. In practice, we observe that once the bootstrapping constraint is introduced, the solver tends to replace bootstrapping with MPC in most cases.

Specifically, we make the following key contributions.

Generality and High-precision cost prediction model: We're the first to employ MLIR's TOSA-level dialect as a unified operator interface to develop a cost prediction model. The coefficient of determination (R^2) value [33] reaches 95% to 99%. This model offers the flexibility to be used either independently or in a hybrid manner for any complex MPC and FHE applications.

Dynamic Protocol Assignment: We establish an ILP-based optimization model for hybrid protocol assignment, employing two constraints to balance continuous same-protocol assignment with cross-protocol conversion, and dynamically to update FHE costs based on homomorphic multiplication depth.

Network-Adaptive Private Computing Framework: We build the framework NETADAPT and leverage NETADAPT to provide an efficient private inference solution for BERT models. Compared to BumbleBee, the communication volume is reduced by 18.59 GB, and the inference time is improved by 6.68 to 12.92 times. We open-source NETADAPT [34].

II. Preliminaries and Background

A. Notations

We provide descriptions of the frequently used notations in this paper as shown in Table I. FSOP refers to the unified representation of protocols based on FHE, while MSOP refers to the unified representation of protocols based on MPC. The symbol Π_p represents a specific protocol implementation, and if the protocol belongs to MSOP, it is denoted as Π_p^{MPC}.

TABLE I: A table of notations for the mathematical symbols used

Notation	Description
$\mathcal{P}$	Participant
$\mathcal{A}$	Alice (Client)
$\mathcal{B}$	Bob (Server)
$\mathbf{x}/\mathbf{X}$	Vector/Matrix
$\langle m\rangle_{\mathcal{P}}/\langle m\rangle_{\mathcal{P}}^{B}$	Additive/Boolean sharing of m held by $\mathcal{P}$
$[\![\mathbf{m}]\!]_{\mathcal{P}}$	The message encrypted by participant $\mathcal{P}$
Ω	Representation of the Meta-Op
FSOP	FHE-based specific protocol
MSOP	MPC-based specific protocol
Π	Specific protocol implementation
CM	Indicate the cost model, such $\mathrm{CM}_{\mathsf{MPC}}(\Omega_l)$
$\hat{C}$	The cost of Ω predicted by CM

B. FHE

Fully Homomorphic Encryption (FHE) is first proposed by Rivest et al. [11]. We chose a variant of the Cheon-Kim-Kim-Song (CKKS) scheme [35], specifically the Residue Number System-based CKKS (RNS-CKKS) [36], whose security is based on the Ring Learning With Errors (RLWE) problem proposed by Lyubashevsky et al. [37]. The RNS-CKKS scheme comprises several core algorithms, including **KeyGen**, **Encrypt**, **Decrypt**, **HAdd**, **HMult**, **Relin**, **Rescale**, **Rotate**, **Bootstrapping**, and **Square**. A formal description with complete details can be found in [35].

C. MPC

MPC guarantees data privacy and correctness while eliminating the need for a trusted third party. It encompasses various techniques, including Oblivious Transfer (OT) [38], Look-Up Tables (LUTs) [39], and Secret Sharing (SS) [40], [18]. This paper focuses on SS, which allows a set of 2 participants $\mathcal{P} \in \{\mathcal{A}, \mathcal{B}\}$ to share a piece of information in such a way that only authorized subsets of the participants can recover the secret. Then we have the **Additive Scheme**: For every secret $s \in \mathbb{Z}_{2^k}$, an additive scheme satisfies $s = \langle s\rangle_{\mathcal{A}} + \langle s\rangle_{\mathcal{B}} \mod 2^k$.

D. Tensor Operator Set Architecture (TOSA)

Tensor Operator Set Architecture (TOSA) [41] provides a set of whole-tensor operations commonly employed by Deep Neural Networks. TOSA operates on tensors, which are multidimensional data arrays. TOSA operators follow a standardized syntax structure, defined as follows:

$$op_o ::= op_{name}(op_1\ op_2\ \ldots\ op_n) : \text{input-type} \rightarrow \text{output-type} \quad (1)$$

Where op_{name} is the name of the TOSA operator (e.g., $tosa.matmul$, $tosa.add$, etc.). op_i refers to the input tensor of the operator. op_o refers to the output tensor of the operator. The input-type and output-type describe the data type and shape of the input and output tensors (e.g., tensor$< 21 \times 3 \times f32 >$).

For example, the matrix multiplication operation in TOSA IR can be defined as:

```
%output = tosa.matmul(%input1, %input2) :
(tensor<16x32xf32>, tensor<32x64xf32>)
                    -> tensor<16x64xf32>   (2)
```

Most of the widely used inference operators in mainstream ML frameworks can be expressed in TOSA.[1].

E. Threat Model

NetAdapt currently supports CryptFlow2 [29] and Nexus [30] as backends, both of which provide a set of primitives that are secure under the two-party, semi-honest threat model. NetAdapt is not limited to this threat model: it inherits the threat model of any supported backend, so adding a new backend will introduce a corresponding threat model. It only requires that provably secure conversion protocols exist between these primitives. Additionally, when converting FHE primitives to MPC primitives, we provide secure conversion protocols (see Appendix A).

III. Overview & Design of NetAdapt

NetAdapt is an end-to-end automated protocol assignment framework for privacy-preserving machine learning inference tasks, which can automatically determine the optimal protocol implementation by considering both the network environment and operator-specific characteristics. NetAdapt is composed of three components (As shown in Figure 2).

Parser. The Parser (III-A) translates high-level neural networks into primitive operators corresponding to basic FHE-based or MPC-based protocols.

Cost Model. The Cost Model (III-B) takes a primitive operator and the current network environment (e.g., network bandwidth and network latency) as input, and accurately predicts the runtime cost of the protocol designed for that operator using either FHE or MPC.

ILP Solver. The ILP Solver (III-C) formulates the selection between FHE-based and MPC-based protocols as an optimization problem with constraints, and its solution corresponds to the optimal protocol choices for different network environments.

Remark: We assume stable network conditions with no significant latency variations between protocol assignment and deployed execution. Accordingly, the ILP solver runs offline to precompute the optimal protocol assignment, which is deployed at runtime without online inference overhead.

[1]https://mlir.llvm.org/docs/Dialects/TOSA

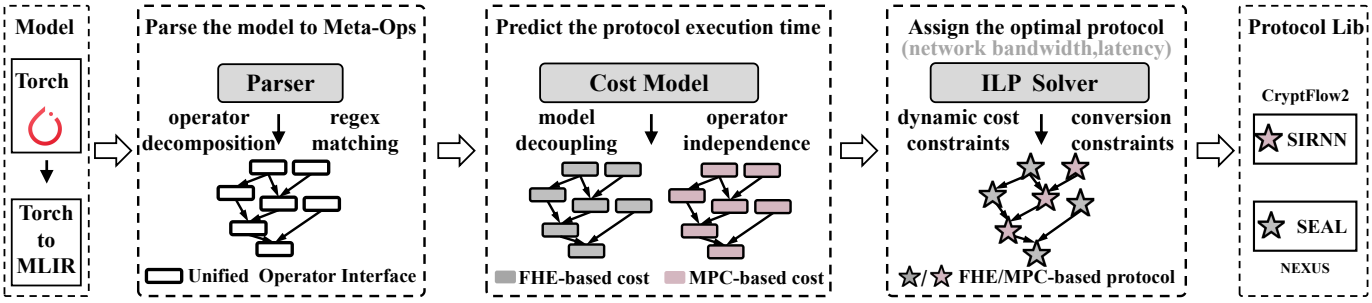


Fig. 2: The overview of NETADAPT. The workflow converts PyTorch models to MLIR, parses them into Meta-Ops, predicts protocol execution times using the Cost Model, assigns optimal FHE/MPC protocols via ILP under specified network configurations, and finally executes the selected protocols through the protocol Lib.

A. Parser: Obtain Meta-Op, the Unified Operator

Design Goals. To support adaptability across different neural network models while remaining compatible with protocols implemented using various cryptographic techniques, the first step is to establish a unified operator abstraction. This requires: 1) Compatibility with mainstream neural network operators, ensuring that the abstraction can support a wide range of widely used neural network models. 2) Decomposability of neural network operators into basic operators that allow a one-to-one mapping to protocols instantiated with different underlying cryptographic primitives.

Limitations and Our Approach. Existing works, such as the CostCO [19] and Silph [20], compile the computations into Boolean circuits (comprising of AND and XOR gates) and calculates cost model at the circuit level. The typical protocol, e.g., Garbled Circuits (GC) [42], for evaluation of a Boolean circuit, is computationally expensive and impractical for large-scale deep learning tasks. NETADAPT adopts a graph-level operator abstraction to establish an effective intermediary layer. Specifically, we use the TOSA dialect in MLIR [41] as an intermediate representation, leveraging the torch-mlir compiler [43] to transform PyTorch models into TOSA operators. We then parse the TOSA operators into Meta-Ops in Static Single Assignment (SSA) form, which serve as the foundational unit for constructing protocol-specific cost models.

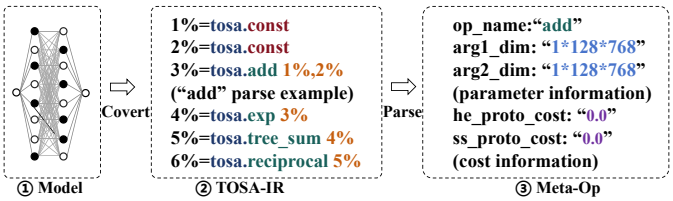


Fig. 3: Operator Modeling: From ML models to Operator-friendly Cryptographic Primitives Meta-Op (It is denoted as $\Omega_l = (\texttt{op}, \texttt{info}, \texttt{cost})$).

The specific formalization transformation process is as follows. As shown in Figure 3, First, define a set of neural network operators $\mathcal{O} = \{O_1, O_2, \ldots, O_l\}$, where each O_i represents an operation (e.g., `reduce_sum`, `mul`, `add`, `rsqrt`). The parser converts the $\mathcal{O}$ into a unified intermediate representation, namely Meta-Ops ($\Omega = \Omega_1, \Omega_2, \ldots, \Omega_k$), that is compatible with both neural networks and cryptographic protocols. Each Meta-Op $\Omega_l = (\text{op}, \text{info}, \texttt{cost}))$ consists of (1) an abstract operator identified by its operator name, (2) a set of computational features (e.g., input/output parameters, tensor dimensions, and operator context information), and (3) the cost information of protocol-specific realizations. Table II presents the unified paradigm for describing a Meta-Op.

TABLE II: Unified Description of a Meta-Op's Paradigm (Such as `tosa.Matmul` op)

Category	Field	Value / Example
Core Operator Info	Operator Name	$MatrixMul$
	Operator ID	OP_001
Input Parameters	*Input 1 Details*	
	Parameter ID	$\%arg0$
	Dimension	$(128, 768)$
	Is_Const	$FALSE$
	Input 2 Details	
	Parameter ID	$\%20$
	Dimension	$(768, 768)$
	Is_Const	$True$
Output Parameters	*Output 1 Details*	
	Parameter ID	$output_matrix_C$
	Dimension	$(128, 768)$
	Is_const	$FALSE$
Cost Information	FHE Protocol Cost	0.0 ms
	MPC Protocol Cost	0.0 ms
	FHE Bootstrapping Cost	0.0 ms
	Protocol Conversion Cost	0.0 ms

Models are first constructed using Torch operators (e.g., `torch.nn .Softmax`), and are then converted into MLIR using `torch-mlir`, from which the TOSA intermediate representation (IR) is obtained. Then, NETADAPT provides an automated parsing tool, built upon predefined matching rules, to extract relevant operator information from the TOSA IR and translate it into Meta-Ops.

Overall, **the Parser consists of three main stages**: Operator Decomposition, Parsing into Meta-Ops, and Protocol Binding. Through this process, a given TOSA IR can be transformed into a unified interface (Meta-Op) that is compatible with underlying protocols instantiated using different cryptographic techniques. Implementation details are provided in Section IV-A.

B. Cost Model: Runtime Prediction for FHE and MPC Protocols

Design Goals. The Cost Model aims to predict the runtime overhead of privacy-preserving protocols associated with

Meta-Ops. To this end, it is designed to satisfy the following goals: 1). The cost model should be scalable across protocols instantiated with different cryptographic techniques (e.g., FHE and MPC), and allow new protocols or cryptographic primitives to be integrated without requiring substantial redesign of the overall modeling framework. 2). The cost model should achieve high prediction accuracy across diverse operators and protocol configurations, enabling reliable protocol selection.

Limitations and Our Approach. Existing Circuit-level cost modeling, as adopted by CostCO and Silph [19], [20], suffers from prohibitive overhead and poor scalability due to expensive Boolean circuit construction when applied to large-scale deep learning workloads. Learning-based methods mitigate this issue by predicting runtime directly from operator- or protocol-level features. Nevertheless, unified cost modeling still has limitations, as cost-dominating model features differ substantially across cryptographic technologies, and the different protocols' runtimes can vary by orders of magnitude even within the same cryptographic technology.

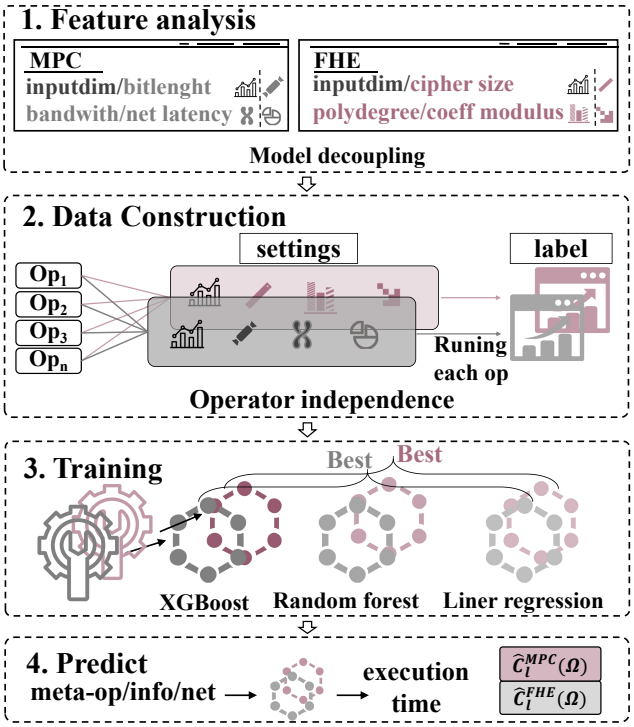


Fig. 4: Workflow of the Cost Model (CM).

NETADAPT designs a learning-based cost model, referred to as CM. As shown in Figure 4, CM predicts the `cost`, such as the execution time of each Meta-Op under different protocols, based on its `info` and the network configuration. During training, CM constructs operator-specific datasets and independently trains multiple regression models, including Linear Regression, Random Forest, and XGBoost, selecting the best-performing model as the final cost predictor.

The design of CM is guided by two core principles, which are detailed below. ① Model decoupling. For different cryptographic technologies, NETADAPT constructs two independent sub-models: CM_{FHE} for FHE and CM_{MPC} for MPC. Specifically, the feature set of CM_{FHE} includes encryption parameters such as `polynomial degree`, `modulus chain`, and the supported `multiplication depth` (key factors determining FHE performance). In contrast, CM_{MPC} incorporates network-related factors (e.g., `bandwidth` and `latency`) as well as cryptographic `ring size` and `fixed-point encoding`. Thanks to this decoupled design, CM_{FHE} can be applied to any homomorphic computation task based on RNS-CKKS, while CM_{MPC} can be applied to any 2-PC MPC task.② Operator independence. Using standard one-hot encoding reduces modeling accuracy, since it ignores large runtime differences between protocols even within the same cryptographic technology, causing cross-interference between Meta-Op samples. NETADAPT adopts an operator independence strategy, where each Meta-Op is modeled by its own regression model. For example, FHE operators such as homomorphic multiplication and addition each have a dedicated model. This design makes the system highly flexible and scalable: if a protocol changes, only its corresponding model needs updating, and new protocols can be added directly without affecting other models.

Overall, **the CM consists of four main stages**: Feature Construction, Training Data Construction, Regression Model Training, and Cost Prediction. Through this process, the execution time of a given Meta-Op Ω_l under FHE-based (FSOP_l) and MPC-based (MSOP_l) specific protocols, as well as any additional conversion or bootstrapping time, can be predicted, enabling protocol selection. Implementation details are provided in Section IV-B.

C. ILP Solver: Protocol Assignment

Design Goals. Given the `cost` predicted by the CM under the specified `info` and network environment, the protocol assignment problem for the entire privacy-preserving inference pipeline is formulated as an optimization problem, where the ILP solver seeks to minimize the end-to-end execution time. This requires: 1). The ILP solver should explicitly manage protocol conversion costs between different cryptographic techniques (e.g., FHE and MPC), so as to avoid inefficient frequent switching across protocols. 2). The ILP solver should dynamically manage the execution cost of FHE by capturing the cost variation caused by multiplication depth accumulation during the assignment process.

Limitations and Our Approach. Existing protocol assignment methods (e.g., Silph [20]) exhibit limitations in hybrid private computation scenarios. First, they typically support only a single cryptographic technique (e.g., FHE or MPC), and thus fail to model the conversion costs between different cryptographic domains. In hybrid FHE–MPC settings, ignoring such costs may lead to frequent protocol switching and significantly degrade overall execution efficiency. Second, existing methods often assume static execution costs, which cannot capture the dynamic cost variation in FHE caused by multiplication depth accumulation. As consecutive FHE multiplications increase the required multiplication depth, the encryption parameters must be strengthened accordingly, leading to higher computational cost even for identical operations. Moreover, once the accumulated depth exceeds a preset depth

bound, additional bootstrapping operations are required to refresh ciphertext noise, further introducing substantial overhead. As a result, these limitations may lead to frequent protocol switching or underestimated FHE costs, preventing globally optimal protocol assignments.

In NETADAPT, to achieve a globally optimal protocol assignment, we introduce three key types of constraints when constructing the ILP solver's objective function. ①. To address protocol conversion in mixed-protocol scenarios, NETADAPT incorporates Protocol Conversion Constraints. The conversion between FHE and MPC data domains follows the approach in FASTLMPI [44], the detailed implementation is provided in Appendix A. And the CM predicts the execution time incurred when a protocol conversion occurs between a given Meta-Op and its preceding Meta-Op. This conversion cost is included in the ILP's optimization objective, enabling the solver to minimize execution time during protocol selection. ②. To capture the dynamic execution cost characteristics of FHE, NETADAPT constructs a dynamic cost function for FHE that is linearly dependent on the accumulated multiplicative depth; as the depth increases, the FHE computation cost rises correspondingly. When the multiplicative depth exceeds a predefined threshold, additional bootstrapping operations are required to refresh the ciphertext noise and ensure correct decryption. This dynamic penalty mechanism pushes the ILP solver to limit consecutive FHE operations. As multiplicative depth grows, increasing penalties lead the solver to switch to MPC before bootstrapping is required. Consequently, bootstrapping is virtually non-existent in the optimal assignments generated by NETADAPT.

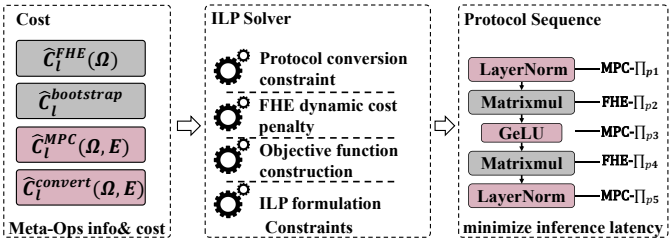


Fig. 5: Workflow of ILP Solver.

Building on the above, NETADAPT further defines ILP Formulation Constraints. Specifically, each Meta-Op can be assigned to either an FSOP or an MSOP, but not both. The system also detects whether an operator's input is a constant and, when applicable, leverages a more efficient FHE plaintext–ciphertext multiplication. By introducing this set of constraints, the ILP model ensures the correctness of protocol assignment while optimizing performance. The ILP workflow is illustrated in Figure 5.

Overall, **the ILP Solver consists of four main stages**: Cost Function Construction, ILP Formulation, Constraints, and Solving & Output. A detailed formal description of the ILP Solver implementation is provided in Section IV-C.

IV. IMPLEMENTATION OF NETADAPT

A. Parser Implementation

This section comprehensively presents the specific implementation details of the Parser.

Step 1. Operators decomposition. Given certain composite TOSA operators (e.g., `tosa.tanh`, `tosa.sigmoid`), which are decomposed into primitive single operators (e.g., `op.exp`, `op.add`, `op.reciprocal`). And the dense resources (e.g., neural network parameters) are handled via a loading interface that forwards the weights to the MPC or FHE contexts.

Step 2. Parsing to Meta-Ops. We design an automated parser based on regular expression matching by following the input–output parameter structure of TOSA operators. The regular expression-based patterns used for double-input operators is defined as:

Listing 1: Regular expression pattern for double-input operators

```
R"(.*?(%[\w\d]+)\s*=\s*(\w+\.\w+)\s*([\w
\d%]+),\s*([\w\d%]+).*?:\s*\(tensor<([^>]
+)>,\s*tensor<([^>]+)>\)\s*->\s*tensor<(
[^>]+)>(\s*#.*)?)");
```

The parser extracts the operator name, input/output parameters, tensor dimensions, and operator IDs from individual operators, and converts them into Meta-Ops $\Omega = \{\Omega_1, \Omega_2, \ldots, \Omega_k\}$. Each Meta-Op $\Omega_l = (op, \mathbf{info}_l, \mathbf{cost}_l)$ consists of an abstract operator (e.g., $op.add$), a feature vector $\mathbf{info}_l$, and the protocol initialization cost $\mathbf{cost}_l$. The feature vector $\mathbf{info}_l$ encodes input dimensions, data types, and other contextual attributes, while the value of $\mathbf{cost}_l$ is predicted by the cost model (CM) introduced later. Through the above parsing and abstraction process, we obtain a unified description of the Meta-Op paradigm, as shown in Table II. For example, for the softmax function in the Attention mechanism, the computation is as follows:

$$\mathbf{y} = \mathsf{SoftMax}(x_i) = \frac{exp(x_i - \overline{x})}{\sum_j exp(x_i - \overline{x})}. \tag{3}$$

where $\overline{x}$ is the max value of the input vector [2]. Table III presents a concise illustration of the Meta-Ops derived from the SoftMax operation.

TABLE III: Meta-OPs of SoftMax.

```
func.func @softmax(input[12x128x128]) ->
    output[12x128x128] {
    %0 = op.max, info{0}, cost{0}
    %1 = op.sub, info{1}, cost{1}
    %2 = op.exp, info{2}, cost{2}
    %3 = op.reduce_sum, info{3}, cost{3}
    %4 = op.reciprocal, info{4}, cost{4}
    %5 = op.mul, info{5}, cost{4}
    return %55[12x128x128]
}
```

This approach standardizes the representation of underlying operators, successfully parses complex models such as BERT, and is easily extendable to new operators and models.

[2]In the original Torch implementation (https://docs.pytorch.org/docs/stable/generated/torch.nn.Softmax.html), the softmax function does not require subtracting $\overline{x}$. However, in FHE and MPC, subtracting $\overline{x}$ is necessary to ensure that all inputs to the exponential function are non-positive, thereby achieving numerical stability. [45], [30]

Step 3. Protocol Binding. It should be noted that NETADAPT focuses on privacy-preserving inference in the 2-PC setting. Accordingly, we provide protocol interfaces for each Meta-Op that are specifically designed for the 2-PC scenario.

For arithmetic operators such as `tosa.matmul`, `tosa.add`, and `tosa.exp`, the corresponding Meta-Ops can be directly mapped to their protocol implementations, e.g., Π_{matmul} and Π_{add}. When `tosa.exp` is mapped to Ω_{exp}, the specific protocol of them under MPC is detailed in Figure 8 in the Appendix B. And the Taylor series approximation is as follows:

$$op.exp \approx (1 + \frac{x}{2^r})^{2^r} \quad , \ x \leq 0 \tag{4}$$

when $r = 8$, the average error is less than 10^{-5}. Then, the specific protocol for Ω_{exp} based on FHE is as shown in Figure 9 in Appendix B. For control-flow and data-movement operators such as `tosa.select`, protocol-specific implementations are required. Specifically, in the MPC setting, `tosa.select` is mapped to the $\Pi^{\mathsf{MPC}}_{\mathsf{if_else}}$ protocol from SIRNN [39]. In the FHE setting, it is implemented via homomorphic evaluation of $result = cond \cdot x + (1 - cond) \cdot y$ using symbolic function algorithms from Nexus [30]. For other MPC protocol implementations, NETADAPT adopts protocols provided by the SIRNN library [39] and the SCI library [29]. For FHE-based implementations, NETADAPT employs the Π_{sign} protocol from Nexus [30]. In addition, the $\Pi^{\mathsf{FHE}}_{\mathsf{matmul}}$ protocol is implemented based on the methods proposed in FASTLMPI [44] and BOLT [27].

B. Cost Model Implementation

This section outlines the implementation of the Cost Model (CM) for the FHE and MPC protocols. For the operator sequence $\Omega = \{\Omega_1, \Omega_2, \ldots, \Omega_k\}$, first map it to FSOP and MSOP. For FSOP, FHE includes the operators shown in Table IV (left). For MSOP, shown in Table IV (right). We conduct the cost model $\mathrm{CM}_{\mathsf{FHE}}$ and $\mathrm{CM}_{\mathsf{MPC}}$ by taking these operators.

TABLE IV: Basic operators for CM.

FHE operators		MPC operators	
add_plain	encode/decode	add	great_equal
add/sub	encrypt/decrypt	sub	if_else
sub_plain	rotate_vector_inplace	abs	reciprocal
mul_plain	mod_switch_to_next	exp	mul_both_private
mul	relinearize_inplace	rsqrt	mul_first_public
square	rescale_to_next_inplace	mul_second_public	He_ss_convert
select	negate_inplace		

Step 1. Feature Construction. We denote the feature vectors corresponding to MSOP and FSOP by $\mathbf{F}^{\mathsf{MPC}}_l$ and $\mathbf{F}^{\mathsf{FHE}}_l$, respectively. Accordingly, we have

$$\Omega_l = (\mathtt{op}_l, \mathtt{info}_l) \xrightarrow{f_{\mathsf{FHE}}} (\mathrm{FSOP}_l, \mathbf{F}^{\mathsf{FHE}}_l),$$

$$\text{and } \xrightarrow{f_{\mathsf{MPC}}} (\mathrm{MSOP}_l, \mathbf{F}^{\mathsf{MPC}}_l).$$

Based on these representations, we construct the $\mathbf{F}^{\mathsf{MPC}}_l$ and $\mathbf{F}^{\mathsf{FHE}}_l$.

FHE features. Include input/output dimensions (`input_dim`, `out-put_dim`) and homomorphic encryption parameters: polynomial degree (`poly_degree`), coefficient modulus (`coeff_modulus`), and ciphertext size ($\frac{\texttt{input_dim}}{\texttt{poly_degree/2}}$):

$$\mathbf{F}^{\mathsf{FHE}}_l = \begin{Bmatrix} \texttt{input_dim}, \texttt{output_dim}, \texttt{poly_degree}, \\ \texttt{coeff_modulus}, \texttt{cipher_size} \end{Bmatrix}.$$

MPC features. Include input dimensions, output dimensions, communication, and encoding parameters: `bitlength`, fixed-point scale (`scale`), network bandwidth (`bandwidth`), and latency (`net_latency`):

$$\mathbf{F}^{\mathsf{MPC}}_l = \begin{Bmatrix} \texttt{input_dim}, \texttt{output_dim}, \texttt{bitlength}, \\ \texttt{scale}, \texttt{bandwidth}, \texttt{net_latency} \end{Bmatrix}.$$

Step 2. Training Data Construction . Once the feature sets $\mathbf{F}^{\mathsf{FHE}}_l$ and $\mathbf{F}^{\mathsf{MPC}}_l$ are determined, we generate the datasets by systematically varying the input dimensions and protocol parameters.

FHE datasets. For each FSOP_l, `input_dim` and `output_dim` vary from 10 to 34,000 with a step of 300. The `poly_degree` ranges from 2^{10} to 2^{15}, and the `coeff_modulus` from 4 to 21. Under these settings, the protocol is executed to measure runtime $\mathbf{T}^{\mathsf{FHE}}_l$ for each feature combination. The dataset is $\mathcal{D}^{\mathsf{FHE}}_l = \{(\mathrm{FSOP}_l, \mathbf{F}^{\mathsf{FHE}}_l, \mathbf{T}^{\mathsf{FHE}}_l)\}$.

MPC datasets. For each MSOP_l, `input_dim` `output_dim` vary from 10 to 34,000 with a step of 200. `bitlength` and fixed-point `scale` are fixed at 37 and 12, respectively. Network bandwidth (`bandwidth`) ranges from 100 to 1000 Mbps, and network latency (`net_latency`) is chosen from $\{3, 5, 10, 20, 30, 40\}$ ms. `bandwidth` and `latency` are simulated using Linux traffic control (tc). Protocol executions under these configurations yield runtimes $\mathbf{T}^{\mathsf{MPC}}_l$, forming the dataset $\mathcal{D}^{\mathsf{MPC}}_l = \{(\mathrm{MSOP}_l, \mathbf{F}^{\mathsf{MPC}}_l, \mathbf{T}^{\mathsf{MPC}}_l)\}$.

These datasets provide comprehensive coverage of feature combinations, enabling regression-based cost modeling for each protocol.

Step 3. Regression Model Training For each dataset $\mathcal{D}^{\mathsf{FHE}}$ and $\mathcal{D}^{\mathsf{MPC}}$, we train three regression models[3] and select the best-performing one for each protocol. Formally, the cost models are $\mathrm{CM}_{\mathsf{FHE}} = \arg\min_{\mathrm{Model}\in\{\mathrm{RF,XGB,LR}\}}(\mathrm{Model}, \mathcal{D}^{\mathsf{FHE}})$, $\mathrm{CM}_{\mathsf{MPC}} = \arg\min_{\mathrm{Model}\in\{\mathrm{RF,XGB,LR}\}}(\mathrm{Model}, \mathcal{D}^{\mathsf{MPC}})$

Direct training on $\mathcal{D}^{\mathsf{FHE}}$ or $\mathcal{D}^{\mathsf{MPC}}$ may yield inaccurate predictions because one-hot encoding of FSOPs or MSOPs does not account for large runtime variations across protocols. For example, the actual running time of the Π_{add} ranges from 7.25 μs to 7914.75 μs, while that of Π_{encode} ranges from 170.25 μs to 29515.875 μs.. To address this, we adopt a Multi-Output Model strategy [46]: the dataset is segmented according to protocols (e.g., data indices $1 \sim 2000$ correspond to Π_{add}, $2001 \sim 4000$ to Π_{sub}), and each segment is trained independently with the three candidate models. The best-performing model within each segment forms the final $\mathrm{CM}_{\mathsf{FHE}}$ and $\mathrm{CM}_{\mathsf{MPC}}$ for that protocol.

Step 4. Cost Prediction. Given a Meta-Op Ω_l, CM predicts runtime by querying the selected cost model. In the FHE setting, $\hat{C}^{\mathsf{FHE}}_l = \mathrm{CM}_{\mathsf{FHE}}(\Omega_l)$, and in the MPC setting,

[3]Random Forest (RF), XGBoost (XGB), and Linear Regression (LR)

$\hat{C}_l^{\mathsf{MPC}} = \mathrm{CM}_{\mathsf{MPC}}(\Omega_l, E)$, where E denotes the network environment. Additionally, CM distinguishes between ciphertext-plaintext and ciphertext-ciphertext operations based on Meta-Op metadata (`Is_const` flag), as the latter are significantly more expensive in FHE [47].

C. ILP Solver Implementation

Our additive cost model is justified by the layer-by-layer execution nature of PPML workloads (e.g., Transformers). Since these models lack inter-operator branching or fusion, the total cost strictly follows a sequential critical path. While individual operators are internally parallelized, this is already encapsulated within our profiled cost constants, making a direct summation both accurate and effective for ILP optimization. Given the results of the CM:

$$\hat{C}_1^{\mathsf{MPC}}, \ldots, \hat{C}_k^{\mathsf{MPC}}, \hat{C}_1^{\mathsf{FHE}}, \ldots, \hat{C}_k^{\mathsf{FHE}}, \hat{C}_l^{\text{bootstrap}}, \text{ and } \hat{C}_l^{\text{convert}}.$$

The ILP solver outputs an optimal protocol assignment sequence $P = \{p_1, p_2, \cdots, p_n\}$, where $p_i \in \{\Pi_l^{\mathsf{FHE}}, \Pi_l^{\mathsf{MPC}}\}$, such that the total cost of the entire computation sequence is minimized. The ILP solver performs the following steps:

Step 1. Cost Function Construction. We define a total cost function $Cost(\mathcal{O}, E)$ for a computation sequence $\mathcal{O}$ under a network environment E. If two adjacent operators are assigned different protocols, a conversion cost $\hat{C}_l^{\text{convert}}$ is added. Moreover, a counter p_{count} tracks the number of consecutive depth-consuming FHE operations; if it exceeds the threshold $n_{\text{threshold}}$, a bootstrapping cost $\hat{C}_l^{\text{bootstrap}}$ is incurred, and the counter resets. Thus, the total cost is expressed as:

$$\begin{aligned} Cost(\mathcal{O}, E) = &\sum_{l=1}^{k} \hat{C}_l(\Omega_l, E) + \sum_{i=2}^{k-1} \mathbf{1}[p_{i-1} \neq p_i] \hat{C}_l^{\text{convert}}(\Omega_l, E) \\ &+ \sum_{l=1}^{k} \mathbf{1}[p_l = \text{FHE} \wedge p_{\text{count}} > n_{\text{threshold}}] \hat{C}_l^{\text{bootstrap}}(\Omega_l). \end{aligned} \tag{5}$$

Step 2. ILP Formulation. Based on the analysis of the cost of computation, we define the following decision variables to ensure that for each operator's protocol assignment, only one cost is selected between MSOP and FSOP. We also introduce conversion cost and bootstrapping cost decision variables to constrain FHE and MPC domain conversions. Then, PAS constructs an ILP model using the following decision variables:

- $x_l^{\mathsf{FHE}}, x_l^{\mathsf{MPC}} \in \{0, 1\}$: whether Ω_l is assigned to FHE or MPC.
- $s_l^{\mathsf{FHE}\to\mathsf{MPC}}, s_l^{\mathsf{MPC}\to\mathsf{FHE}} \in \{0, 1\}$: whether a protocol switch occurs at Ω_l.
- $bot_l \in \{0, 1\}$: whether bootstrapping is required at Ω_l.

The objective is to minimize the overall cost O, consisting of protocol computation costs, conversion costs, and bootstrapping costs:

$$\begin{aligned} O = &\sum_{l=1}^{k} \left(x_l^{\mathsf{FHE}} \hat{C}_{\mathsf{FHE}}(\Omega_l) + x_l^{\mathsf{MPC}} \hat{C}_{\mathsf{MPC}}(\Omega_l, E) \right) \\ &+ \sum_{l=2}^{k} \left(s_l^{\mathsf{FHE}\to\mathsf{MPC}} + s_l^{\mathsf{MPC}\to\mathsf{FHE}} \right) \hat{C}^{\text{convert}}(\Omega_l, E) \\ &+ \sum_{l=1}^{k} bot_l \cdot \hat{C}^{\text{bootstrap}}(\Omega_l). \end{aligned} \tag{6}$$

Step 3. Constraints. To ensure the effectiveness of protocol assignment, it is necessary to construct constraint conditions for the solution model to limit the feasible solution space and ensure that the protocol assignment meets the requirements. PAS introduces the following constraints:

- *Unique assignment:* $x_l^{\mathsf{FHE}} + x_l^{\mathsf{MPC}} = 1, \ \forall l \in [1, k]$.
- *Switching indicators:* $s_l^{\mathsf{FHE}\to\mathsf{MPC}}$ and $s_l^{\mathsf{MPC}\to\mathsf{FHE}}$ are activated only when adjacent operators differ in protocol.
- *Bootstrapping:* If $n_{\text{threshold}}$ consecutive FHE assignments occur, then $bot_l = 1$ at the end of the window.
- *Dynamic update of FHE cost:* consecutive FHE multiplications increase effective cost by applying a penalty factor proportional to the depth.
- *Input parameter type:* if an operator input is constant, the cost is set to ciphertext-plaintext FHE computation; otherwise ciphertext-ciphertext.

Step 4. Solving and Output. Finally, the ILP algorithm [48] is invoked to compute the globally optimal assignment. The solver outputs the protocol mapping $P = \{p_1, \ldots, p_n\}$, protocol switching positions, and bootstrapping events, while ensuring all constraints are satisfied.

The ILP Solver thus provides a protocol schedule that minimizes the total cost under given network conditions and computation requirements. The protocol assignments determined by the ILP solver for the Transformer model are provided in Appendix B.

V. Evaluation

In this section, we evaluate the performance of NetAdapt in three aspects: the prediction performance of the CM III-B, the ILP Solver III-C performance based on dynamic constraints, and the inference performance of NetAdapt on the BERT [49] model with protocol assignment.

A. Experimental Setup

NetAdapt is implemented in C++ and we use EMP toolkit[4] [50] to implement communication between the parties. Leveraging the SCI[5] library in EzPC [51] for fixed-point numerical computations, and the MPC protocol is derived from SIRNN [39] and BOLT [27]. We use the extended SEAL [6] [52] library for RNS-CKKS homomorphic encryption from Nexus [30]. The external libraries are compiled by GCC (version 11.4.0) on Ubuntu 20.04. We open-source our implementation [34].

Benchmark Environment: All the following experiments are performed on a rack server with the Intel(R) Xeon(R) Gold 6238R CPU @ 2.20GHz and 377 gigabytes of RAM. We control the communication bandwidth using the Linux Traffic Control (tc) command. We simulate the WAN environments. which are LAN ={2Gbps, 0ms}, WAN ={100Mbps, 80 ms}, and the number of threads is set to 32. The SoftMax function has a relatively small dimension, so its protocol is implemented using a single thread.

Concrete FHE's and MPC's Parameters: This paper uses

[4]https://github.com/emp-toolkit/emp-tool
[5]https://github.com/mpc-msri/EzPC/tree/master/SCI
[6]https://github.com/microsoft/SEAL

bit-length $k = 37$ and precision $s = 12$ for the expressed of fixed-point numbers, and use the SEAL parameters HE_{SCI}= $\{N = 8192, q \approx 180, p \approx 29\}$ (Similar to [27], [16]) for FHE.
Model Architectures: We measure the performance of NETADAPT on $\text{BERT}_{\text{BASE}}$ [49] and denote the number of layers (i.e., Transformer blocks) as L, the hidden size as H, and the number of self-attention heads as A. The size of feed-forward is $4H$, i.e., 3072 for the $H = 768$ and $L = 12$.

B. Performance of Cost Model

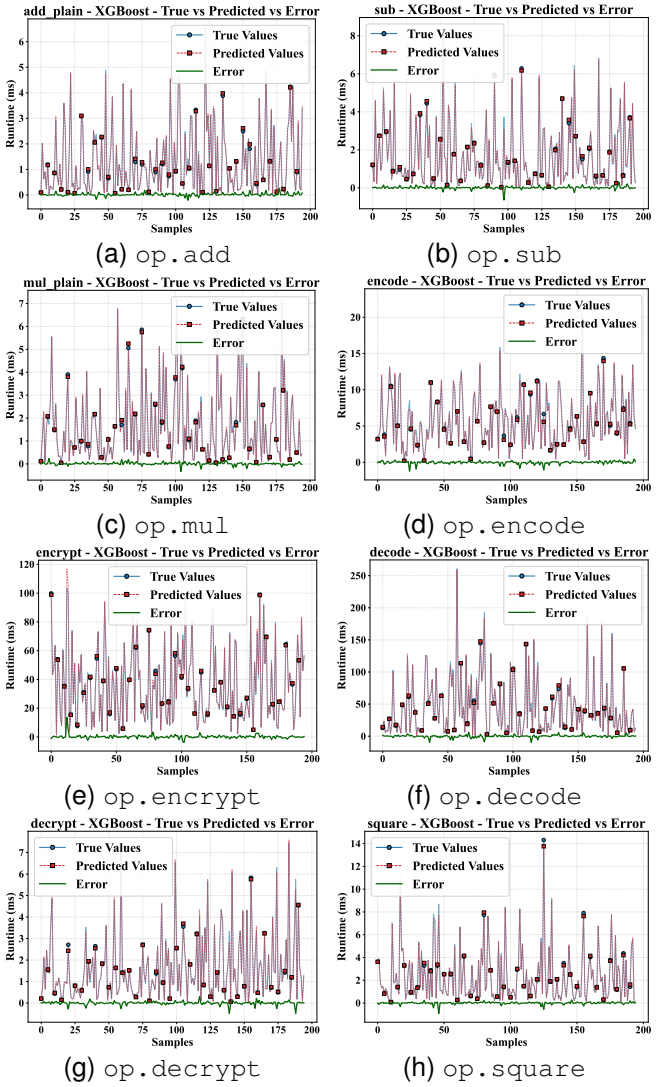
The CM provides high extensibility and replaceability due to its decoupled FHE and MPC cost models, with each operator's cost prediction model being independent. We calculate the fitting capabilities of three different regression models during the model training process. Table V presents their Mean Absolute Error (MAE) and Coefficient of Determination (R^2) values.

TABLE V: Comparison of the performance of the cost model in the agreement. (We list a few of the commonly used protocols.)

Model name	FSOP	MAE(ms)	R^2	MSOP	MAE(μs)	R^2
RandomForest		4.972	0.997		**52.71**	**0.959**
XGBoost	`select`	**2.675**	**0.999**	`if_else`	69.07	0.957
Polynomial		63.8	0.823		827.84	0.159
RandomForest		149.6	0.984		3.42	0.963
XGBoost	`add`	**47.11**	**0.998**	`add`	**1.21**	**0.996**
Polynomial		534.50	0.825		2.449	0.983
RandomForest		99.22	0.983		4.610	0.968
XGBoost	`sub`	**29.26**	**0.998**	`sub`	**2.491**	**0.994**
Polynomial		431.52	0.806		3.309	0.985
RandomForest		90.488	0.967		25.709	0.978
XGBoost	`negate`	**40.500**	**0.992**	`abs`	**18.509**	**0.997**
Polynomial		274.05	0.841		182.437	0.807
RandomForest		125.334	0.984		30.81	0.993
XGBoost	`mul`	**37.427**	**0.999**	`mul`	**23.907**	**0.996**
Polynomial		506.607	0.831		225.923	0.784

As shown in Table V, the values of R^2 all exceeded 0.95 to 0.99. This indicates that the regression model fits the data quite well. However, the MAE values of different protocols vary significantly. The fundamental reason for this is that the costs of different protocols are quite different. For instance, the computational complexity of "`add`" and "`if_else`" under MPC is different. In addition, we observe that the MAE of some protocols is relatively large. This is because during the experiment, the distribution range of our training data is wide, such as the input dimensions ranging from 10 to 34,000. The Protocol costs are easily influenced by the computation dimensions, which affect the MAE of individual operators. Therefore, we trained three regression models and selected the one with the smallest MAE and the highest R^2 as the cost model.

To evaluate the accuracy of the trained cost model, we randomly selected 200 samples from the test set for error analysis. Figure 6 shows the results for FHE, while the corresponding evaluation for MPC is deferred to Appendix G (Figure 14). As observed, the green line (error) is nearly a smooth straight line, with only a few minor jumps. We attribute these slight fluctuations to program execution instability caused by cache effects, which we address by applying median filtering.



(a) `op.add` (b) `op.sub` (c) `op.mul` (d) `op.encode` (e) `op.encrypt` (f) `op.decode` (g) `op.decrypt` (h) `op.square`

Fig. 6: (FHE) Cost model performance of the commonly used operators.

C. Performance of ILP Solver

We verify the performance of the protocol assignment using the ILP Solver, and compare the hybrid protocols with protocols that only use MPC or FHE technologies. As shown in Table VI, we present the protocol assignment performance of the SoftMax, GeLU, and LayerNorm operators used in the BERT model. As a result, for the hybrid protocol assigned by NETADAPT, we test its actual running time and compare it with the predicted time. The difference is small, such as for the SoftMax, LayerNorm, and GeLU functions. The differences between the predicted results and actual time are 27.6 ms, 64.2 ms, and 0.741 s under LAN settings, respectively. From Table VI, it can be seen that as the network setting changes from LAN to WAN, the protocol assignment is more inclined to choose FSOP.
To demonstrate the advantages of the hybrid protocol, we compare it with the protocols that solely used MPC technology or FHE technology. As Table VII shows, for the SoftMax function, the performance of the hybrid protocol was 4.39 times better than that of the FSOP. Compared with MSOP,

TABLE VI: Performance analysis of the ILP Solver. The input of NETADAPT is Meta-Ops, and the computing cost is obtained through the CM. The output is the hybrid protocol and the predicted runtime ("Pred. Runtime"). "Runtime" refers to the actual running time of the hybrid protocol. (Under LAN={2 Gbps, 0 ms}, WAN={100 Mbps, 80 ms} settings.)

Operation	Protocol Assignment												Runtime	Pred. Runtime
SoftMax_LAN	reduce_max ○	sub ●	exp ●	reduce_sum ○	reciprocal ●	mul ●							419.0 ms	391.4 ms
SoftMax_WAN	reduce_max ●	sub ●	exp ●	reduce_sum ●	reciprocal ●	mul ●							5260.01 ms	4880.0 ms
LayerNorm‡	cast ○	reduce_sum ○	mul ◐	sub ◐	mul ◐	reduce_sum ○	mul ○	cast ○	reduce_sum ○	mul ◐	add ◐	rsqrt ●	955.2 ms	1019.4 ms
	sub ◐	mul ◐	reshape ●	mul ◐	reshape ●	add ●								
GeLU_LAN	mul ◐	abs ○	mul ●	add ●	mul ●	mul ●	add ●	mul ●	mul ●	add ●	mul ●	mul ●	3.996 s	3.255 s
	add ●	reciprocal ●	mul ●	mul ●	sub ○	greater_equal ○	negate ○	select ○	add ○	mul ●	mul ●			
GeLU_WAN	mul ◐	abs ●	mul ●	add ●	mul ●	mul ●	add ●	mul ●	mul ●	add ●	mul ●	mul ●	105.22 s	92.17 s
	add ●	reciprocal ●	mul ●	mul ●	sub ○	greater_equal ○	negate ●	select ●	add ●	mul ●	mul ●			

‡ In the LAN and WAN settings, LayerNorm assigned the same hybrid protocol.
Note: ● denotes FSOP, ○ denotes MSOP, ◐ denotes FSOP_plain.

TABLE VII: Performance comparison between hybrid protocols and protocols using only FHE or MPC. (Under LAN={2 Gbps, 0 ms} setting.)

Protocols	Predicted Runtime (s)	Hybrid Runtime (s)	MPC Runtime (s)	FHE Runtime (s)
SoftMax	0.391	**0.419**	12.074	5.26
GELU	3.255	**3.996**	6.603	56.143
LayerNorm	1.019	**0.955**	13.779	36.487

the performance has been improved by 28.8 times, while the communication overhead has been reduced by 7 GB (MPC-based=7913.86 MB, hybrid-based=5.303). For GeLU, the performance of the hybrid protocol is 16.8 times higher than that of the FSOP and 1.7 times higher than that of the MSOP. At the same time, the communication overhead is reduced by 8.34 GB (MPC-based = 10337.99 MB, hybrid-based = 1798.23). For the LayerNorm function, the hybrid protocol outperforms the FSOP by 38.2 times, and outperforms the MSOP by 14.4 times, while reducing the communication volume by 0.96 GB (MPC-based = 1001.81 MB, hybrid-based = 12.79 MB). For the special case of the SoftMax function, as shown by the gray numbers in Table VII, its MSOP is slower than the FSOP in the LAN environment. We split the protocol and found that using MPC technology to implement the exp protocol takes 11.8 seconds, accounting for 97.8% of the all execution time. Therefore, an efficient MPC-based exp protocol is urgently needed.

TABLE VIII: The adaptability of the hybrid protocol to the network environment. (LAN={2 Gbps, 0 ms}, WAN={100 Mbps, 80 ms})

Operation		Runtime (s)		Comm. (MB)
		LAN	WAN	
SoftMax_LAN	128×128	**0.419**	12.07	0.442
SoftMax_WAN		5.26	**5.26**	**0**
GeLU_LAN	128×3072	**3.996**	180.57	1798.23
GeLU_WAN		18.32	**105.22**	**884.20**
LayerNorm	128×768	0.96	3.32	12.79

We also verify the adaptability of the network environment. We set the network environment to WAN = {100 Mbps, 80 ms}, and adopt ILP Solver for protocol allocation. We obtain a hybrid protocol and compare it with the hybrid protocol generated under the LAN environment. The results are shown in Table VIII. We use NETADAPT to assign protocols in both LAN and WAN environments, obtaining various protocol combinations to adapt to different network conditions. Both GeLU and SoftMax demonstrate the network environment adaptation feature of NETADAPT. For LayerNorm, since the use of `reduce_sum` with FHE is very time-consuming, and considering the dimension of LayerNorm is 128×768, during the summation process, it is necessary to rotate a ciphertext $\log_2(768)$ times. However, we set `poly_degree` 8192, thus requiring 24 FHE ciphertexts. Therefore, when the network environment is a WAN, using MPC to implement `reduce_sum` is more efficient, because it only requires transposing the shared data and then performing summation.

Sensitivity to Network Variations and Stability of Protocol Selection. To further investigate the robustness of NETADAPT under diverse network conditions, we extend our evaluation beyond the two representative settings (LAN and WAN) used in the main experiments. Specifically, we consider a wider spectrum of network environments characterized by bandwidth–latency pairs: $(2000, 0)$, $(1000, 10)$, $(600, 20)$, $(400, 30)$, $(200, 40)$, $(100, 80)$. We analyze the protocol selection behavior under these settings using representative operators such as GELU. As shown in Table XI, GELU consists of 23 primitive operators. Among them, only 5 operators switch their protocol choices (between SS and HE) as the network conditions vary. More specifically, (i) 1 operator changes at $(1000, 10)$, (ii) 2 operators change at $(600, 20)$, and (iii) 2 operators change at $(400, 30)$. The remaining majority of operators maintain consistent protocol assignments across all evaluated network settings.

Scalability and Overhead of the ILP Solver. The overhead of the ILP solver in NETADAPT is primarily determined by the number of operators, as each operator introduces a constant number of variables and constraints. Empirically, the solving time scales approximately linearly with the operator count. For example, Softmax (5 operators), LayerNorm (22), and GELU (23) are solved within 15–20 ms, while optimizing a full GPT-2 model requires only 75 ms. Since the optimization is performed offline, this overhead does not affect inference latency, indicating that NETADAPT can scale to larger models without high additional cost.

D. Private BERT Inference Performance

Microbenchmarks: We test the hybrid protocol of SoftMax, GeLU, and LayerNorm generated by NETADAPT, and evaluate

its performance in both LAN and WAN environments. We also compare the results with those of the current SOTA models, IRON [53], BOLT [27], Nexus [30], and BumbleBee [28]. The experimental results are shown in Table IX. The hybrid protocol generated by NETADAPT performs well both in LAN and WAN environments. Such as SoftMax, the protocol assigned by NETADAPT achieves a 4-fold improvement over BOLT [27] in the LAN environment. In the WAN environment, due to the sole adoption of FHE technology, it is slightly faster than Nexus [30]. Compared with the SOTA Bumblebee [28], the communication overhead is reduced by 85.52 MB. The hybrid protocol of LayerNorm requires only 0.96 seconds in a LAN environment and 3.32 seconds under WAN. The communication overhead is 13% of that of BumBleBee. Due to the use of polynomial approximation methods by BOLT, Bumblebee, etc. for GeLU, which is more efficient, the calculation time of the GeLU hybrid protocol in NETADAPT has only slightly increased.

TABLE IX: Comparison of Hybrid protocols with SOTA in terms of running time and communication overhead. Timing results are averaged from 5 runs. Under LAN={2 Gbps, 0 ms}, WAN={100 Mbps, 80 ms} network environment settings.

Method		Time (s)		Comm.(MB)
		LAN	WAN	LAN — WAN
SoftMax 128×128	IRON	5.36	246.09	3356.92
	BOLT	1.96	125.92	1285.13
	Nexus	5.26	5.26	-
	BumbleBee	4.61	31.29	85.96
	NETADAPT	**0.42**	**5.26**	**0.442** — -
LayerNorm 128×768	IRON	2.33	102.60	871.46
	BOLT	1.82	90.17	844.6
	Nexus	39.60	39.60	-
	BumbleBee	14.85	71.67	98.14
	NETADAPT	**0.96**	**3.32**	**12.79**
GeLU 128×3072	IRON	10.30	502.39	7524
	BOLT	4.33	107.89	2769.89
	Nexus	168.13	168.13	-
	BumbleBee	14.85	130.09	**601.79**
	NETADAPT	**3.996**	**105.22**	1798.23—884.20

End-to-End private inference performance of BERT. In Table X, we compare the performance of the hybrid protocol generated by NETADAPT with other private inference methods of BERT, including IRON [53], BOLT [27], BumbleBee [28], and Nexus [30]. In summary, under different network environments, NETADAPT achieves $6.68\times - 12.92\times$ improvements in inference runtime compared to BumbleBee (with BumbleBee runtime of 3131.64, 17409 vs. NETADAPT 242.4, 2606.76 under LAN and WAN).

We also provide Figure 7 to visually demonstrate the performance of the hybrid protocols generated by NETADAPT. **It is important to note that**, due to Nexus relying on pure FHE technology, its CPU runtime is significantly longer. To better highlight the performance differences in Table 10 and Figure 10, we only tested a single layer of the 12-layer transformer in BERT. **Additionally**, while Nexus does not require communication during the inference process itself, the input must be sent from the client to the server at the start and end of the inference, and the result is returned to the client afterward. Therefore, there is still a difference in runtime between LAN and WAN for Nexus.

TABLE X: End-to-end (12-layer Transformer encoder-only model (BERT)) comparisons with existing private inference frameworks. Under LAN={2 Gbps, 0 ms}, WAN={100 Mbps, 80 ms} network environment settings. Framework marked with † is FHE-based (1-layer BERT).

Method	Time (s)		Comm.(GB)	
	LAN	WAN	LAN	WAN
Nexus [30]†	3043.14	5840.34	**0.16**	**0.16**
BumbleBee [28]	3131.64	17409	34.03	34.03
IRON [53]	417.08	18439	272.22	272.22
BOLT [27]	369.19	6211.3	72.21	72.21
NETADAPT	**242.4**	**2606.76**	50.01	15.44

We also evaluate the performance breakdown of NETADAPT on the $\text{BERT}_{\text{BASE}}$ and GPT-2 model, with the detailed results provided in Appendix G2.

VI. RELATED WORK

Cost Model for Privacy Computing. Several studies focus on the cost models for privacy computing tasks, such as [32], [54], [31], [19]. When CostCo [19] developed a framework for automatically generating cost models at the circuit level based on design, they used the number of gates and the depth of the circuit as features to design the cost function and then solved it. CheapMPC [31] represents the program as a DAG (Directed Acyclic Graph), records the execution time and communication time of each operator, and sums them up to serve as the cost function for establishing the cost model. [54] represents eight commonly used basic operations, and it solves them using the cost function in [32]. These costs are achieved by allocating importance parameters and time accumulation.

Hybrid Protocol Assignment. A hybrid protocol can achieve higher performance. Existing works, such as [54], [20], [31], and so on, focus on the assignment of the optimal protocol combination. [54] used both heuristic methods and integer programming to solve the problem, and analyzed the performance differences between the two approaches. CheapMPC [31] employs two heuristic algorithms for the solution, namely Bottom-up Heuristic and Top-Down Heuristic. Silph [20] adopts the graph partitioning method and designs a hybrid ILP optimization framework to assign the protocol.

Hybrid Protocol Frameworks. Frameworks such as ABY [17], ABY3 [18], EzPC [51], and Secretflow [55] leverage DSLs and dedicated compilers to automatically translate complex applications into computations supported by underlying cryptographic libraries. ABY and ABY3 provide hybrid protocol frameworks supporting arithmetic sharing, boolean sharing, and Yao's sharing; EzPC employs an end-to-end compiler that invokes the SCI library to realize privacy-preserving computation; and Secretflow uses an MLIR-based dialect to map applications to cryptographic protocols, executing privacy tasks with a fixed hybrid protocol.

VII. DISCUSSION

Extensibility of NETADAPT. NETADAPT constructs independent cost models for each matched protocol based on the operator granularity of TOSA IR, covering all Meta-Ops in BERT to perform private inference for the BERT model. It can be extended to models like GPT [56]. However, support for visual

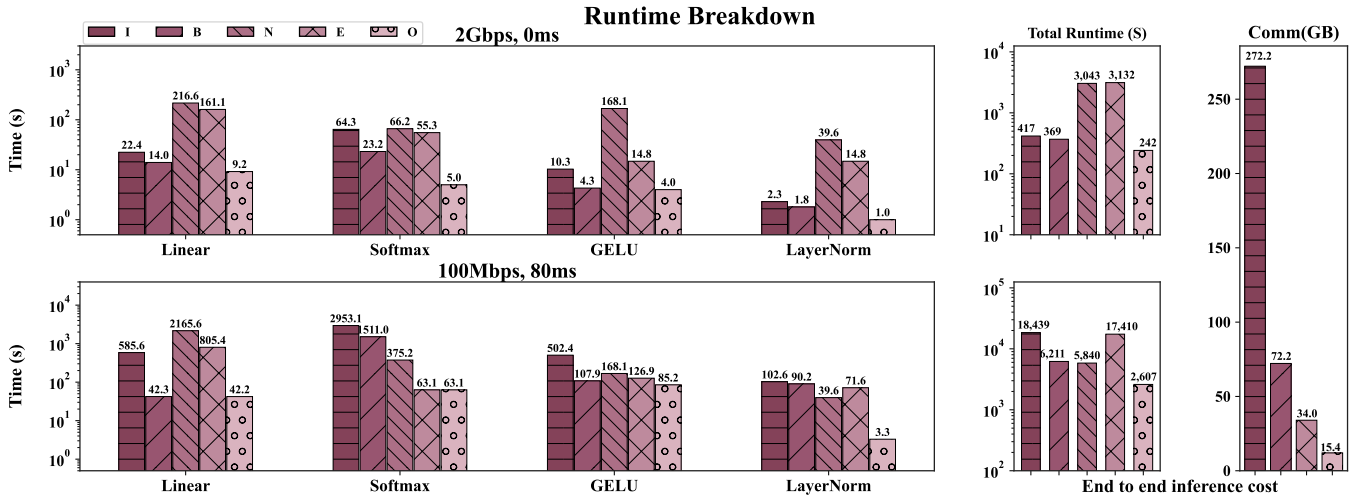


Fig. 7: End-to-End linear & non-linear operators' performance (One layer-BERT) & Total private inference time of 12-layers BERT & Total communication overhead of 12-layers BERT (**Nexus for 1-layer BERT**). (" " (I) denotes IRON, (" " (B) denotes BOLT, (" " (N) denotes Nexus, (" " (E) denotes BumbleBee, " " (O) denotes our NETADAPT. Under LAN={2 Gbps, 0 ms}, WAN={100 Mbps, 80 ms} network environment settings. "Linear" represents all the matrix multiplication. Refer to Table XIII.

models is limited as the underlying protocol only provides a generic matrix multiplication protocol and lacks specialized optimization for convolution operations. This limitation stems from the maturity of the underlying protocol. Nevertheless, NETADAPT supports the extension of cost models for basic operators beyond those in Table IV, enabling support for more Torch machine learning models.

Limitations of NETADAPT. NETADAPT can provide optimal protocol assignment in different network environments through its cost model and dynamic ILP solver. However, the current network environments are defined based on predefined bandwidth and latency, whereas in real-world applications, network conditions may fluctuate in real-time, which poses a challenge for NETADAPT. Future improvements could integrate real-time network monitoring services to dynamically adjust the ILP solver, enabling finer-grained adaptation to evolving network conditions and further enhancing performance.

VIII. CONCLUSION & FUTURE WORK

This paper proposes a cost-model-driven hybrid protocol assignment framework for 2-PC neural network inference. This method generates protocol assignments for different network environments, addressing the limitations of existing methods in adapting to network context fluctuations. Experiments on the representative BERT model, which uses larger matrix sizes and more complex activation functions, show that the hybrid protocol assigned by NETADAPT yields significant performance improvements for private-preserving inference tasks. Future research will focus on two primary directions to enhance NETADAPT. Convolution operations currently use generic matrix multiplication protocols, which are not ideal for vision tasks. Enhancing these protocols will boost inference efficiency in privacy-preserving vision applications. NETADAPT works under fixed network conditions, reducing its effectiveness during real-time fluctuations. A dynamic protocol strategy that adapts to real-time changes will improve robustness in dynamic settings.

REFERENCES

[1] C. Dong, J. Weng, J. Liu, Y. Zhang, Y. Tong, A. Yang, Y. Cheng, and S. Hu, "Fusion: Efficient and secure inference resilient to malicious servers," 2023.

[2] W. Liu, P. Lu, and J. Gao, "Rgcgt: A high-order feature learning framework for predicting disease-metabolite interaction using residual graph convolution and graph transformer," *Expert Systems with Applications*, p. 126637, 2025.

[3] J. Xu and Y. Wang, "Enhancing healthcare recommendation systems with multimodal llms-based moe architecture," in *5th International Conference on Signal Processing and Machine Learning (CONF SPML 2025)*, vol. 2025. IET, 2025, pp. 123–129.

[4] T. R. Akash, N. Lessard, N. R. Reza, and M. S. Islam, "Investigating methods to enhance data privacy in business, especially in sectors like analytics and finance," *Journal of Computer Science and Technology Studies*, vol. 6, no. 5, pp. 143–151, 2024.

[5] A.-T. Tran, T.-D. Luong, and V.-N. Huynh, "A comprehensive survey and taxonomy on privacy-preserving deep learning," *Neurocomputing*, p. 127345, 2024.

[6] Y. Gao, H. Deng, Z. Zhu, X. Chen, Y. Xie, P. Duan, and P. Chen, "Peafowl: Private entity alignment in multi-party privacy-preserving machine learning," *IEEE TIFS*, 2025.

[7] L. Zhou, B. Zhang, Z. Wang, T. Lu, Q. Song, S. Zhang, H. Cui, and Y. Yu, "On probabilistic truncation in privacy-preserving machine learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 39, no. 21, 2025, pp. 22 955–22 964.

[8] P. Mohassel and Y. Zhang, "Secureml: A system for scalable privacy-preserving machine learning," in *2017 IEEE symposium on security and privacy (SP)*. IEEE, 2017, pp. 19–38.

[9] M. Al-Rubaie and J. M. Chang, "Privacy-preserving machine learning: Threats and solutions," *IEEE Security & Privacy*, vol. 17, no. 2, pp. 49–58, 2019.

[10] S. E. V. S. Pillai and K. Polimetla, "Enhancing network privacy through secure multi-party computation in cloud environments," pp. 1–6, 2024.

[11] R. L. Rivest, L. Adleman, M. L. Dertouzos *et al.*, "On data banks and privacy homomorphisms," *Foundations of secure computation*, vol. 4, no. 11, pp. 169–180, 1978.

[12] G. Su, J. Wang, X. Xu, Y. Wang, and C. Wang, "The utilization of homomorphic encryption technology grounded on artificial intelligence for privacy preservation," *International Journal of Computer Science and Information Technology*, vol. 2, no. 1, pp. 52–58, 2024.

[13] Y. Wei, X. Wang, S. Bian, Y. Huang, W. Zhao, and Y. Jin, "Ppgnn: Fast and accurate privacy-preserving graph neural network inference via parallel and pipelined arithmetic-and-logic fhe accelerator," ser. DAC '24. Association for Computing Machinery, 2024.

[14] A. Gillioz, J. Casas, E. Mugellini, and O. Abou Khaled, "Overview of the transformer-based models for nlp tasks," in *2020 15th Conference on computer science and information systems (FedCSIS)*, 2020.

[15] A. D. Rasamoelina, F. Adjailia, and P. Sinčák, "A review of activation function for artificial neural network," in *2020 IEEE 18th world symposium on applied machine intelligence and informatics (SAMI)*. IEEE, 2020, pp. 281–286.

[16] Z. Huang, W. j. Lu, C. Hong, and J. Ding, "Cheetah: Lean and fast secure {Two-Party} deep neural network inference," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 809–826.

[17] D. Demmler, T. Schneider, and M. Zohner, "Aby-a framework for efficient mixed-protocol secure two-party computation." in *NDSS*, 2015.

[18] P. Mohassel and P. Rindal, "Aby3: A mixed protocol framework for machine learning," pp. 35–52, 2018.

[19] V. Fang, L. Brown, W. Lin, W. Zheng, A. Panda, and R. A. Popa, "Costco: An automatic cost modeling framework for secure multi-party computation," in *2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2022, pp. 140–153.

[20] E. Chen, J. Zhu, A. Ozdemir, R. S. Wahby, F. Brown, and W. Zheng, "Silph: A framework for scalable and accurate generation of hybrid mpc protocols," in *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2023, pp. 848–863.

[21] Q. Zhang, T. Xiang, C. Xin, and H. Wu, "From individual computation to allied optimization: Remodeling privacy-preserving neural inference with function input tuning," in *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2024, pp. 4810–4827.

[22] Q. Zhang, T. Xiang, C. Xin, and H. Wu, "United we stand: Accelerating privacy-preserving neural inference by conjunctive optimization with interleaved nexus," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, no. 15, 2024, pp. 16 794–16 802.

[23] T. Chen, H. Bao, S. Huang, L. Dong, B. Jiao, D. Jiang, H. Zhou, J. Li, and F. Wei, "THE-X: Privacy-preserving transformer inference with homomorphic encryption," in *Findings of the Association for Computational Linguistics: ACL 2022*, S. Muresan, P. Nakov, and A. Villavicencio, Eds. Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 3510–3520.

[24] Z. Li, K. Yang, J. Tan, W.-j. Lu, H. Wu, X. Wang, Y. Yu, D. Zhao, Y. Zheng, M. Guo, and J. Leng, "Nimbus: Secure and efficient two-party inference for transformers," in *Advances in Neural Information Processing Systems*, vol. 37. Curran Associates, Inc., 2024.

[25] T. Xu, L. Wu, R. Wang, and M. Li, "Privcirnet: Efficient private inference via block circulant transformation," in *Advances in Neural Information Processing Systems*, A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, Eds., vol. 37. Curran Associates, Inc., 2024, pp. 111 802–111 831. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2024/file/ca9873918aa72e9033041f76e77b5c15-Paper-Conference.pdf

[26] T. Xu, W. jie Lu, J. Yu, Y. Chen, C. Lin, R. Wang, and M. Li, "Breaking the layer barrier: Remodeling private transformer inference with hybrid CKKS and MPC," in *34th USENIX Security Symposium (USENIX Security 25)*. Seattle, WA: USENIX Association, Aug. 2025, pp. 2653–2672. [Online]. Available: https://www.usenix.org/conference/usenixsecurity25/presentation/xu-tianshi

[27] Q. Pang, J. Zhu, H. Mollering, Z. Wenting, and T. Schneider, "Bolt: Privacy-preserving, accurate and efficient inference for transformers."

[28] W.-j. Lu, Z. Huang, Z. Gu, J. Li, J. Liu, K. Ren, C. Hong, T. Wei, and W. Chen, "Bumblebee: Secure two-party inference framework for large transformers," *Cryptology ePrint Archive*, 2023.

[29] D. Rathee, M. Rathee, N. Kumar, N. Chandran, D. Gupta, A. Rastogi, and R. Sharma, "Cryptflow2: Practical 2-party secure inference," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 325–342.

[30] J. Zhang, J. Liu, X. Yang, Y. Wang, K. Chen, X. Hou, K. Ren, and X. Yang, "Secure transformer inference made non-interactive," *Cryptology ePrint Archive*, 2024.

[31] E. Pattuk, M. Kantarcioglu, H. Ulusoy, and B. Malin, "Cheapsmc: A framework to minimize secure multiparty computation cost in the cloud," in *Data and Applications Security and Privacy XXX: 30th Annual IFIP WG 11.3 Conference, DBSec 2016, Trento, Italy, July 18-20, 2016. Proceedings 30*. Springer, 2016, pp. 285–294.

[32] A. Schroepfer and F. Kerschbaum, "Forecasting run-times of secure two-party computation," in *2011 Eighth International Conference on Quantitative Evaluation of SysTems*. IEEE, 2011, pp. 181–190.

[33] O. Renaud and M.-P. Victoria-Feser, "A robust coefficient of determination for regression," *Journal of Statistical Planning and Inference*, vol. 140, no. 7, pp. 1852–1862, 2010.

[34] "Netadapt," https://anonymous.4open.science/r/NetAdapt, Jan. 2026.

[35] J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song, "A full rns variant of approximate homomorphic encryption," in *International Conference on Selected Areas in Cryptography*. Springer, 2018, pp. 347–368.

[36] E. Lee, J.-W. Lee, J. Lee, Y.-S. Kim, Y. Kim, J.-S. No, and W. Choi, "Low-complexity deep convolutional neural networks on fully homomorphic encryption using multiplexed parallel convolutions," in *International Conference on Machine Learning*, 2022.

[37] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," pp. 1–23, 2010.

[38] M. Naor and B. Pinkas, "Efficient oblivious transfer protocols." in *SODA*, vol. 1, 2001, pp. 448–457.

[39] D. Rathee, M. Rathee, R. K. K. Goli, D. Gupta, R. Sharma, N. Chandran, and A. Rastogi, "Sirnn: A math library for secure rnn inference," in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 1003–1020.

[40] H. Ghodosi, J. Pieprzyk, and R. Steinfeld, "Multi-party computation with conversion of secret sharing," *Designs, Codes and Cryptography*, vol. 62, pp. 259–272, 2012.

[41] C. Lattner, M. Amini, U. Bondhugula, A. Cohen, A. Davis, J. Pienaar, R. Riddle, T. Shpeisman, N. Vasilache, and O. Zinenko, "Mlir: A compiler infrastructure for the end of moore's law," *arXiv preprint arXiv:2002.11054*, 2020.

[42] A. C.-C. Yao, "How to generate and exchange secrets," in *27th annual symposium on foundations of computer science (Sfcs 1986)*. IEEE, 1986, pp. 162–167.

[43] H. Kwon, H. M. Kim, C.-G. Lyuh, J.-K. Kim, J. Han, and Y. Kwon, "Aiwarek: Compiling pytorch model for ai processor using mlir framework," in *2022 IEEE 4th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. IEEE, 2022, pp. 463–465.

[44] Y. Chen, Z. Tang, T. Lu, B. Zhang, Z. Shi, and Z. Wang, "Accelerating private large transformers inference through fine-grained collaborative computation," 2025. [Online]. Available: https://arxiv.org/abs/2412.16537

[45] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[46] H. Borchani, G. Varando, C. Bielza, and P. Larranaga, "A survey on multi-output regression," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 5, no. 5, pp. 216–233, 2015.

[47] X. Ren, L. Su, Z. Gu, S. Wang, F. Li, Y. Xie, S. Bian, C. Li, and F. Zhang, "Heda: multi-attribute unbounded aggregation over homomorphically encrypted database," *Proceedings of the VLDB Endowment*, vol. 16, no. 4, pp. 601–614, 2022.

[48] J. P. Vielma, "Mixed integer linear programming formulation techniques," *Siam Review*, vol. 57, no. 1, pp. 3–57, 2015.

[49] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[50] X. Wang, A. J. Malozemoff, and J. Katz, "EMP-toolkit: Efficient MultiParty computation toolkit," https://github.com/emp-toolkit, 2016.

[51] N. Chandran, D. Gupta, A. Rastogi, R. Sharma, and S. Tripathi, "Ezpc: Programmable and efficient secure two-party computation for machine learning," in *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2019, pp. 496–511.

[52] "Microsoft SEAL (release 4.1)," https://github.com/Microsoft/SEAL, Jan. 2023, microsoft Research, Redmond, WA.

[53] M. Hao, H. Li, H. Chen, P. Xing, G. Xu, and T. Zhang, "Iron: private inference on transformers," *Proceedings of the 36th International Conference on Neural Information Processing Systems*, 2022.

[54] F. Kerschbaum, T. Schneider, and A. Schröpfer, "Automatic protocol selection in secure two-party computations," in *Applied Cryptography and Network Security: 12th International Conference, ACNS 2014, Lausanne, Switzerland, June 10-13, 2014. Proceedings 12*, 2014.

[55] J. Ma, Y. Zheng, J. Feng, D. Zhao, H. Wu, W. Fang, J. Tan, C. Yu, B. Zhang, and L. Wang, "{SecretFlow-SPU}: A performant and {User-Friendly} framework for {Privacy-Preserving} machine learning," in *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, 2023.

[56] Y. Chang, X. Wang, J. Wang, Y. Wu, L. Yang, K. Zhu, H. Chen, X. Yi, C. Wang, Y. Wang *et al.*, "A survey on evaluation of large language models," *ACM Transactions on Intelligent Systems and Technology*, vol. 15, no. 3, pp. 1–45, 2024.

[57] H. Qu and G. Xu, "Improvements of homomorphic secure evaluation of inverse square root," in *International Conference on Information and Communications Security*. Springer, 2023, pp. 110–127.

APPENDIX

A. Conversion between FHE and MPC

Since the FSOP and MSOP are over different computation domains, it is necessary to convert the data from the field $\mathbb{Z}_p$ to the ring $\mathbb{Z}_{2^k}$. We elaborate on the field-ring conversion process. Given the equation $y_0 + y_1 \equiv y \pmod p$, our goal is to compute $x_0, x_1 \in \mathbb{Z}_{2^k}$ s.t. $x_0 + x_1 \equiv y \pmod{2^k}$. Assuming $2^k > p$, this reduces to a *share extension (SExt)* problem [39]. The conversion, whose security against semi-honest adversaries has been proven in [27], proceeds in the following steps:

Compute the wrap-bit w. First, we calculate the wrap-bit (a comparison is required) w, defined as the indicator function $w = 1_{(y_0+y_1 \geq p)}$ (a multiplexer is required). Formally:

$$w = \begin{cases} 1, & \text{if } y_0 + y_1 \geq p, \\ 0, & \text{otherwise.} \end{cases}$$

This bit w indicates whether the sum y_0+y_1 exceeds the prime modulus p.

Convert wrap-bit to Ring. Next, we convert the binary wrap-bit w into ring shares $w_0, w_1 \in \mathbb{Z}_{2^k}$ such that $w_0 + w_1 \equiv w \pmod{2^k}$. This step is known as *Boolean-to-Arithmetic (B2A) conversion [18]*, transforming a Boolean value into additive shares over the ring $\mathbb{Z}_{2^k}$.

Compute x_0 and x_1. Finally, we compute the target values using the formula: $x_b = y_b - p \cdot w_b \pmod{2^k}$ for $b = 0, 1$. This adjusts the original shares y_0, y_1 by subtracting multiples of p scaled by the wrap-bit shares w_0, w_1, ensuring the result lies in the ring $\mathbb{Z}_{2^k}$.

In NETADAPT, conversions between the FSOP and MSOP (or from MSOP to FSOP) implicitly perform the required field-ring conversion. For clarity, this step is omitted from the protocol descriptions.

B. Protocols used in NETADAPT

This Section provides a detailed description of the protocols in NETADAPT.

Protocol $\Pi^{\mathsf{MPC}}_{\mathsf{exp}}(m, s, n, s')$

Input: $\mathcal{A}$ & $\mathcal{B}$ hold share $\langle x \rangle^m$
Output: $\mathcal{A}$ & $\mathcal{B}$ get $\langle y \rangle^n$ for $y = exp(-x)$
Execution:

- Each party calls $\Pi^{m,d}_{\mathsf{DigDec}}$ to get $\langle h_i \rangle^d_{i\in[k]}$, $h_i \in 0, 1^d$ s.t. $x = h_{k-1}||...||h_0, (dk = m)$.
- For $i \in [k]$, let $\langle L_i \rangle : 0, 1^d \to \mathbb{Z}_2^{s'+2}$ s.t. $\langle L_i(j) \rangle = \text{FIX}^a(\Pi^{T,m,n}_{\mathsf{LUT}}(2^{di-s}j), s' + 2, s')$
- Each party gets $\langle g \rangle$ by calling $\Pi^{s'+2,s'+2,2s'+2,s'}_{\mathsf{SMultTR}}(\langle L_{k-1}[x_{k-1}] \rangle) * \cdots * \langle L_0[x_0] \rangle)*$. $\langle g \rangle$ has bitwidth $s' + 2$ and scale s'.
- Each party returns $\langle y \rangle^n = \Pi_{\mathsf{SExt}}(\langle g \rangle, s' + 2, n)$

[a]"FIX" indicates the process of converting the result obtained from the lookup table into a fixed-point number and then transforming it into a Ring element.

Fig. 8: Protocol $\Pi^{\mathsf{MPC}}_{\mathsf{exp}}$ for 2PC.

Where m and n denote the bit-length and s and s' denote the scaling factors. Implementation details of auxiliary protocols such as Π_{LUT} and Π_{SExt} can be found in SIRNN [39]. In the FHE setting, Ω_{exp} is implemented using polynomial or Taylor-series-based approximation methods [28], [30][7].

Protocol $\Pi^{\mathsf{FHE}}_{\mathsf{exp}}$

Input: $\mathcal{B}$ holds FHE-ciphertext $[\![-\mathbf{x}]\!]_{\mathcal{A}}$
Output: $\mathcal{B}$ gets $[\![\mathbf{y}]\!]_{\mathcal{A}}$ for $y = exp(x)$
Execution:

- Encode $\frac{1}{2^r}$ through SIMD, obtaining $p = Plaintext(\frac{1}{2^r})$. Calculate $[\![l]\!]_{\mathcal{A}} = [\![\mathbf{x}]\!]_{\mathcal{A}} \boxdot p$, and perform $Rescale([\![l]\!]_{\mathcal{A}})$.
- Encode 1 through SIMD, getting $n = Plaintext(1)$. Compute $[\![y]\!]_{\mathcal{A}} = [\![\mathbf{x}]\!]_{\mathcal{A}} \boxplus n$. [a]
- for— i, where $i < \log(2^r)$ do
  - $HSquare([\![y]\!]_{\mathcal{A}}, [\![y]\!]_{\mathcal{A}})$
  - $HRelin([\![y]\!]_{\mathcal{A}})$
  - $HRescale([\![y]\!]_{\mathcal{A}})$

[a]$\boxdot$ denotes two ciphertext multiplication, $\boxplus$ denotes two ciphertext addition.

Fig. 9: Protocol $\Pi^{\mathsf{FHE}}_{\mathsf{exp}}$ for 2PC.

The protocols used in the Transformer model include proto-

Protocol $\Pi^{\mathsf{FHE}}_{\mathsf{matmul}}$

Initialization:
$\mathcal{A}$ and $\mathcal{B}$ generate each private key $sk_{\mathcal{A}}$, $sk_{\mathcal{B}}$ and public key $pk_{\mathcal{A}}$, $pk_{\mathcal{B}}$.
Inputs:
$\mathcal{A}$: holds the matrix $\mathbf{A} \in \mathbb{Z}_p^{d_m * d_n}$
$\mathcal{B}$: holds the matrix $\mathbf{B} \in \mathbb{Z}_p^{d_n * d_h}$
Output: $\mathcal{A}$ and $\mathcal{B}$ get $\langle \mathbf{Y} \rangle_{\mathcal{A}}$, $\langle \mathbf{Y} \rangle_{\mathcal{B}}$, respectively (or $\mathcal{B}$ get $[\![\mathbf{Y}]\!]_{\mathcal{A}}$) for $\mathbf{Y} = \mathbf{A} \odot \mathbf{B}$
Execution:
$\mathcal{A}$: Encodes $\mathbf{A}$ to obtain $\widetilde{\mathbf{A}}$; Encrypts $\widetilde{\mathbf{A}}$ by $pk_{\mathcal{A}}$ yields SIMD-ciphertesxts $[\![\widetilde{\mathbf{A}}]\!]_{\mathcal{A}}$; Sends $[\![\widetilde{\mathbf{A}}]\!]_{\mathcal{A}}$ to $\mathcal{B}$.
$\mathcal{B}$: Receives $[\![\widetilde{\mathbf{A}}]\!]_{\mathcal{A}}$; Computes $[\![\mathbf{Y}]\!]_{\mathcal{A}} = [\![\widetilde{\mathbf{A}}]\!]_{\mathcal{A}} \boxdot \mathbf{B}$; Generates uniformly random $\langle \mathbf{Y} \rangle_{\mathcal{B}} = \mathbf{R} \in \mathbb{Z}_p^{d_m * d_h}$; Computes $[\![\langle \mathbf{Y} \rangle_{\mathcal{A}}]\!]_{\mathcal{A}} = [\![\mathbf{Y}]\!]_{\mathcal{A}} \boxminus \mathbf{R}$; Sends $[\![\langle \mathbf{Y} \rangle_{\mathcal{A}}]\!]_{\mathcal{A}}$ to $\mathcal{A}$.
$\mathcal{A}$: Receives $[\![\langle \mathbf{Y} \rangle_{\mathcal{A}}]\!]_{\mathcal{A}}$; Decrypts $[\![\langle \mathbf{Y} \rangle_{\mathcal{A}}]\!]_{\mathcal{A}}$ to get $\langle \mathbf{Y} \rangle_{\mathcal{A}}$. [a]

[a]$\odot$ denotes plaintext matrix multiplication, $\boxdot$ denotes ciphertext matrix multiplication, and $\boxminus$ denotes ciphertext subtraction.

Fig. 10: Protocol $\Pi^{\mathsf{FHE}}_{\mathsf{matmul}}$ for 2PC.

col matrix multiplication Π_{Matmul}, protocol softmax Π_{SoftMax}, protocol layerNorm $\Pi_{\mathsf{layerNorm}}$, protocol GELU Π_{GeLU}. These protocols are built upon the primitive protocols provided by CryptFlow [29], FASTLMPI [44], BOLT [27], and Nexus [30].

C. Protocol Π_{matmul}

For Π_{matmul}, both BOLT and FASTLMPI employ FHE techniques optimized through SIMD encoding. This choice is motivated by the fact that MPC-based Π_{matmul} requires the generation of Beaver triples [39], which typically depend on a trusted third party for preprocessing, or need to be generated online using OT [38] or FHE techniques. In contrast, the encoding-optimized FHE-based Π_{matmul} is more efficient and requires less communication.. In NETADAPT, we adopt this encoding-optimized FHE approach for the Π_{matmul}. The protocol $\Pi^{\mathsf{FHE}}_{\mathsf{matmul}}$ is shown in Figure 10.

[7]the MPC protocol $\Pi^{\mathsf{MPC}}_{\mathsf{exp}}(m, s, n, s')$ provided by SIRNN [39] can be employed

D. Protocol Π_{softmax}

The protocol Π_{softmax} is composed of reducemax (`max`), subtraction (`sub`), exponentiation (`exp`), reducesum (`tree_sum`), reciprocal (`inv`), and Multiplication (`mul`).

1) sub-protocol Π_{max}: For the `max` protocol Π_{max}, the implementation uses MPC and computes the maximum via a tree-style reduction. The algorithm first transposes the input vectors so that elements at the same position can be processed independently. It then constructs a comparison tree, where in each round secure comparisons and conditional selections are used to determine the larger values, which are progressively merged until the final maximum is obtained. The MPC-based protocol follows BOLT [27] and is denoted as $\Pi_{\mathsf{max}}^{\mathsf{MPC}}$. The FHE-based max protocol relies on a homomorphic approximation of the sign function [30] and is denoted as $\Pi_{\mathsf{max}}^{\mathsf{FHE}}$. Since FHE cannot directly compare encrypted values, the maximum between two ciphertexts x and y is computed approximately as:

$$\max(x, y) \approx \frac{x+y}{2} + \frac{(x-y)\cdot \mathrm{sgn}(x-y)}{2},$$

where $\mathrm{sgn}(\cdot)$ denotes the homomorphic sign evaluation. This approximation forms the core operation for the FHE-based implementation of the maximum function.

2) sub-protocol Π_{exp}: The protocol $\Pi_{\mathsf{exp}}^{\mathsf{FHE}}$, shown in Figure 9, uses a Taylor series approximation, while the protocol $\Pi_{\mathsf{exp}}^{\mathsf{MPC}}$, shown in Figure 8, relies on a lookup table for the efficient computation of the exponential function.

3) sub-protocol Π_{inv}: For the `reciprocal` protocol Π_{inv}, a Newton iteration method is used, as described below:

$$y_{k+1} = y_k\,(2 - x\,y_k)\,, \qquad k = 0, 1, 2, \ldots$$

where x is the ciphertext input, y_k is the approximation of the inverse obtained at the k-th iteration, and eventually $y_k \approx \frac{1}{x}$. The `reciprocal` is approximated and then realized using FHE, denoted as Π_{inv}^{FHE}. In contrast, the `reciprocal` protocol in MPC, denoted as Π_{inv}^{MPC} employs an efficient lookup table method for computing inverses, as described in Cryptflow2 [29].

4) sub-protocol Π_{sub}, $\Pi_{\mathsf{tree_sum}}$ and Π_{mul}: For the protocols Π_{sub}, $\Pi_{\mathsf{tree_sum}}$ and Π_{mul}, the computations are relatively straightforward and can be directly implemented using MPC and FHE. For $\Pi_{\mathsf{tree_sum}}^{\mathsf{FHE}}$, it is realized in FHE using homomorphic rotation.

Based on the aforementioned sub-protocols, we can implement two SoftMax protocols, Π_{softmax}-`LAN` and Π_{softmax}-`WAN`, which are adapted to different network environments and are shown in Figure 11 and Figure 12, respectively.

E. Protocol $\Pi_{\mathsf{layernorm}}$

The protocol $\Pi_{\mathsf{LayerNorm}}$ is composed of cast (`cast`), reducesum (`tree_sum`), multiplication (`mul`), addition (`add`), reciprocal (`inv`), reshaping (`reshape`), subtraction (`sub`), and SquaredRoot (`rsqrt`).

1) `cast` and `reshape` operations.: The `cast` operation denotes a data type conversion, such as from `fp16` to `fp32`, with an input of float16 and an output of float32. However, under MPC or FHE ciphertexts, no actual data type conversion occurs. Therefore, the execution times of $\Pi_{\mathsf{cast}}^{\mathsf{FHE}}$ and $\Pi_{\mathsf{cast}}^{\mathsf{MPC}}$ are set to 0. Similarly, the `reshape` operation modifies the tensor shape, e.g.,

$$\%19 = \texttt{tosa.reshape}\ \ \%1\{\texttt{new_shape} = \text{array} < i64 : 1, 1, 768 >\} : (\text{tensor} < 768xf32 >) \rightarrow \text{tensor} < 1x1x768xf32 > . \quad (7)$$

In protocol implementation, FHE represents the tensor via SIMD encoding, while MPC uses a vector representation; in both cases, the operation is performed during initialization. Hence, the execution times of $\Pi_{\mathsf{reshape}}^{\mathsf{FHE}}$ and $\Pi_{\mathsf{reshape}}^{\mathsf{MPC}}$ are also set to 0.

2) sub-protocol Π_{rsqrt}.: The inverse square root can be approximated under both FHE and MPC, but the techniques

Protocol Π_{softmax}-`LAN`

Initialization:
$\mathcal{A}$ and $\mathcal{B}$ generate each private key $sk_{\mathcal{A}}$, $sk_{\mathcal{B}}$ and public key $pk_{\mathcal{A}}$, $pk_{\mathcal{B}}$.
Inputs:
$\mathcal{A}$: holds $\langle \mathbf{X}\rangle_{\mathcal{A}} \in \mathbb{Z}_{2^k}^{d_m * d_h}$
$\mathcal{B}$: holds $\langle \mathbf{X}\rangle_{\mathcal{B}} \in \mathbb{Z}_{2^k}^{d_m * d_h}$ or $[\![\mathbf{X}]\!]_{\mathcal{A}}$
Output: $\mathcal{A}$ and $\mathcal{B}$ get $\langle \mathbf{Y}\rangle_{\mathcal{A}}$, $\langle \mathbf{Y}\rangle_{\mathcal{B}}$, respectively (or $\mathcal{B}$ get $[\![\mathbf{Y}]\!]_{\mathcal{A}}$) for $\mathbf{Y} = softmax(\mathbf{X}) \in \mathbb{Z}_{2^k}^{d_m}$
Execution:
$\mathcal{P}$: Invokes $\Pi_{\mathsf{max}}^{\mathsf{MPC}}$ to obtain $\langle \overline{\mathbf{X}}\rangle_{\mathcal{P}}$, $\mathcal{P} \in \{\mathcal{A}, \mathcal{B}\}$.
$\mathcal{A}$: Encrypt $\langle \overline{\mathbf{X}}\rangle_{\mathcal{A}}$ uses $pk_{\mathcal{A}}$ to get $[\![\langle \overline{\mathbf{X}}\rangle_{\mathcal{A}}]\!]$. Sends $[\![\langle \overline{\mathbf{X}}\rangle_{\mathcal{A}}]\!]$ to $\mathcal{B}$.
$\mathcal{B}$: Receives $[\![\langle \overline{\mathbf{X}}\rangle_{\mathcal{A}}]\!]$; Computes $[\![\overline{\mathbf{X}}]\!]_{\mathcal{A}} = [\![\langle \overline{\mathbf{X}}\rangle_{\mathcal{A}}]\!] \boxplus \langle \overline{\mathbf{X}}\rangle_{\mathcal{B}}$; Computes $[\![\mathbf{M}]\!]_{\mathcal{A}} = [\![\mathbf{X}]\!]_{\mathcal{A}} \boxminus [\![\overline{\mathbf{X}}]\!]_{\mathcal{A}}$. Computes $[\![\mathbf{E}]\!]_{\mathcal{A}} = \Pi_{\mathsf{exp}}^{\mathsf{FHE}}([\![\mathbf{M}]\!]_{\mathcal{A}})$. Generates uniformly random $\langle \mathbf{T}\rangle_{\mathcal{B}} = \mathbf{R} \in \mathbb{Z}_p^{d_m * d_h}$; Computes $[\![\langle \mathbf{T}\rangle_{\mathcal{A}}]\!]_{\mathcal{A}} = [\![\mathbf{E}]\!]_{\mathcal{A}} \boxminus \mathbf{R}$; Sends $[\![\langle \mathbf{T}\rangle_{\mathcal{A}}]\!]_{\mathcal{A}}$ to $\mathcal{A}$.
$\mathcal{A}$: Receives $[\![\langle \mathbf{T}\rangle_{\mathcal{A}}]\!]_{\mathcal{A}}$; Decrypts $[\![\langle \mathbf{T}\rangle_{\mathcal{A}}]\!]_{\mathcal{A}}$ by using $sk_{\mathcal{A}}$ to get $\langle \mathbf{T}\rangle_{\mathcal{A}}$.
$\mathcal{P}$: Invokes $\Pi_{\mathsf{tree_sum}}^{\mathsf{MPC}}$ to obtain $\langle \mathbf{S}\rangle_{\mathcal{P}} = \Pi_{\mathsf{tree_sum}}^{\mathsf{MPC}}(\langle \mathbf{T}\rangle_{\mathcal{P}})$.
$\mathcal{A}$: Encrypts $\langle \mathbf{S}\rangle_{\mathcal{A}}$ by using $pk_{\mathcal{A}}$ to get $[\![\langle \mathbf{S}\rangle_{\mathcal{A}}]\!]_{\mathcal{A}}$. Sends $[\![\langle \mathbf{S}\rangle_{\mathcal{A}}]\!]_{\mathcal{A}}$ to $\mathcal{B}$.
$\mathcal{B}$: Receives $[\![\langle \mathbf{S}\rangle_{\mathcal{A}}]\!]_{\mathcal{A}}$. Computes $[\![\mathbf{S}]\!]_{\mathcal{A}} = [\![\langle \mathbf{S}\rangle_{\mathcal{A}}]\!]_{\mathcal{A}} \boxplus \langle \mathbf{S}\rangle_{\mathcal{B}}$. Invokes Π_{inv} to get $[\![\mathbf{I}]\!]_{\mathcal{A}} = \Pi_{\mathsf{inv}}^{\mathsf{FHE}}([\![\mathbf{S}]\!]_{\mathcal{A}})$. Finally, Invokes $\Pi_{\mathsf{mul}}^{\mathsf{FHE}}$ to obtain $[\![\mathbf{Y}]\!]_{\mathcal{A}} = \Pi_{\mathsf{mul}}^{\mathsf{FHE}}([\![\mathbf{I}]\!]_{\mathcal{A}} \boxdot [\![\mathbf{E}]\!]_{\mathcal{A}})$. [a]

[a] $\odot$ denotes plaintext matrix multiplication, $\boxdot$ denotes ciphertext matrix multiplication, $\boxplus$ denotes ciphertext add, and $\boxminus$ denotes ciphertext subtraction.

Fig. 11: Protocol Π_{softmax}-`LAN` for 2PC.

Protocol Π_{softmax}-`WAN`

Initialization:
$\mathcal{A}$ and $\mathcal{B}$ generate each private key $sk_{\mathcal{A}}$, $sk_{\mathcal{B}}$ and public key $pk_{\mathcal{A}}$, $pk_{\mathcal{B}}$.
Inputs:
$\mathcal{B}$: holds the ciphertext $[\![\mathbf{X}]\!]_{\mathcal{A}}$
Output: $\mathcal{A}$ and $\mathcal{B}$ get $\langle \mathbf{Y}\rangle_{\mathcal{A}}$, $\langle \mathbf{Y}\rangle_{\mathcal{B}}$, respectively (or $\mathcal{B}$ get $[\![\mathbf{Y}]\!]_{\mathcal{A}}$) for $\mathbf{Y} = softmax(\mathbf{X}) \in \mathbb{Z}_{2^k}^{d_m}$
Execution:
$\mathcal{B}$: Invokes $\Pi_{\mathsf{max}}^{\mathsf{FHE}}$ to obtain $[\![\overline{\mathbf{X}}]\!]_{\mathcal{A}}$. Computes $[\![\mathbf{M}]\!]_{\mathcal{A}} = [\![\mathbf{X}]\!]_{\mathcal{A}} \boxminus [\![\overline{\mathbf{X}}]\!]_{\mathcal{A}}$. Computes $[\![\mathbf{E}]\!]_{\mathcal{A}} = \Pi_{\mathsf{exp}}^{\mathsf{FHE}}([\![\mathbf{M}]\!]_{\mathcal{A}})$. Computes $[\![\mathbf{S}]\!]_{\mathcal{A}} = \Pi_{\mathsf{tree_sum}}^{\mathsf{FHE}}([\![\mathbf{E}]\!]_{\mathcal{A}})$. Computes $[\![\mathbf{I}]\!]_{\mathcal{A}} = \Pi_{\mathsf{tree_sum}}^{\mathsf{FHE}}([\![\mathbf{S}]\!]_{\mathcal{A}})$. Computes $[\![\mathbf{Y}]\!]_{\mathcal{A}} = \Pi_{\mathsf{mul}}^{\mathsf{FHE}}([\![\mathbf{I}]\!]_{\mathcal{A}} \boxdot [\![\mathbf{E}]\!]_{\mathcal{A}})$. [a]

[a] $\odot$ denotes plaintext matrix multiplication, $\boxdot$ denotes ciphertext matrix multiplication, and $\boxminus$ denotes ciphertext subtraction.

Fig. 12: Protocol Π_{softmax}-`WAN` for 2PC.

TABLE XI: Π_{gelu} selection under different network settings, where latency is measured in ms and bandwidth in MB.

Network	mul	abs	mul	add	mul	mul	add	mul	mul	add	mul	mul	add	reciprocal	mul	mul	sub	greater_equa	negate	select	add	mul	mul
(2000, 0)	◐	○	●	●	●	●	●	●	●	●	●	●	●	●	●	●	○	○	○	○	○	●	◐
(1000, 10)	◐	○	●	●	●	●	●	●	●	●	●	●	●	●	●	●	○	○	●	○	○	●	◐
(600, 20)	◐	○	●	●	●	●	●	●	●	●	●	●	●	●	●	●	◐	○	●	○	○	●	◐
(400, 30)	◐	○	●	●	●	●	●	●	●	●	●	●	●	●	●	●	◐	○	●	●	●	●	◐
(200, 40)	◐	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	◐	○	●	●	●	●	◐
(100, 80)	◐	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	◐	○	●	●	●	●	◐

Protocol $\Pi_{\mathsf{layernorm}}$

Initialization:
$\mathcal{A}$ and $\mathcal{B}$ generate each private key $sk_{\mathcal{A}}$, $sk_{\mathcal{B}}$ and public key $pk_{\mathcal{A}}$, $pk_{\mathcal{B}}$.
Inputs:
$\mathcal{A}$: holds $\langle \mathbf{X} \rangle_{\mathcal{A}} \in \mathbb{Z}_{2^k}^{d_n * d_m}$
$\mathcal{B}$: holds $\langle \mathbf{X} \rangle_{\mathcal{B}} \in \mathbb{Z}_{2^k}^{d_n * d_m}$
Output: $\mathcal{A}$ get $[\![\mathbf{Y}]\!]_{\mathcal{A}}$ or $\mathcal{B}$ get $[\![\mathbf{Y}]\!]_{\mathcal{B}}$ for $\mathbf{Y} = layernorm(\mathbf{X}) \in \mathbb{Z}_{2^k}^{d_n * d_m}$
Execution:
$\mathcal{P}$: Invokes $\Pi^{\mathsf{MPC}}_{\mathsf{tree_sum}}$ to obtain $\langle \mathbf{E} \rangle_{\mathcal{P}}$, $\mathcal{P} \in \{\mathcal{A}, \mathcal{B}\}$.
$\mathcal{A}$: Encrypt $\langle \mathbf{E} \rangle_{\mathcal{A}}$ and $\langle \mathbf{X} \rangle_{\mathcal{A}}$ uses $pk_{\mathcal{A}}$ to get $[\![\langle \mathbf{E} \rangle_{\mathcal{A}}]\!]_{\mathcal{A}}$ and $[\![\langle \mathbf{X} \rangle_{\mathcal{A}}]\!]_{\mathcal{A}}$. Sends $[\![\langle \mathbf{E} \rangle_{\mathcal{A}}]\!]_{\mathcal{A}}$ and $[\![\langle \mathbf{X} \rangle_{\mathcal{A}}]\!]_{\mathcal{A}}$ to $\mathcal{B}$.
$\mathcal{B}$: Receives $[\![\langle \mathbf{E} \rangle_{\mathcal{A}}]\!]_{\mathcal{A}}$ and $[\![\langle \mathbf{X} \rangle_{\mathcal{A}}]\!]_{\mathcal{A}}$; Computes $[\![\mathbf{X}]\!]_{\mathcal{A}} = [\![\langle \mathbf{X} \rangle_{\mathcal{A}}]\!]_{\mathcal{A}} \boxplus \langle \mathbf{X} \rangle_{\mathcal{B}}$, Computes $[\![\mathbf{E}]\!]_{\mathcal{A}} = [\![\langle \mathbf{E} \rangle_{\mathcal{A}}]\!]_{\mathcal{A}} \boxplus \langle \mathbf{E} \rangle_{\mathcal{B}}$; Invokes $\Pi^{\mathsf{FHE}}_{\mathsf{mul}}$ to obtain $[\![\overline{\mathbf{E}}]\!]_{\mathcal{A}} = \Pi^{\mathsf{FHE}}_{\mathsf{mul}}([\![\mathbf{E}]\!]_{\mathcal{A}}, \frac{1}{d_m})$. Invokes $\Pi^{\mathsf{FHE}}_{\mathsf{sub}}$ to obtain $[\![\mathbf{D}]\!]_{\mathcal{A}} = \Pi^{\mathsf{FHE}}_{\mathsf{sub}}([\![\mathbf{X}]\!]_{\mathcal{A}}, [\![\overline{\mathbf{E}}]\!]_{\mathcal{A}})$. Invokes $\Pi^{\mathsf{FHE}}_{\mathsf{mul}}$ to obtain $[\![\mathbf{Q}]\!]_{\mathcal{A}} = \Pi^{\mathsf{FHE}}_{\mathsf{mul}}([\![\mathbf{D}]\!]_{\mathcal{A}}, [\![\mathbf{D}]\!]_{\mathcal{A}})$. Generates uniformly random $\langle \mathbf{Q} \rangle_{\mathcal{B}} = \mathbf{R}_1 \in \mathbb{Z}_{2^k}^{d_n * d_m}$; Computes $[\![\langle \mathbf{Q} \rangle_{\mathcal{A}}]\!]_{\mathcal{A}} = [\![\mathbf{Q}]\!]_{\mathcal{A}} \boxminus \langle \mathbf{Q} \rangle_{\mathcal{B}}$; Sends $[\![\langle \mathbf{Q} \rangle_{\mathcal{A}}]\!]_{\mathcal{A}}$ to $\mathcal{A}$.
$\mathcal{A}$: Receives $[\![\langle \mathbf{Q} \rangle_{\mathcal{A}}]\!]_{\mathcal{A}}$; Decrypts $[\![\langle \mathbf{Q} \rangle_{\mathcal{A}}]\!]_{\mathcal{A}}$ by using $sk_{\mathcal{A}}$ to get $\langle \mathbf{Q} \rangle_{\mathcal{A}}$.
$\mathcal{P}$: Invokes $\Pi^{\mathsf{MPC}}_{\mathsf{tree_sum}}$ to obtain $\langle \mathbf{T} \rangle_{\mathcal{P}} = \Pi^{\mathsf{MPC}}_{\mathsf{tree_sum}}(\langle \mathbf{Q} \rangle_{\mathcal{P}})$. Computes $\langle \mathbf{V} \rangle_{\mathcal{P}} = \langle \mathbf{T} \rangle_{\mathcal{P}} \times \frac{1}{d_m}$.
$\mathcal{A}$: Encrypts $\langle \mathbf{V} \rangle_{\mathcal{P}}$ by using $pk_{\mathcal{A}}$ to get $[\![\langle \mathbf{V} \rangle_{\mathcal{A}}]\!]_{\mathcal{A}}$. Sends $[\![\langle \mathbf{V} \rangle_{\mathcal{A}}]\!]_{\mathcal{A}}$ to $\mathcal{B}$.
$\mathcal{B}$: Receives $[\![\langle \mathbf{V} \rangle_{\mathcal{A}}]\!]_{\mathcal{A}}$. Computes $[\![\mathbf{V}]\!]_{\mathcal{A}} = [\![\langle \mathbf{V} \rangle_{\mathcal{A}}]\!]_{\mathcal{A}} \boxplus \langle \mathbf{V} \rangle_{\mathcal{B}}$. Invokes $\Pi^{\mathsf{FHE}}_{\mathsf{add}}$ to obtain $[\![\langle \mathbf{V} \rangle]\!]_{\mathcal{A}} = \Pi^{\mathsf{FHE}}_{\mathsf{add}}([\![\langle \mathbf{V} \rangle]\!]_{\mathcal{A}}, \varepsilon)$. Invokes $\Pi^{\mathsf{FHE}}_{\mathsf{rsqrt}}$ to get $[\![\mathbf{R}]\!]_{\mathcal{A}} = \Pi_{\mathsf{rsqrt}}([\![\mathbf{V}]\!]_{\mathcal{A}})$. Invokes $\Pi^{\mathsf{FHE}}_{\mathsf{mul}}$ to obtain $[\![\hat{\mathbf{X}}]\!]_{\mathcal{A}} = \Pi^{\mathsf{FHE}}_{\mathsf{mul}}([\![\mathbf{D}]\!]_{\mathcal{A}}, [\![\mathbf{R}]\!]_{\mathcal{A}})$. Invokes $\Pi^{\mathsf{FHE}}_{\mathsf{mul}}$ to obtain $[\![\mathbf{Z}]\!]_{\mathcal{A}} = \Pi^{\mathsf{FHE}}_{\mathsf{mul}}([\![\hat{\mathbf{X}}]\!]_{\mathcal{A}}, \gamma)$. Finally, Invokes $\Pi^{\mathsf{FHE}}_{\mathsf{add}}$ to obtain $[\![\hat{\mathbf{Y}}]\!]_{\mathcal{A}} = \Pi^{\mathsf{FHE}}_{\mathsf{add}}([\![\mathbf{Z}]\!]_{\mathcal{A}}, \beta)$. [a]

[a] $\odot$ denotes plaintext matrix multiplication, $\boxdot$ denotes ciphertext matrix multiplication, $\boxplus$ denotes ciphertext add, and $\boxminus$ denotes ciphertext subtraction.

Fig. 13: Protocol $\Pi_{\mathsf{layernorm}}$ for 2PC.

differ due to their respective constraints. Both rely on a Goldschmidt-style iteration: $b_i = \tilde{x}\, y_i^2, r_i = \frac{3-b_i}{2}, y_{i+1} = y_i\, r_i$, where $\tilde{x}$ is the appropriately scaled input.

In the FHE setting, following [57], the input $\tilde{x} = x$ is used after a linear polynomial approximation, and the iteration may be preceded by a Newton refinement $y_0 \approx x^{-1/2}$, which converges quadratically. The iteration then produces $h_d \approx x^{-1/2}$ while maintaining numerical stability and low multiplicative depth.

In the MPC setting, following [39], $\tilde{x} = u$ is the normalized mantissa obtained from decomposing $x = 2^k u$ $(1 \le u < 2)$, and the initial estimate $y_0 \approx u^{-1/2}$ is obtained via a lookup table on the top bits of u. After the Goldschmidt iteration, exponent compensation recovers the original scale: $y = y_t \cdot 2^{-k/2} \approx x^{-1/2}$.

Based on the above sub-protocols, the complete protocol $\Pi_{\mathsf{layernorm}}$ can be constructed, as illustrated in Fig. 13.

It is worth noting that the computation generated by MLIR contains three `reduce_sum` operations, as shown in Table VI, whereas only two are actually necessary. The extra one arises from MLIR providing reciprocal calculations for 32-bit and 64-bit constants ($\frac{1}{d_m}$), which are used to compute the mean. Therefore, this protocol could be further optimized to eliminate at least one `reduce_sum` and one `mul`.

F. Protocol Π_{gelu}

The protocol Π_{gelu} is composed of multiplication (`mul`), absolute value (`abs`), addition (`add`), reciprocal (`inv`), greater_equal (`gt`), negation (`negate`), selection (`if_else`), and subtraction (`sub`).

1) sub-protocol Π_{abs}*:* The absolute value of a vector can be computed under both FHE and MPC. In the FHE setting, following Nexus [30], the sign of each element is first evaluated using a homomorphic sign function, $\mathrm{sgn}(x)$, implemented via polynomial approximation. The input is then multiplied by the sign: $\mathrm{abs}(x) = x \cdot \mathrm{sgn}(x)$, which produces a ciphertext encoding the absolute value.

In the MPC setting, following CryptFlow2 [29], the most significant bit (MSB) of each element is first extracted to determine the sign. A negated copy of the input is then conditionally selected based on the MSB using a secure multiplexer:

$$\mathrm{abs}(x_i) = \begin{cases} -x_i, & \text{if MSB}(x_i) = 1, \\ x_i, & \text{otherwise.} \end{cases}$$

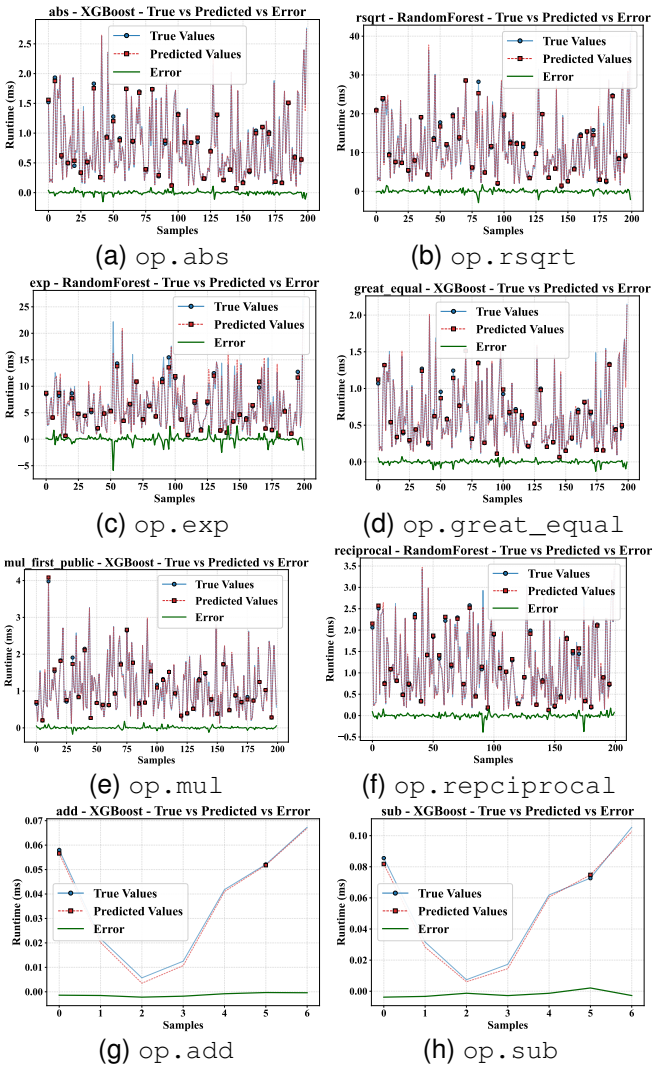
This approach leverages basic MPC primitives, such as bit decomposition and `if_else`, to compute the absolute value securely.

2) sub-protocol $\Pi_{\mathsf{if_else}}$*:* Under FHE, it can be evaluated arithmetically as $\mathrm{select}(c, x, y) = c \cdot x + (1 - c) \cdot y$, where c, x, y are ciphertexts. In the MPC setting, following CryptFlow2 [29], the condition c is a Boolean value, and selection is performed via secure bitwise multiplexing:

$$\mathrm{select}(c_i, x_i, y_i) = \begin{cases} x_i, & \text{if } c_i = 1, \\ y_i, & \text{otherwise.} \end{cases}$$

3) sub-protocol Π_{gt}*:* The secure greater_than can be implemented under both FHE and MPC. In the FHE setting, following Nexus [30], a polynomial approximation of the sign function is applied $\mathrm{GT}(x, y) \approx (\mathrm{sgn}(x - y) + 1)/2$. In the MPC setting, following CryptFlow2, we have $\mathrm{GT}(x, y) = \mathrm{MSB}(y - x)$ where MSB = 1 indicates $x > y$.

Based on the aforementioned sub-protocols, we can implement the protocols Π_{gelu}-`LAN` and Π_{gelu}-`WAN`, such as Π_{softmax}-`LAN` and Π_{softmax}-`WAN`.



(a) `op.abs` (b) `op.rsqrt`

(c) `op.exp` (d) `op.great_equal`

(e) `op.mul` (f) `op.repciprocal`

(g) `op.add` (h) `op.sub`

Fig. 14: (MPC) Cost model performance of the commonly used operators.

G. Experimental Setup

1) The error analysis of the trained cost model: Figure 14 presents the error analysis of the trained cost model on 200 randomly selected MPC test samples, exhibiting a similarly stable error trend as observed in the FHE evaluation.

2) Performance breakdown of $\text{BERT}_{\text{BASE}}$ and GPT-2: Table XIII breaks down the NETADAPT inference runtime and communication for $\text{BERT}_{\text{BASE}}$. The input to the model consists of 128 tokens. Under the LAN setting, the GeLU function and the first matrix multiplication make up about 76% of the communication costs. In the WAN setting, the activation function is the most expensive operation, which takes 81% of the total inference time. This indicates that the bottleneck of private $\text{BERT}_{\text{BASE}}$ inference lies in the nonlinear functions.

In addition, we evaluate the runtime and communication cost of the GPT-2 model under LAN settings; the results are reported in Table XII.

TABLE XII: Breakdown of GPT-2 from NETADAPT.

Operation (GPT-2)	Shape	Time (s) LAN	Comm.
Masked-Multi-Attention	128×768×64 ×12	11.948	2.13
LayerNorm	128×768	0.96	0.18
FFN1	128×768×3072	2.65	0.105
GELU	128×3072	3.99	0.907
FFN2	128×3072×768	2.43	0.157
LayerNorm	128×768	0.96	0.18
Total (layers: =1)	–	22.938	1826.2
Total (layers: =12)	–	275.4 s	21.40 (GB)

TABLE XIII: Performance breakdown of ($\text{BERT}_{\text{BASE}}$). Under LAN={2 Gbps, 0 ms}, WAN={100 Mbps, 80 ms} network environment settings.

Operation		Runtime (s)		Comm. (MB)	
		LAN	WAN	LAN	WAN
MatrixMul	$\left(\mathbb{Q}^{d_{128\times768\times768}}\right)\times3$	1.76	7.50	1479.34	84.72
MatrixMul	$\left(\mathbb{Q}^{d_{128\times64\times128}}\right)\times12$	0.61	8.26	322.96	0
SoftMax	$\left(\mathbb{Q}^{d_{128\times128}}\right)\times12$	5.05	63.12	5.30	0
MatrixMul	$\left(\mathbb{Q}^{d_{128\times128\times64}}\right)\times12$	0.92	7.90	322.98	26.85
MatrixMul	$\mathbb{Q}^{d_{128\times768\times768}}$	0.87	3.91	54.18	28.24
LayerNorm	$\mathbb{Q}^{d_{128\times768}}$	0.96	3.32	12.79	12.79
MatrixMul	$\mathbb{Q}^{d_{128\times768\times3072}}$	2.65	8.19	107.29	107.29
GeLU	$\mathbb{Q}^{d_{128\times3072}}$	4.00	105.22	1798.23	884.20
MatrixMul	$\mathbb{Q}^{d_{128\times3072\times768}}$	2.42	6.49	161	161
LayerNorm	$\mathbb{Q}^{d_{128\times768}}$	0.96	3.32	12.79	12.79
Total		20.2	217.23	4276.86	1317.88
Total(12-layers)		242.4	2606.76	50.01 (GB)	15.44 (GB)