SecMoE: Communication-Efficient Secure MoE Inference via Select-Then-Compute

Bowen Shen[1,2], Yuyue Chen[1], Peng Yang[1], Bin Zhang[2*], Xi Zhang[3], Zoe L. Jiang[1,4*]

[1]School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen, Shenzhen, China
[2]Department of New Networks, Pengcheng Laboratory, Shenzhen, China
[3]College of Management and Economics, Tianjin University, Tianjin, China
[4]Guangdong Key Laboratory of New Security and Intelligence Technology, Shenzhen, China
{shenbowen, chenyuyue, stuyangpeng}@stu.hit.edu.cn, bin.zhang@pcl.ac.cn, jackyzhang@tju.edu.cn, zoeljiang@hit.edu.cn

Abstract

Privacy-preserving Transformer inference has gained attention due to the potential leakage of private information. Despite recent progress, existing frameworks still fall short of practical model scales, with gaps up to a hundredfold. A possible way to close this gap is the Mixture of Experts (MoE) architecture, which has emerged as a promising technique to scale up model capacity with minimal overhead. However, given that the current secure two-party (2-PC) protocols allow the server to homomorphically compute the FFN layer with its plaintext model weight, under the MoE setting, this could reveal which expert is activated to the server, exposing token-level privacy about the client's input. While naively evaluating all the experts before selection could protect privacy, it nullifies MoE sparsity and incurs the heavy computational overhead that sparse MoE seeks to avoid. To address the privacy and efficiency limitations above, we propose a 2-PC privacy-preserving inference framework, SecMoE. Unifying per-entry circuits in both the MoE layer and piecewise polynomial functions, SecMoE obliviously selects the extracted parameters from circuits and only computes one encrypted entry, which we refer to as Select-Then-Compute. This makes the model for private inference scale to 63× larger while only having a 15.2× increase in end-to-end runtime. Extensive experiments show that, under 5 expert settings, SecMoE lowers the end-to-end private inference communication by 1.8∼7.1× and achieves 1.3∼3.8× speedup compared to the state-of-the-art (SOTA) protocols.

Introduction

Transformers (Vaswani et al. 2017) have significantly enhanced machine learning capabilities across a range of tasks. In model scaling, the Mixture of Experts (MoE) architecture has emerged as a powerful technique in Transformer-based models, particularly in natural language processing (Jacobs et al. 1991a). By dynamically selecting a subset of experts for each input token, MoE significantly reduces computational overhead while maintaining high model capacity.

However, despite great advantages, deploying the MoE models in an untrusted environment raises privacy concerns. On the one hand, the server that owns the model weights naturally expects privacy protection of its model weight because training a MoE model requires significant investments in both financial and computational resources. On the other hand, MoE inference requires clients to upload their prompts, which may contain sensitive user data, such as personal health records and biometric information. As a result, the server requires that client learn nothing about the model parameters beyond the inference outputs, and the client requires that the server learn nothing about client's input.

To ensure privacy for Transformer inference, several works have focused on the realm of Privacy-Preserving Machine Learning (PPML), using secure multiparty computation (MPC) (Wagh, Gupta, and Chandran 2019; Srinivasan, Akshayaram, and Ada 2019; Li et al. 2023; Huang et al. 2022; Rathee et al. 2020; Mohassel and Rindal 2018; Hao et al. 2022; Liang et al. 2024; Zeng et al. 2023; Dong et al. 2023; Pang et al. 2024; Lu et al. 2023; Li et al. 2024; Kei and Chow 2025). However, those studies are mainly designed for relatively small-scale Transformer models, like BERT (Kenton and Toutanova 2019) and GPT-2 (Cohen and Gokaslan 2020). Some works (Dong et al. 2023; Lu et al. 2023) also test on larger models like LLaMA-7B (Touvron et al. 2023), but there is still a certain gap with the scale of parameters implemented in plaintext. Moreover, simply increasing the number of Transformer blocks to scale up the parameters leads to linearly growing performance overheads in MPC-based implementations.

To solve the scalability problems mentioned above, it is essential to develop privacy-preserving protocols for model scaling. Previous works (Hao et al. 2022; Pang et al. 2024; Lu et al. 2023; Li et al. 2024) compute linear layers with homomorphic encryption with the plaintext weight matrix held by the server. While sparse MoE requires the activation of only k experts per token, and when plaintext expert weights are chosen before computation, the server will learn which weight matrices are accessed, leaking the client's private token information. The simple idea is to obliviously select the experts after evaluating all the experts securely with plaintext weight matrices. However, this method will nullify the MoE layer's sparsity, incurring the heavy computational overhead that the sparse MoE seeks to avoid.

To address these privacy and efficiency problems, in this work, we adopt the MoE architecture to serve as the

*Corresponding authors.

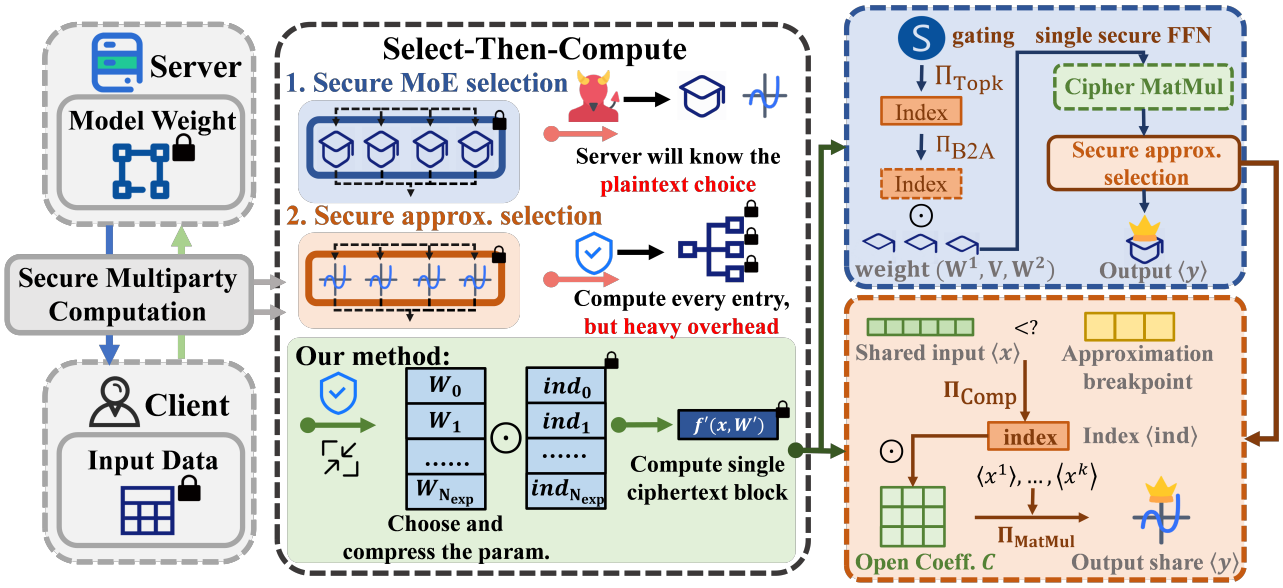

Figure 1: An overview of SecMoE, deploying Select-Then-Compute in Secure MoE FFN and secure approximation selection.

foundation of our approach. As mentioned above, naively deploying existing 2-PC protocols to sparse MoE creates new privacy issues. Therefore, we propose a novel privacy-preserving inference framework for MoE architectures via Select-Then-Compute, named SecMoE. Select-Then-Compute, including the selection phase and the compute phase, works on the functions that have multiple computing entries. The selection phase modifies each computing entry until all entries share the same computing circuit and extracts each entry's parameters as the choices. Next, the parameters will be obliviously selected by the ciphertext vector. Then, in the compute phase, only one encrypted entry will be computed with the chosen parameter. We design the secure sparse MoE and secure polynomial selection using Select-Then-Compute, as illustrated in Figure 1. Specifically, in the MoE layer, our approach leverages lattice-based additive homomorphic encryption, combined with carefully designed selection bit strings, to implement secure sparse MoE. We encrypt the expert selection process in such a way that only the chosen experts' computations are performed, while keeping the selection private from the server. To further boost the model efficiency, we also form the nonlinear function piecewise polynomial approximation into the Select-Then-Compute. Specifically, we transform piecewise polynomials by padding all segments into uniform computing entries. Then, entries are obliviously aggregated by the encrypted selection vector so that only the activated segment is computed in the ciphertext.

In summary, we propose a new secure 2-PC inference framework SecMoE, and contributions are summarized as:

- We propose a new secure sparse MoE layer via Select-Then-Compute. Our protocol obliviously selects the extracted experts' parameter in the selection phase and obtains output through single expert evaluation in the compute phase. This preserves the MoE's sparsity and achieves almost linear communication with the increasing number of experts.
- For nonlinear layers, we redesign secure piecewise polynomial evaluation via Select-Then-Compute. Our protocol secretly selects the open coefficient matrix and performs low-degree polynomial evaluation in ciphertext. This achieves lower error in accuracy and higher efficiency in communication.
- We combine the protocols above as SecMoE and evaluate the performance using 2 MoE models in 5 expert settings under both LAN and WAN networks. Compared with the SOTA PPML frameworks Iron(Hao et al. 2022) and BumbleBee(Lu et al. 2023), SecMoE reduces the communication of the private inference by 1.8∼29.8× and improves end-to-end performance by 1.3∼16.1×.

Preliminaries

Threat Model

Our framework operates in a two-party setting, where the client C possesses a private input, and the server S holds the model weight. It ensures security against a semi-honest adversary, where both parties follow the protocol's rules but attempt to extract unauthorized information from the protocol. Many PPML works use same setting(Juvekar, Vaikuntanathan, and Chandrakasan 2018; Lu et al. 2023), and details refer to the Appendix.

Notation

Let $\mathbb{Z}_{2^\ell}$ denote the ring of integers modulo 2^ℓ. For any positive integer n, let $[n] = \{0, 1, \ldots, n-1\}$ denote its index set. For the homomorphic encryption (HE), we define $\mathbb{A}_{N,2^\ell} = \mathbb{Z}_{2^\ell}[X]/(X^N + 1)$, where N is an integer with a power of two. The elements in $\mathbb{A}_{N,2^\ell}$ are polynomials of degree at most $N-1$. And $[\![\mathrm{M}]\!]$ denotes the homomorphic encryption ciphertext with its plaintext M. In the

MoE Layer, we use W_i to represent the plain weight matrix in the i-th expert. Additive arithmetic secret shares used in our work are represented as $\langle\cdot\rangle$ and boolean shares as $\langle\cdot\rangle^b$. $\Pi_{\mathrm{MUX}}(\langle a\rangle^b, \langle b\rangle)$ represents $\langle a?b:0\rangle$. To illustrate clearly, we set $\Pi_{\mathrm{Mul}}(a,b) = \Pi_{\mathrm{trunc}}(\Pi_{\mathrm{MUL}}(a,b))$, where Π_{MUL} and Π_{trunc} represent 2-PC arithmetic multiplication and secure truncation protocol followed by BumbleBee (Lu et al. 2023).

Cryptographic Primitives

Additive Secret Sharing. In our two-party setting, the nonlinear layers are implemented through the additive secret sharing over the ring $\mathbb{Z}_{2^\ell}$. For example, to secretly share the client C's value $x \in \mathbb{Z}_{2^\ell}$, C uniformly samples a random share $\langle x\rangle_s \in \mathbb{Z}_{2^\ell}$ and sends it to server S as its secret share. S sets its own share as $\langle x\rangle_c = x - \langle x\rangle_s$. The pointwise secure multiplication with shares needs oblivious transfer(Yang et al. 2020)-based Beaver Triples(Beaver 1996).

Lattice-based Additive Homomorphic Encryption. We use the lattice-based additive HE scheme (Lu et al. 2023) to build our linear layers. Homomorphic encryption allows one party to perform computations on the encrypted data of the other party without the need for the decryption key. This HE scheme (Lu et al. 2023) encodes a plaintext vector $x \in (\mathbb{Z}_{2^\ell})^N$ into a plaintext polynomial $\hat{x} \in \mathbb{A}_{N,2^\ell}$, and then $\hat{x}$ is encrypted to a ciphertext $\mathrm{Enc}(\hat{x}) = [\![x]\!] \in \mathbb{A}^2_{N,q}$ where q is a ciphertext modulus. Therefore, the inputs in our linear layers are $\langle x\rangle_c, \langle x\rangle_s \in \mathbb{Z}_{2^\ell}$ with dimension $k \times m$. And sever holds the plaintext weight $\mathrm{W} \in \mathbb{Z}_{2^\ell}$ with dimension $m \times n$. The encoding functions π_L and π_R will encode inputs into polynomials $\hat{x}$ and $\hat{w}$: $\hat{x} = \pi_L(x) : \hat{x}\big[imn + (m-1) - j\big] = x_{i,j}, i \in [k], j \in [m], \hat{w} = \pi_R(\mathrm{W}) : \hat{w}\big[jm + i\big] = \mathrm{W}_{i,j}, i \in [m], j \in [n]$.

Transformer Architecture

Attention Layer. The attention Layer (Vaswani et al. 2017) can be described as mapping a query key $\mathrm{X_Q}$ and a set of key-value pairs ($\mathrm{X_K}$,$\mathrm{X_V}$) to a weighted sum, where the weights are calculated from the query key and the corresponding key values, and the formal expression is as shown in the following formula: $\mathrm{Attention}(\mathrm{X_Q}, \mathrm{X_K}, \mathrm{X_V}) = \mathrm{Softmax}(\frac{\mathrm{X_Q X_K^T}}{\sqrt{\mathrm{d}}})\mathrm{X_V}$, where d is the hidden dimension. $\mathrm{X_Q}$, $\mathrm{X_K}$, and $\mathrm{X_V}$ are different linear projections of the input, satisfying: $\mathrm{X_Q} = \mathrm{W_Q} \cdot \mathrm{X}, \mathrm{X_K} = \mathrm{W_K} \cdot \mathrm{X}, \mathrm{X_V} = \mathrm{W_V} \cdot \mathrm{X}$, $\mathrm{W_Q}, \mathrm{W_K}, \mathrm{W_V}$ are the pre-trained weight matrices. Multi-head attention extends the above mechanism to parallel attention layers.

FeedForward Network (FFN). Our work mainly focuses on FeedForward Network with GeGLU functions. It consists of two fully connected layers with an activation function in between:

$$\mathrm{FeedForward}(x) = \mathrm{W}^2(\sigma(\mathrm{W}^1 x) \otimes \mathrm{V}x), \tag{1}$$

where V, W^1 and W^2 are parameter matrices. As for the activation σ, we apply GeLU function. Most large language models still use ReLU in the FFN (e.g., Switch Transformer Base and Large (Fedus, Zoph, and Shazeer 2022)). GPT models replace it with the smoother activation GeLU and GeGLU. The LLaMA series goes further, adopting the gated SwiGLU activation to boost expressiveness at minimal extra cost (Touvron et al. 2023).

Mixture-of-Experts (MoE). In large language models employing transformer architectures, the mixture-of-experts (MoE) layer is composed of a collection of N_{exp} expert sub-networks $\mathrm{FFN}_0, ..., \mathrm{FFN}_{N_{\mathrm{exp}}-1}$, complemented by a gating network G. The gating network G allocates the input to the most suitable expert sub-networks (Jacobs et al. 1991b; Jordan and Jacobs 1994; Collobert, Bengio, and Bengio 2001). The MoE layer is strategically integrated to replace the FFN within each transformer block, reducing the computational complexity of the FFN with model scaling. Based on the strategy of gating network G, MoE layers can be categorized into two types: dense MoE and sparse MoE, and we mainly focus on the sparse MoE.

Sparse MoE is proposed by Shazeer (Shazeer et al. 2017), selectively activating a subset of expert sub-networks during each block. By calculating a weighted aggregation of outputs from the top-k experts, sparse MoE reduces the significant computational overhead compared to dense MoE:

$$\mathrm{M_{Sparse}}(x) = \sum_{i=1}^{N_{\mathrm{exp}}} \mathrm{Softmax}(\Pi_{\mathrm{Topk}}\left(g(x)\right))_i \, \mathrm{FFN}_i(x), \tag{2}$$

$$\Pi_{\mathrm{Topk}}(g(x)_i) = \begin{cases} g(x)_i, & \text{if } g(x)_i \in g(x)^{K_{\mathrm{exp}}}, \\ -\infty, & \text{else.} \end{cases}, \tag{3}$$

where $g(x)$ represents the input of the Softmax operation and $g(x)^{K_{\mathrm{exp}}}$ is the top-k elements of it. The hyperparameter K_{exp} is the number of experts being selected, which is set as 1 in our work.

Secure Sparse MoE Layer

In this section, we first analyze the privacy problem in the secure MoE layer. Then, we propose the secure sparse MoE protocol via Select-Then-Compute. Further, we form the secure GeLU as a Select-Then-Compute function and propose its design in sparse MoE. We also analyze complexity in both secure sparse MoE and secure GeLU. We defer the correctness and the security proofs to the Appendix.

Secure Sparse MoE Protocol

The sparse MoE and the dense MoE model differ significantly in terms of their computational strategies. While dense MoE applies all available experts during each inference pass, which we do not give special optimizations, sparse MoE is designed to dynamically select only a subset of experts based on the input (Fedus, Zoph, and Shazeer 2022), thus ensuring computational efficiency and scalability. The primary advantage of sparse MoE is its ability to increase the number of model parameters significantly, without the corresponding increase in computational overhead, by activating only K_{exp} experts using the gating network G.

However, in the MPC setting, secure sparse MoE cannot be directly derived from secure dense MoE. If the secure dense MoE approach were adopted unchanged, all experts

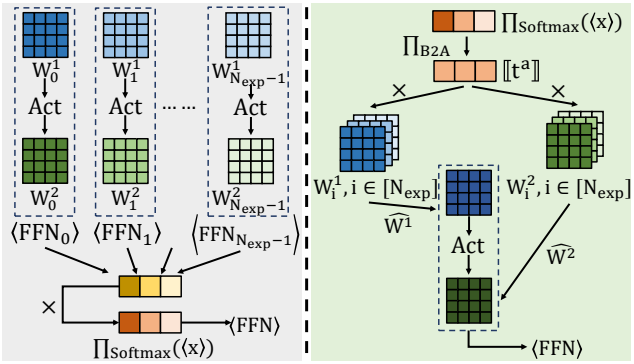

Figure 2: Illustration of secure MoE layer. Comparison between a naive privacy-preserving MoE (left) and our proposed secure sparse MoE (right). The baseline evaluates all experts and aggregates their outputs via Softmax, leading to high overhead. In contrast, our method securely selects top experts and only evaluates one expert, significantly reducing computation and communication cost.

would be evaluated in parallel to maintain the expert selection oblivious. This strategy suffers significant inefficiency, as it leads to the unnecessary computation of all experts, which would incur at least additional $N_{\text{exp}} - K_{\text{exp}}$ experts' computational and communication overheads, which goes against the original intent of sparse MoE. To address the challenges above, we propose a modified approach to implement the secure sparse MoE via Select-Then-Compute, shown in Figure 2, and we illustrate it in the next paragraph.

Select-Then-Compute in Secure Sparse MoE Protocol. In the secure sparse MoE Protocol, the client and server each hold additive secret shares of the input $\langle x \rangle$, and the server additionally holds all experts' plaintext weight matrix sets $(\mathrm{W}_i^1 \in \mathbb{Z}_{2^\ell}^{m \times n}, \mathrm{V}_i \in \mathbb{Z}_{2^\ell}^{m \times n}, \mathrm{W}_i^2 \in \mathbb{Z}_{2^\ell}^{n \times m}), i \in [N_{\text{exp}}]$.

Select-Then-Compute has two phases: the selection phase and the compute phase. The entry in the secure sparse MoE protocol denotes each expert FFN. Noticed that each expert has the same components, and therefore, MoE experts are naturally unified. In the selection phase, SecMoE leverages the advantage of homomorphic encryption's local communication-free advantages. It only requires sharing a N_{exp}-length selection vector, enabling oblivious selection of the encrypted weights. This significantly reduces online communication costs compared to previous works, which is shown in our experiments. Next, in the compute phase, SecMoE computes similarly to previous work, with the key difference being that the output of the selection phase is a homomorphic ciphertext to prevent leaking choice to the server, whereas prior approaches (Hao et al. 2022; Li et al. 2024; Lu et al. 2023) typically used plaintext. While this introduces additional computational overhead of ciphertext-ciphertext matrix multiplication, as noted in Protocol 1, we only invoke this multiplication operator twice, making it more efficient compared to computing the remaining $N_{\text{exp}} - 1$ expert FFNs. The whole protocol is described in Protocol 1.

Protocol 1: Secure Sparse MoE $\Pi_{\text{SparseMoE}}$

Input: The client C and the server S each holds $\langle x \rangle_c$ and $\langle x \rangle_s$, where $x = \langle x \rangle_c + \langle x \rangle_s$. The server holds the FFN weight matrices sets $(\mathrm{W}_i^1 \in \mathbb{Z}_{2^\ell}^{m \times n}, \mathrm{V}_i \in \mathbb{Z}_{2^\ell}^{m \times n}, \mathrm{W}_i^2 \in \mathbb{Z}_{2^\ell}^{n \times m}), i \in [N_{\text{exp}}]$, where m, n and N_{exp} are the model, hidden dimension and number of experts, with the secret key sk.

Output: C and S obtain $\langle y \rangle_c$ and $\langle y \rangle_s$ respectively, where $\Pi_{\text{SparseMoE}}(x) = \langle y \rangle_c + \langle y \rangle_s$.

{**Selection Phase**}

1: C and S compute Π_{Topk} to get top-k large values and output secret-shared index $\langle \text{SortVal} \rangle$.
2: C and S compute $\Pi_{\text{onehot}}(\langle \text{SortVal} \rangle, K_{\text{exp}})$ outputs a secret-shared boolean vector $t^b \in \{0,1\}^{N_{\text{exp}}}$ with $t^b_{\text{index}} = 1$, else 0.
3: C and S compute Π_{B2A} converting the secret input t^b from Boolean to arithmetic form (B2A) and outputs t^a.
4: C encrypts $[\![t^a_c]\!] := \text{Enc}(t^a_c)$ and sends to S. On receiving the cipher, S computes $[\![t^a]\!] := [\![t^a_c]\!] + t^a_s$.
5: S computes point-wise secure multiplication $\Pi_{\text{Mul}}([\![t^a]\!], \mathrm{W}_i^1)$ yielding $[\![\mathrm{W}_r^1]\!] = \sum_{i=0}^{N_{\text{exp}}-1} \mathrm{W}_i^1 \cdot [\![t^a]\!]$, and similar for $\Pi_{\text{Mul}}([\![t^a]\!], \mathrm{V}_i), \Pi_{\text{Mul}}([\![t^a]\!], \mathrm{W}_i^2)$.

{**Compute Phase**}

6: C encrypts $[\![\langle x \rangle_c]\!] := \text{Enc}(\langle x \rangle_c)$ and sends to S. After receiving, S local computes $[\![x]\!] = [\![\langle x \rangle_c]\!] + \langle x \rangle_s$.
7: S computes $[\![\mathrm{W}_r^1]\!] \cdot [\![x]\!]$ and $[\![\mathrm{V}_r]\!] \cdot [\![x]\!]$, and $[\![\mathrm{W}_r^1 \cdot x - \mathrm{R}^1]\!], [\![\mathrm{V}_r \cdot x - \mathrm{R}^V]\!]$ with random masks $\mathrm{R}^1, \mathrm{R}^V$.
8: S sends $[\![\mathrm{W}_r^1 \cdot x - \mathrm{R}^1]\!]$ and $[\![\mathrm{V}_r \cdot x - \mathrm{R}^V]\!]$ to C and sets $(\langle x^{\text{GLU}} \rangle_s, \langle x^V \rangle_s) = (\mathrm{R}^1, \mathrm{R}^V)$. C decrypts and obtains $(\langle x^{\text{GLU}} \rangle_c, \langle x^V \rangle_c) = (\mathrm{W}_r^1 \cdot x - \mathrm{R}^1, \mathrm{V}_r \cdot x - \mathrm{R}^V)$.
9: C and S compute $\langle \text{act} \rangle := \Pi_{\text{GeLU}}(\langle x^{\text{GLU}} \rangle)$ and obtains $\langle \text{act} \rangle_c$ and $\langle \text{act} \rangle_s$.
10: C and S compute point-wise multiplication $\Pi_{\text{Mul}}(\langle \text{act} \rangle_i, \langle x^V \rangle_i)$ to obtain $\langle \text{GLU} \rangle_c$ and $\langle \text{GLU} \rangle_s$.
11: C encrypts $[\![\langle \text{GLU} \rangle_c]\!] := \text{Enc}(\text{GLU}_c)$ and sends to S. S locally computes $[\![\text{GLU}]\!] = [\![\langle \text{GLU} \rangle_c]\!] + \langle \text{GLU} \rangle_s$.
12: S computes the cipher-cipher multiplication $[\![\mathrm{W}_r^2]\!] \cdot \text{GLU}$ with $[\![\text{GLU}]\!]$ and $[\![\mathrm{W}_r^2]\!]$.
13: S computes $[\![\mathrm{W}_r^2 \cdot \text{GLU} - \mathrm{R}^2]\!]$ with random R^2, and sends to C. S sets R^2 as the $\langle y \rangle_s$.
14: C decrypts $[\![\mathrm{W}_r^2 \cdot \text{GLU} - \mathrm{R}^2]\!]$ and sets $\mathrm{W}_r^2 \cdot \text{GLU} - \mathrm{R}^2$ as its output share $\langle y \rangle_c$.

Secure Sparse MoE Complexity Analysis. At the layer level, our approach eliminates redundant computation for at least $N_{\text{exp}} - 1$ non-selected experts by executing the selection phase, preserving sparsity under the MoE setting. The resulting overhead arises primarily from the ciphertext matrix multiplications in the compute phase. In practice, these costs are relatively small compared to the computation saved by redundant $N_{\text{exp}} - 1$ inactive experts.

At the expert level, compared with BumbleBee, SecMoE requires the conversion protocol Π_{B2A} on the N_{exp}-dimensional one-hot vector with $cost_{\text{B2A}}$, the communication for $[\![\langle t^a_c \rangle]\!]$ and local Π_{MatMul}. In the compute phase, un-

like BumbleBee computes $\mathrm{W}\cdot[\![x]\!]$, SecMoE requires three cipher multiplications $[\![\mathrm{W}_r^1]\!]\cdot[\![x]\!]$, $[\![\mathrm{V}_r]\!]\cdot[\![x]\!]$, and $[\![\mathrm{W}_r^2]\!]\cdot[\![\mathrm{GLU}]\!]$. However, compared to the redundant computation of all experts dominating the costs, the additional cost introduced by ciphertext multiplication is fixed and relatively small. Therefore, this optimization becomes increasingly effective as the number of experts grows, by up to 29.8×.

Secure GeLU Protocol

Select-Then-Compute in Secure Polynomial Evaluation. Previous works adopt piecewise polynomial approximation to calculate the GeLU function. Bumblebee (Lu et al. 2023) approximates the GeLU function with 4-piece polynomials at most degree-6. Nimbus (Li et al. 2024) uses 3-piece polynomials at most degree-2, boosting further efficiency. However, these works primarily optimize polynomial-based expressions at the design stage, yet uniformly rely on the same secure polynomial evaluation paradigm during computation, illustrated in the Appendix. To address this inefficient communication problem, we describe the secure polynomial evaluation as a Select-Then-Compute process, and its entry denotes each segmented polynomial.

Before the Select-Then-Compute process, we first describe the preparation for polynomials. To form the approximation of the GeLU gate inside each expert, we follow the approximation interval $[-5, 3]$ adopted by Bumblebee (Lu et al. 2023). Within this interval, we fit piecewise quadratic polynomials using `numpy.polynomial`, solving a least-squares problem in Chebyshev space to obtain the coefficients of two polynomials, achieving maximum absolute error of 1.2×10^{-2} and mean absolute error of 1.7×10^{-3} over the evaluation range. The values smaller than -5 are mapped to 0, while values larger than 3 are mapped to the x, as shown in Equation 4:

$$\mathrm{GeLU}(x) = \begin{cases} 0 & x \in (-\infty, -5] \\ P_1(x) & x \in (-5, -3] \\ P_2(x) & x \in (-3, -1] \\ P_3(x) & x \in (-1, 1] \\ P_4(x) & x \in (1, 3] \\ x & x \in (3, \infty) \end{cases} \tag{4}$$

and $P_1(x), P_2(x), P_3(x), P_4(x)$ are degree-2 polynomials.

In the selection phase, to evaluate the piecewise polynomials in a pre-chosen manner, we first collect the polynomial coefficients into a matrix, where the row index $i \in \{1, \ldots, m_{\text{seg}}\}$ enumerates the m_{seg} segments and the column index $j \in \{1, \ldots, n_{\text{seg}}\}$ enumerates the coefficients from the highest to the constant term. We pad each entry with 0 until it reaches the largest power among all segments. The resulting coefficient details are illustrated in the Appendix. Given an input $\langle x\rangle$, we first compute secure comparisons between $\langle x\rangle$ and each open plaintext breakpoint $b_i, i \in \{1, \ldots, m_{\text{seg}}-1\}$, to obtain a boolean one-hot segment selector. Therefore, in the selection phase, the hybrid coefficient row corresponding to x can be retrieved by a single masked matrix–vector product. In the compute phase, given that we unify each entry power, the final value is then obtained by securely evaluating one polynomial with the largest power.

Protocol 2: Secure GeLU Π_{GeLU}

Input: The client C and server S each holds $\langle x\rangle_c$ and $\langle x\rangle_s$, where $x = \langle x\rangle_c + \langle x\rangle_s$.

Output: Client and server obtain $\langle y\rangle_c$ and $\langle y\rangle_s$ respectively, where $\Pi_{\text{GeLU}}(x) = \langle y\rangle_c + \langle y\rangle_s$. $c_i, i \in [m_{\text{seg}}]$ is the plaintext polynomial coefficients open to both parties, where m_{seg} represents rows of coefficients.

1: C and S compute $\langle \text{ind}\rangle^b$ and each obtain boolean vector share $\langle \text{ind}\rangle_0^b$ and $\langle \text{ind}\rangle_1^b$. $\langle \text{ind}\rangle^b$ is represented as $(\Pi_{\text{comp}}\{x < b_1\}, ..., \Pi_{\text{comp}}\{x < b_{m_{\text{seg}}-1}\})^\top$.

2: C and S compute $\langle c_r\rangle$ and each obtain arithmetic vector shares $\langle c_r\rangle_c$ and $\langle c_r\rangle_s$. $\langle c_r\rangle$ is calculated as $(\Pi_{\text{MUX}}(\langle \text{ind}^\top\rangle^b, c_0), ..., \Pi_{\text{MUX}}(\langle \text{ind}^\top\rangle^b, c_{n_{\text{seg}}})) = (\langle c_r\rangle_0, \langle c_r\rangle_1, \ldots, \langle c_r\rangle_{n_{\text{seg}}})$.

3: C and S compute $\langle x^2\rangle := \Pi_{\text{Mul}}(x, x)$ and obtain $\langle x^2\rangle_c, \langle x^2\rangle_s$.

4: For our approximation, the coefficient rows are $k = 2$, and C, S compute $\langle y\rangle := \Pi_{\text{Mul}}(\langle x^2\rangle, \langle c_r\rangle_0) + \Pi_{\text{Mul}}(\langle x\rangle, \langle c_r\rangle_1) + \langle c_r\rangle_2$. C and S obtain $\langle y\rangle_c, \langle y\rangle_s$.

Secure GeLU Complexity Analysis. The GeLU is approximated on m_{seg} intervals by a quadratic per interval with open coefficient matrix. We then estimate per-element costs with batch, which scales communication but does not increase rounds. Let $cost_{\text{mul}}$, $cost_{\text{mux}}$, and $cost_{\text{comp}}$ be the communication cost for secure arithmetic multiplication, boolean-arithmetic multiplication, and the cost of secure comparison at fixed bit precision.

In the selection phase, the protocol identifies the interval via $m_{\text{seg}} - 1$ parallel secure comparisons. In the compute phase, $n_{\text{seg}} + 1$ Π_{MUX} are computed to get ciphertext coefficients. The protocol then computes self-multiplications $\langle x^k\rangle$, where k is the power of the polynomial, and performs secure multiplications with the coefficients and shares. The local additive share additions are free. Thus, the per-element complexity is Comm. $= (m_{\text{seg}} - 1)\, cost_{\text{comp}} + (n_{\text{seg}} + 1)\, cost_{\text{mux}} + m_{\text{seg}}(k-1)^2\, cost_{\text{mul}}$. This makes the power of every entry uniform and equal to the maximum power among all segments. Under a piecewise quadratic fit, the multiplicative depth is thus capped at two, and the number of secure multiplications per element is fixed.

Further Optimization. To further reduce the number of secure comparisons and minimize communication rounds, we unify the computation $1\{x < b_i\}$, the input comparisons against each breakpoint. Then we reverse lower $\langle \text{ind}\rangle_i \oplus \text{True}$ and concatenate upper $\langle \text{ind}\rangle_{i+1}$ comparison results to efficiently obtain the intermediate comparisons for intervals $[b_i, b_{i+1}], i \in [m_{\text{seg}} - 1]$, reducing communication overhead.

On the other hand, in the PPML setting, the server publicly opens the coefficients of the polynomial approximation for the non-linear function. Consequently, during our preprocessing phase, we also open the matrix C. When both parties compute the $(\Pi_{\text{MUX}}(\langle \text{ind}^\top\rangle^b, c_i))$, the computation

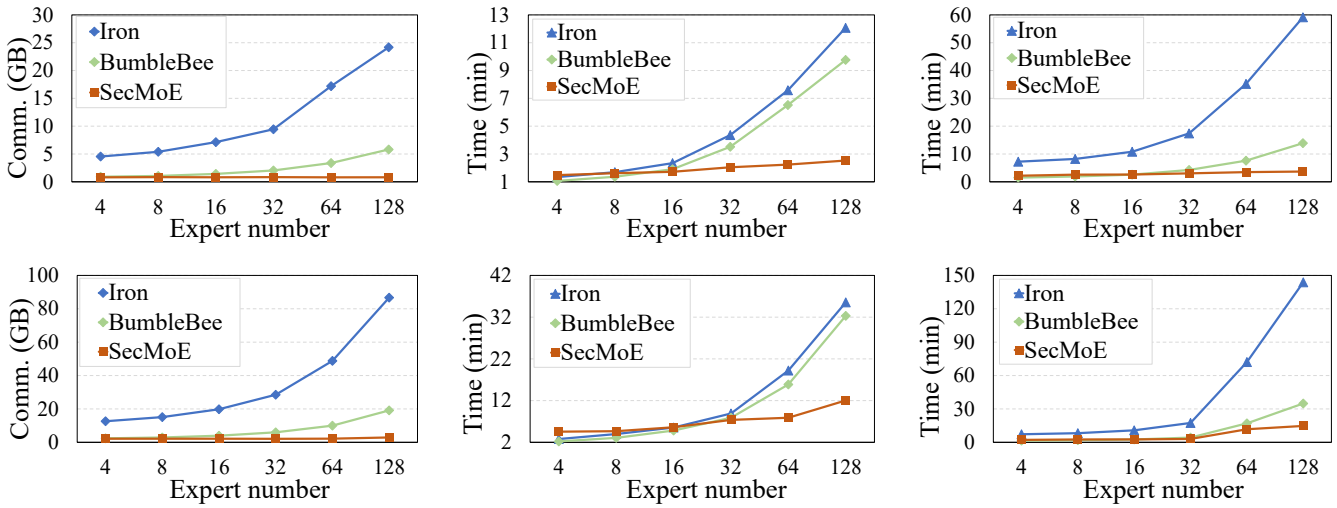

Figure 3: The private MoE-Small (top) and Switch-Base (bottom) inference for Iron, BumbleBee, and SecMoE. Begin with communication (MB) and time (s) under LAN (middle) and WAN (right) network settings.

for the zero entries of C can be directly set to 0, which reduces the number of Π_{MUX} to save part of communication.

Other Nonlinear Protocols.

For the Softmax function, given the input x, the output of the Softmax function is computed as $\frac{\exp(x_i - max(x))}{\sum \exp(x_j - max(x))}$. Like the previous works (Dong et al. 2023; Lu et al. 2023; Li et al. 2024; Kei and Chow 2025), we focus on piecewise polynomial approximation and adopt BumbleBee (Lu et al. 2023) exponential function. Using Taylor series iteration, the power of the polynomial reaches 2^6, which is not suitable for Select-Then-Compute. And therefore, we keep the original approximation and show the details in the Appendix.

The LayerNorm is denoted as $\text{LayerNorm}(x) = \gamma \cdot (x - (1/d \cdot \sum_j x_j)) \cdot (\sum_j (x[j] - \mu)^2)^{-1/2} + \beta$, where γ and β are hyperparameters and j is the index of input array x. We also follow the LayerNorm employment in BumbleBee.

Experiments

Experiment Setting. Our experimental works on the ring $\mathbb{Z}_{2^{64}}$ with a fixed-point precision of $s = 18$. Benchmarks are simulated on two nodes, with each equipped with 64 vCPUs and 128 GB RAM. Network conditions are emulated under the LAN (1 Gbps bandwidth, 0.5 ms latency) and the WAN network (400 Mbps bandwidth, 4 ms latency). The input consists of eight tokens in one batch.

Metrics. We assess the efficiency of SecMoE by measuring the running time and communication cost. We focus on evaluating the end-to-end performance and do not distinguish between offline and online costs. The online cost is significantly lower compared with the offline cost. Therefore, it is unfair to only compare the online performance. We choose Accuracy (ACC) and the Matthews Correlation Coefficient (MCC) metric, which will be detailed when the averaged results are presented in Table 3.

Model	Parameters	d_{model}	d_{ff}	Num. Heads
BERT-base	110M	768	3072	12
GPT-2	117M	768	3072	12
T5-small	60M	512	2048	8
T5-Base	0.2B	768	2048	12
MoE-Small	124M(8e)	512	2048	8
Switch-Base	0.62B(8e)	768	2048	12
Switch-Base	7B(128e)	768	2048	12

Table 1: Model configurations. d_{model}, d_{ff}, Num. Heads are the dimensions of the hidden layer and model, and the number of transformer attention heads.

Model and Datasets. Table 1 summarizes the model configurations used in our study. To obtain a controlled MoE setting, we construct MoE-small by starting from T5-small and replacing each dense FFN with sparse MoE blocks equipped with N_{exp} experts, keeping the backbone d_{model} and depth unchanged. We evaluate our method on the two MoE models, MoE-small and Switch-Base, $N_{\text{exp}} \in \{8, 16, 32, 128\}$. Compared with the dense baselines, the MoE variants achieve substantially larger parameter capacity (up to 63×), incurring approximately 15.2× more costs.

Accuracy Comparison. We evaluate accuracy on several validation datasets, including three GLUE (Wang et al. 2018) tasks, the sentence acceptability (CoLA), and natural language inference (RTE and QNLI). The evaluation result is shown in Table 3. Our experiment is built on BumbleBee. This baseline has minor errors due to truncation after multiplication and the conversion from the floating to fixed-point value, which can be fine-tuned to less than 0.05%.

Efficiency Comparison. The efficiency comparisons for private inference under LAN and WAN settings are shown

	Runtime (min) in MoE-Small			Runtime (min) in Switch-Base			Communication Comparison (GB)			
	32e	64e	128e	32e	64e	128e	16e	32e	64e	128e
	LAN Network						MoE-Small			
Iron	4.35 (2.1×)	7.58 (3.3×)	12.07 (4.7×)	8.91 (1.2×)	19.2 (2.4×)	35.5 (2.9×)	7.13 (8.9×)	9.44 (11.2×)	17.19 (21.2×)	24.17 (29.4×)
BumbleBee	3.51 (1.7×)	6.51 (2.9×)	9.76 (3.8×)	7.88 (1.0×)	15.8 (2.0×)	32.3 (2.6×)	1.42 (1.8×)	2.04 (2.4×)	3.37 (4.2×)	5.81 (7.1×)
SecMoE	**2.04**	**2.23**	**2.52**	**7.37**	**7.88**	**12.1**	**0.81**	**0.84**	**0.81**	**0.82**
	WAN Network						Switch-Base			
Iron	17.40 (5.7×)	35.19 (10.1×)	59.14 (16.1×)	17.40 (5.7×)	72.12 (6.1×)	143.78 (9.7×)	19.81 (9.2×)	28.51 (13.6×)	48.80 (22.4×)	86.68 (29.8×)
BumbleBee	4.24 (1.3×)	7.59 (2.1×)	13.88 (3.8×)	4.24 (1.3×)	17.05 (1.4×)	34.89 (2.3×)	3.96 (1.8×)	5.99 (2.9×)	9.98 (4.6×)	19.13 (6.6×)
SecMoE	**3.04**	**3.47**	**3.68**	**3.04**	**11.72**	**14.73**	**2.15**	**2.09**	**2.18**	**2.91**

Table 2: Comparing the runtime (min) and communication (GB) costs of Iron, BumbleBee, and SecMoE under various expert settings and networks.

Dataset	Size	Metric	Plaintext	Our work
CoLA	1043	MCC	41.0	41.0
QNLI	1000	ACC	90.3	90.2
RTE	277	ACC	69.9	70.0

Table 3: Accuracy comparison of plaintext floating-point baseline and SecMoE.

in Figure 3. We use Iron (Hao et al. 2022) and Bumblebee (Lu et al. 2023) as baselines. Some recent works also propose PPML solutions. MPCFormer (Li et al. 2023) and SHAFT (Kei and Chow 2025) are implemented on the Crypten (Knott et al. 2021) framework, whose preprocessing relies on a trusted dealer, and therefore, we do not compare with them. Nimbus (Li et al. 2024) proposes a protocol allowing the server to send encrypted weights to the client at the preprocessing stage. However, in MoE models, the expanded parameters place high demands on the client's RAM capacity (for up to 30 GB). Therefore, we do not compare with Nimbus and leave the MoE private inference supporting on-demand RAM loading as future work.

Figure 3 compares end-to-end latency (lower is better) across the number of experts $N_{\text{exp}} \in \{8, 16, 32, 128\}$ for Iron, BumbleBee, and our SecMoE, where $K_{\text{exp}} = 1$ expert is selected. While the two baselines grow rapidly with N_{exp}, SecMoE remains flat and obtains $5.67\times$ to $11.24\times$ lower latency than Iron and $1.13\times$ to $2.43\times$ over BumbleBee. The advantage widens at larger expert counts, due to SecMoE 's selection phase reducing redundant experts' computing. We have not tested larger experts (e.g., 256 or more) because of the out-of-memory problem caused by loading model parameters and storing pseudorandom correlations like Beaver Triples(Beaver 1996).

Specifically, in the left part of Table 2, SecMoE maintains communication efficiency across both network settings and architectures, with the advantage widening at larger expert counts. In the LAN setting, SecMoE achieves $4.7\times$ and $3.9\times$ speedups over Iron and BumbleBee on the MoE-small model. Under the WAN setting, SecMoE further extends the advantage due to the low communication in the MoE layers. On MoE-small with 128 experts, SecMoE takes 3.68 mins, achieving a $16.1\times$ speedup compared with Iron. Even against the stronger framework BumbleBee, SecMoE is $3.8\times$ faster. On the larger Switch-Base model, SecMoE maintains strong gains, achieving up to $9.7\times$ faster than Iron and $2.3\times$ faster than BumbleBee.

For the secure GeLU, we test SecMoE against BumbleBee for the communication and runtime under LAN and WAN. In the MoE-Small model with 128 experts, GeLU in SecMoE reduces 44% of communication and 11% of LAN runtime compared to BumbleBee. In the Switch-Base model with 128 experts, SecMoE's GeLU advantage extends to both runtime and bandwidth, achieving a $7.1\times$ speedup over BumbleBee and reducing communication by 81%.

Communication Analysis. As for the communication shown in the right part of Table 2, SecMoE's performance remains stable despite the varying number of experts, reducing communication by up to $29.4\times$ compared to Iron and $6.6\times$ compared to BumbleBee. For instance, on the MoE-small model in LAN, SecMoE's computation grows only 24% from 32e (2.04 mins) to 128e (2.52 mins), whereas Iron and BumbleBee increase by 178% over the same range, showing SecMoE 's robustness to model scaling.

Conclusion

To address the inefficiency in MPC-based MoE, we propose SecMoE, a 2-PC privacy-preserving inference framework. SecMoE employs two optimizations: secure sparse MoE and secure polynomial selection via Select-Then-Compute, which respectively avoid redundant expert computation and reduce communication in piecewise evaluation. Extensive experiments show that, SecMoE lowers communication by $1.8\sim29.8\times$ and speeds up end-to-end inference by up to $16.1\times$. To our knowledge, SecMoE is the first practical 2-PC protocol for secure MoE inference.

Acknowledgements

The work was supported by the National Natural Science Foundation of China (Grant No. 72442029), National Key R&D Program of China (2022YFB3102100), Shenzhen Stable Supporting Program (General Project) (GXWD20231130110352002), Humanities and Social Science Fund of Ministry of Education of China (No. 24JZD026), Guangdong Provincial Key Laboratory of Novel Security Intelligence Technologies (2022B1212010005).

References

Beaver, D. 1996. Correlated pseudorandomness and the complexity of private computations. In *The ACM symposium on Theory of computing*, 479–488.

Cohen, V.; and Gokaslan, A. 2020. OpenGPT-2: Open language models and implications of generated text. *XRDS: Crossroads, The ACM Magazine for Students*, 27(1): 26–30.

Collobert, R.; Bengio, S.; and Bengio, Y. 2001. A parallel mixture of SVMs for very large scale problems. *Advances in Neural Information Processing Systems*, 14.

Dong, Y.; Lu, W.-j.; Zheng, Y.; Wu, H.; Zhao, D.; Tan, J.; Huang, Z.; Hong, C.; Wei, T.; and Cheng, W. 2023. Puma: Secure inference of llama-7b in five minutes. *arXiv preprint arXiv:2307.12533*.

Fedus, W.; Zoph, B.; and Shazeer, N. 2022. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120): 1–39.

Hao, M.; Li, H.; Chen, H.; Xing, P.; Xu, G.; and Zhang, T. 2022. Iron: Private inference on transformers. *Advances in Neural Information Processing Systems*, 35: 15718–15731.

Huang, Z.; Lu, W.-j.; Hong, C.; and Ding, J. 2022. Cheetah: Lean and fast secure {two-party} deep neural network inference. In *31st USENIX Security Symposium*, 809–826.

Jacobs, R. A.; Jordan, M. I.; Nowlan, S. J.; and Hinton, G. E. 1991a. Adaptive Mixtures of Local Experts. *Neural computation*, 3(1): 79–87.

Jacobs, R. A.; Jordan, M. I.; Nowlan, S. J.; and Hinton, G. E. 1991b. Adaptive mixtures of local experts. *Neural computation*, 3(1): 79–87.

Jordan, M. I.; and Jacobs, R. A. 1994. Hierarchical mixtures of experts and the EM algorithm. *Neural computation*, 6(2): 181–214.

Juvekar, C.; Vaikuntanathan, V.; and Chandrakasan, A. 2018. GAZELLE: A low latency framework for secure neural network inference. In *27th USENIX Security Symposium*, 1651–1669.

Kei, A. Y.; and Chow, S. S. 2025. SHAFT: Secure, Handy, Accurate, and Fast Transformer Inference. In *Network and Distributed System Security Symposium*, volume 2025.

Kenton, J. D. M.-W. C.; and Toutanova, L. K. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Annual Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics*, 4171–4186.

Knott, B.; Venkataraman, S.; Hannun, A.; Sengupta, S.; Ibrahim, M.; and van der Maaten, L. 2021. Crypten: Secure multi-party computation meets machine learning. *Advances in Neural Information Processing Systems*, 34: 4961–4973.

Li, D.; Shao, R.; Wang, H.; Guo, H.; Xing, E. P.; and Zhang, H. 2023. MPCFormer: fast, performant and private Transformer inference with MPC. In *11th International Conference on Learning Representations*, 1–16.

Li, Z.; Yang, K.; Tan, J.; Lu, W.-j.; Wu, H.; Wang, X.; Yu, Y.; Zhao, D.; Zheng, Y.; Guo, M.; et al. 2024. Nimbus: Secure and efficient two-party inference for transformers. *Advances in Neural Information Processing Systems*, 37: 21572–21600.

Liang, Z.; Wang, P.; Zhang, R.; Xu, N.; Zhang, S.; Xing, L.; Bai, H.; and Zhou, Z. 2024. Merge: Fast private text generation. In *AAAI Conference on Artificial Intelligence*, volume 38, 19884–19892.

Lu, W.; Huang, Z.; Gu, Z.; Li, J.; Liu, J.; Ren, K.; Hong, C.; Wei, T.; and Chen, W. 2023. BumbleBee: Secure Two-party Inference Framework for Large Transformers. *32nd Annual Network and Distributed System Security Symposium*.

Mohassel, P.; and Rindal, P. 2018. ABY3: A mixed protocol framework for machine learning. In *ACM SIGSAC conference on computer and communications security*, 35–52.

Pang, Q.; Zhu, J.; Möllering, H.; Zheng, W.; and Schneider, T. 2024. BOLT: Privacy-Preserving, Accurate and Efficient Inference for Transformers. In *IEEE Symposium on Security and Privacy*, 130–130.

Rathee, D.; Rathee, M.; Kumar, N.; Chandran, N.; Gupta, D.; Rastogi, A.; and Sharma, R. 2020. Cryptflow2: Practical 2-party secure inference. In *ACM SIGSAC Conference on Computer and Communications Security*, 325–342.

Shazeer, N.; Mirhoseini, A.; Maziarz, K.; Davis, A.; Le, Q.; Hinton, G.; and Dean, J. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*.

Srinivasan, W. Z.; Akshayaram, P.; and Ada, P. R. 2019. DELPHI: A cryptographic inference service for neural networks. In *29st USENIX Security Symposium*, 2505–2522.

Touvron, H.; Lavril, T.; Izacard, G.; Martinet, X.; Lachaux, M.-A.; Lacroix, T.; Rozière, B.; Goyal, N.; Hambro, E.; Azhar, F.; et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. *Advances in neural information processing systems*, 30: 1–11.

Wagh, S.; Gupta, D.; and Chandran, N. 2019. SecureNN: 3-party secure computation for neural network training. In *Privacy Enhancing Technologies*, 26–49.

Wang, A.; Singh, A.; Michael, J.; Hill, F.; Levy, O.; and Bowman, S. R. 2018. GLUE: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*.

Yang, K.; Weng, C.; Lan, X.; Zhang, J.; and Wang, X. 2020. Ferret: Fast extension for correlated OT with small communication. In *ACM SIGSAC Conference on Computer and Communications Security*, 1607–1626.

Zeng, W.; Li, M.; Xiong, W.; Tong, T.; Lu, W.-j.; Tan, J.; Wang, R.; and Huang, R. 2023. Mpcvit: Searching for accurate and efficient mpc-friendly vision transformer with heterogeneous attention. In *IEEE/CVF International Conference on Computer Vision*, 5052–5063.