ARION: Attention-Optimized Transformer Inference on Encrypted Data

Linhan Yang, Jingwei Chen, Wangchen Dai, Shuai Wang, Wenyuan Wu, Yong Feng

Abstract—**Privacy-preserving Transformer inference (PPTI) is essential for deploying large language models (LLMs) such as BERT and LLaMA in sensitive domains. In these models, the attention mechanism is both the main source of expressiveness and the dominant performance bottleneck under fully homomorphic encryption (FHE), due to large ciphertext matrix multiplications and the softmax nonlinearity. This paper presents ARION, a non-interactive FHE-based PPTI protocol that specifically optimizes the computation of encrypted attention. First, for the three consecutive ciphertext matrix multiplications in multi-head attention, we introduce the double Baby-Step Giant-Step algorithm, which significantly reduces the number of ciphertext rotations. On BERT-Base, ARION achieves an 82.5% reduction in rotations over the state-of-the-art PPTI protocol MOAI (2025), corresponding to a 5.7x reduction in rotation cost. Second, we propose a linear–nonlinear fusion technique tailored to the softmax evaluation in attention. By decomposing softmax into shift-by-maximum, exponentiation, and reciprocal sub-steps and fusing them with the surrounding encrypted matrix operations, ARION enables efficient attention evaluation while remaining compatible with diverse ciphertext packing formats. We implement ARION using Lattigo and first evaluate attention kernels on popular LLMs including BERT-Tiny, BERT-Base, and LLaMA, confirming the practicality and scalability of the proposed optimizations for encrypted attention computation. For end-to-end applications, on classification tasks for several benchmark datasets, ARION attains accuracy comparable to plaintext inference and yields up to 2.5x end-to-end speedups over MOAI for BERT-Base.** [1]

I. Introduction

Artificial intelligence technologies, especially large language models (LLMs), are increasingly being adopted in critical domains such as healthcare, finance, and public administration. However, these application scenarios often involve processing large volumes of sensitive data, making data privacy protection a core concern for both academia and industry. To address this challenge, privacy-preserving machine learning (PPML) has emerged and continues to advance both theoretically and practically.

Among various PPML techniques, privacy-preserving Transformer inference (PPTI) has emerged as a key research focus, particularly in inference scenarios. Existing cryptographic solutions can be broadly categorized into two classes: multi-party computation (MPC)-based protocols and fully homomorphic encryption (FHE)-based protocols. While MPC-based approaches offer strong security guarantees, they typically suffer from substantial communication overhead and require numerous rounds of interaction, which makes large-scale deployment challenging. For example, for BERT-Base [1], a state-of-the-art MPC-based method, BLB [2], incurs approximately 16 GB of communication per encrypted inference.

FHE [3], as an encryption scheme that allows arbitrary computations directly on ciphertexts, provides a solid theoretical foundation for end-to-end secure inference without exposing the underlying data. One prominent advantage of FHE-based PPTI is its low communication cost; for instance, NEXUS [4] performs encrypted inference on BERT-Base with only 0.16 GB of communication. Nevertheless, applying FHE to real-world deep learning inference—especially for models like Transformers [5] that involve extensive matrix computations and complex nonlinear operations—remains a major computational efficiency bottleneck. For example, the state-of-the-art FHE-based non-interactive PPTI protocol THOR (CCS'25) [6] requires about 10 minutes on a single GPU to perform a single encrypted BERT-Base inference on a sequence of 128 tokens. Even when the model is retrained with polynomial activation functions, Powerformer [7] (ACL'25) still incurs a per-query latency of about 5.74 minutes on a single GPU. A very recent protocol, MOAI [8] (ePrint 2025/991), reports amortized per-sequence runtimes of approximately 9.6 minutes on CPU. Overall, these works have steadily improved the efficiency of FHE-based non-interactive PPTI protocols, yet the resulting end-to-end latency remains too high for practical deployment.

A. Challenges of FHE in Non-Interactive PPTI

In Transformer models, such as GPT [9], BERT [1], LLaMa [10], etc., there exist extensive and diverse matrix computations. These numerous matrix operations are not only high-dimensional but also highly diverse in type. For instance, in BERT-Base inference, some matrices have dimensions as large as 3072. During plaintext inference, matrix multiplication accounts for over 90% of the total computation time. For encrypted inference, taking a recent PPTI protocol NEXUS [4] as an example, various encrypted matrix computations

Manuscript received December 18, 2025. This work was supported in part by National Key Research and Development Program of China (2025YFA1017201) and National Natural Science Foundation of China (12571553). *(Corresponding author: Jingwei Chen)*

Linhan Yang and Wangchen Dai are with School of Cyber Science and Technology, Sun Yat-Sen University, Shenzhen 518107, China (e-mail:yanglh36@mail2.sysu.edu.cn; daiwch@mail.sysu.edu.cn)

Shuai Wang are with School of Information Science and Engineering, Chongqing Jiaotong University, Chongqing 400074, China (e-mail:shuaiwang@mails.cqjtu.edu.cn)

Jingwei Chen, Wenyuan Wu and Yong Feng are with Chongqing Key Lab of Secure Computing for Biology, Chongqing Institute of Green and Intelligent Technology, CAS, Chongqing 400714, China, and also with Chongqing College, University of CAS, Chongqing 400714, China (e-mail: chenjingwei@cigit.ac.cn; wuwenyuan@cigit.ac.cn; yongfeng@cigit.ac.cn).

[1]Our implementation is available at https://github.com/hangenba/Arion.

constitute approximately half of the total runtime in encrypted inference for BERT-Base. In particular, these computations involve ciphertext-plaintext matrix multiplication (CPMM), ciphertext-ciphertext matrix multiplication (CCMM), ciphertext matrix transposition (CMT), and other operations, each possibly requiring different ciphertext packing formats. Ensuring compatibility and correctness among these differing packing formats, while achieving efficient encrypted matrix operations, constitutes the first major challenge in the design of FHE-based PPTI protocols. Most existing works have focused primarily on addressing this challenge.

Moreover, the nonlinear operations within Transformers present significant challenges for encrypted inference, particularly the softmax computation in the attention module. The softmax function involves multiple nonlinear sub-operations, including maximum value extraction (max), exponentiation (exp), and reciprocal ($1/x$). Mainstream FHE-based implementations for these functions typically rely on polynomial approximations. Although increasing the polynomial degree can reduce approximation errors, it simultaneously increases ciphertext noise accumulation, consequently increasing the need for bootstrapping and ultimately degrading performance. Furthermore, the inputs to these nonlinear functions typically originate directly from upstream encrypted matrix operations, featuring diverse packing formats that are challenging to integrate seamlessly. Thus, achieving compatibility of ciphertext packing methods across the intricate interplay of linear and nonlinear computational steps while maintaining overall computational efficiency remains a critical technical challenge for applying FHE to PPTI.

B. Contributions

We propose Attention-optimized tRansformer Inference On eNcrypted data (ARION), an FHE-based non-interactive PPTI protocol, which significantly enhances the efficiency of attention computation on encrypted data. Our contributions include:

- We propose a general optimized algorithm for three consecutive ciphertext matrix multiplications, named *double Baby-Step Giant-Step* (*double-BSGS*), which substantially reduces the number of ciphertext rotations required in the attention mechanism, thereby improving the efficiency of linear computations under encryption. For BERT-Base, our encrypted attention computation achieves an approximately 82.5% reduction (or 5.7x reduction) in ciphertext rotations compared with the state-of-the-art FHE-based PPTI protocol MOAI [8] (Table III).Furthermore, by applying double-BSGS to DASHformer [23], a compact BERT-based model with a single Transformer layer and 139K parameters developed for the iDASH 2024 competition[2], our solution won one of the two first places in this competition (Table IX).
- We propose a *linear–nonlinear fusion* technique that decomposes the softmax computation into three sub-steps: shift by max, exponentiation (exp), and reciprocal ($1/x$) so that each sub-step can be seamlessly integrated with the corresponding matrix operations at different stages, ensuring full compatibility with the matrix packing method. This optimization, together with double-BSGS, constitutes a core innovative component of ARION.
- We implement ARION with Lattigo [22] on CPU, and perform comprehensive end-to-end evaluation on two LLMs, BERT-Tiny and BERT-Base. Experimental results demonstrate that the classification accuracy loss is comparable with the plaintext computation, and the amortized inference cost per sequence is 9.3 seconds and 225 seconds for BERT-Tiny and BERT-Base, respectively. This represents a 34.6x and 2.5x speedup compared with state-of-the-art CPU-based algorithms Tricycle [15] and MOAI [8], respectively (Table VII).

The proposed optimizations in ARION are applicable to any Transformer model employing standard scaled dot-product attention, such as BERT and LLaMA (Table VI).

C. Limitations

Although our current implementation focuses on CPUs, whereas state-of-the-art PPTI systems such as THOR [20], PowerFormer [7], and MOAI [8] already provide GPU-based implementations, we emphasize that the main contribution of this work lies in algorithmic improvements that yield substantial speedups in encrypted computation. These improvements are orthogonal to hardware-specific optimizations (e.g., GPUs), and hence ARION can naturally be further accelerated by combining our methods with hardware accelerators.

II. Related Work

Privacy-preserving machine learning (PPML) encompasses a wide range of research areas, including multi-party computation (MPC), fully homomorphic encryption (FHE), differential privacy, trusted execution environments, and federated learning. Among these, MPC and FHE have received particular attention due to their cryptographic foundations and provable security guarantees; interested readers are referred to [24], [25] and references therein for further reading. Broadly speaking, existing research on privacy-preserving Transformer inference (PPTI) can be classified into MPC-based [26]–[34], hybrid MPC-HE [2], [35]–[39], and FHE-based approaches [4], [8], [11], [13], [14], [20], [40], [41]. MPC-based protocols primarily rely on cryptographic primitives such as secret sharing and oblivious transfer. Hybrid MPC-HE methods typically utilize MPC to handle nonlinear components of the Transformer (e.g., softmax, GELU, LayerNorm, Tanh), while employing HE to compute the linear components (e.g., matrix multiplications). For a comprehensive discussion of related work, we refer readers to the recent survey [42]. Here, we mainly focus on FHE-based PPTI protocols.

A. FHE-Based PPTI Protocols

Table I summarizes recent FHE-based non-interactive PPTI protocols. Note that THE-X [40] is also only based on HE, but it requires interactions to complete the computation of nonlinear functions like softmax and GELU.

Both PolyTransformer [11] and PowerSoftmax [13] follow the approach of replacing activation functions with polynomial

[2]https://www.humangenomeprivacy.org/2024/

TABLE I: Comparison of FHE-based non-interactive PPTI protocols (time in seconds).

Protocols	Batch	Retraining	Flexibility	Models	Platform	HE Library	Tokens	(Amortized) Runtime
PolyTransformer [11]	✗	✓	✗	BERT-like†	CPU (64) +GPU	HeLayers [12]	128	211
PowerSoftmax [13] *	✗	✓	✗	RoBERTa-like ‡	CPU+GPU	HeLayers [12]	128	≈ 400
FHE-BERT-Tiny [14]	✗	✗	✗	BERT-Tiny	CPU (8)	OpenFHE	40	≈ 600
Tricycle [15] *	✗	✗	✗	BERT-Tiny	CPU (128)	OpenFHE [16]	44	322
NEXUS [4]	✓	✗	✗	BERT-Base	CPU (32)	SEAL [17], [18]	128	857⋆
					GPU	Phantom [19]	128	37⋆
THOR [20]	✗	✗	✗	BERT-Base	GPU	Liberate.FHE [21]	128	626
Powerformer [7]	✗	✓	✗	BERT-Base	GPU	Liberate.FHE [21]	128	344
MOAI [8] *	✓	✗	✓	BERT-Base	CPU (112)	SEAL [17], [18]	128	574
					GPU	Phantom [19]	128	141
ARION (This work)	✓	✗	✓	BERT-Tiny	CPU (64)	Lattigo [22]	128	9.3
				BERT-Base			128	225

Batch refers to whether the protocol supports processing inference for multiple sequences simultaneously in a single execution. Retraining indicates whether retraining of the model is required. Flexibility indicates whether the protocol supports sequence with arbitrary length tokens. CPU (n) indicates that the number of used threads is n. $*$ Those are preprints by the submission date of this paper (December 2025). † This is a retrained BERT-based model with 6 Transformer layers and 53.3 million parameters. ‡ This is a retrained RoBERTa-based model with 32 Transformer layers and 1.4 billion parameters. ⋆ The runtime reported by NEXUS seems not end-to-end runtime. NEXUS also provides the runtime results on LLaMA 3-8B, which are not included here.

functions, retraining the model, and then performing encrypted inference. On the one hand, such polynomial substitution may result in accuracy degradation; on the other hand, retraining LLMs is extremely costly. In addition, these two works primarily focus on adapting LLMs to be HE-friendly and are implemented based on HeLayers [12]. The state-of-the-art of this line is Powerformer [7], which performs a single encrypted BERT-Base inference on a single GPU in about 5.74 minutes.

From the perspective of model complexity, both FHE-BERT-Tiny [14] and Tricycle [15] are PPTI protocols specifically designed for BERT-Tiny. The former employs precomputed values for the mean and variance in LayerNorm, while the latter extends bicyclic encoding [43] to tricycle encoding to accelerate matrix computations. NEXUS [4], THOR [20], and MOAI [8] all support encrypted inference for BERT-Base.

In terms of design goals, existing FHE-based PPTI protocols can be roughly categorized according to whether they support batch inference, with both categories leveraging the SIMD capabilities of CKKS. Protocols that do not support batch inference [7], [14], [15], [20] often pack data from Transformer's multi-heads into a single ciphertext for SIMD parallelism, making GPU implementation feasible; however, their runtime typically increases linearly with the number of sequence tokens, as reported in [14]. Protocols that support batch inference can process multiple sequences in a single execution [4], [8], [41]. It is worth noting that, although batch inference typically incurs a longer wall-clock time per execution, it allows the use of larger batch sizes to amortize the cost of expensive homomorphic operations and thus reduce the average per-sequence latency. For BERT-Base, both the state-of-the-art non-batch protocol THOR (GPU) [20] and the state-of-the-art batch protocol MOAI (CPU) [8] require approximately 10 minutes to complete inference on a single sequence.

In summary, as shown in Table I, although FHE-based PPTI protocols are intrinsically non-interactive, their current efficiency remains insufficient for real-world deployment. ARION presented in this work significantly narrows this gap, achieving 34.6x and 2.5x speedups on CPU for encrypted inference on BERT-Tiny and BERT-Base, respectively.

B. *What's New of ARION*

From the perspective of attention computation, the protocol most closely related to ARION is MOAI [8]. Both protocols adopt similar data packing strategies to facilitate batch processing. We note that this strategy was first introduced to PPTI by part of the authors of this paper when preparing solutions for the iDASH 2024 competition [41]. In addition, ARION differentiates itself from MOAI by proposing the double-BSGS algorithm. This method specifically optimizes the triple matrix multiplication $(\boldsymbol{QK}^\top)\boldsymbol{V}$ within the attention module. By integrating this algorithm with our linear-nonlinear fusion technique, ARION overcomes the limitations of previous methods that often sacrifice accuracy for packing compatibility. This design allows for the precise execution of softmax sub-steps (maximum extraction, exponentiation, and reciprocal) within the optimized matrix flow. Consequently, ARION maintains high inference accuracy while significantly improving efficiency. In particular, for a token length of 128, ARION achieves an 82.5% reduction in ciphertext rotations compared to MOAI.

Additionally, it is worth noting that ARION allows users to specify the maximum token length for sequences to be inferred. For example, with the CKKS polynomial ring dimension fixed at $N = 2^{16}$, a single execution of ARION can perform batched encrypted inference for up to 2^{15-l} sequences, each with a maximum token length of 2^l. For BERT, the maximum token length is 512, i.e., $l \leq 9$. This flexibility enables ARION to efficiently accommodate a wide range of sequence lengths and batch sizes under practical parameter settings.

Finally, to our best knowledge, our open-sourced implementation of ARION is the first end-to-end and non-interactive PPTI protocol implemented with one of the most popular FHE libraries Lattigo [22].

III. Preliminaries

All vectors are in row and denoted in bold font. Let $\mathbf{1}$ be a vector in which all entries are equal to 1, with its dimension being easily determined from the context. For a positive integer z, $[z]$ is the set of $\{0, 1, \ldots, z-1\}$.

A. Fully Homomorphic Encryption: the CKKS Scheme

We employ the CKKS scheme [44] for approximate homomorphic arithmetic. CKKS supports SIMD (Single-Instruction-Multiple-Data) operations, allowing parallel processing of complex vectors packed into a single ciphertext. The scheme provides standard homomorphic operations including addition (Add), multiplication (Mul), rotation (Rot), and bootstrapping (Bts). CKKS is semantic secure under the RLWE assumption [45]. For a detailed description of these primitives and parameters, please refer to Appendix A.

B. The BERT Model

In this paper, we focus on the inference of Transformer-based large language models, specifically BERT [1]. A standard Transformer architecture consists of L stacked layers, where each layer comprises an Attention module and a Feed-Forward Network (FFN). Here we define the following notations used throughout the paper:

- m: the sequence length (number of tokens).
- d: the hidden size (embedding dimension).
- H: the number of attention heads.
- d_h: the dimension of each head, typically $d_h = d/H$.
- d_f: the dimension of the fully connected layer.
- n: the batch size.

The input to the model is a matrix $\boldsymbol{X} \in \mathbb{R}^{m\times d}$ (after embedding). The core computations involve multi-head attention mechanisms, layer normalization (LayerNorm), and non-linear activation functions (e.g., GELU). The detailed mathematical formulation of the BERT workflow is provided in Appendix B. The specific hyperparameters for the BERT variants evaluated in this work (BERT-Tiny and BERT-Base) are listed in Table II.

TABLE II: Hyperparameters for BERT variants.

Model	L	H	d	d_h	d_f	#Params
BERT-Tiny	2	2	128	64	512	4.4M
BERT-Base	12	12	768	64	3072	110M

C. Matrix Multiplication in Secure Transformer Inference

Homomorphic inference over BERT involves a large number of ciphertext-plaintext matrix multiplications (CPMM) and ciphertext-ciphertext matrix multiplications (CCMM), which are the most intricate components. During the iDASH 2024 competition [41], part of the authors of this paper proposed a packing method that homomorphic inference on DASHformer [23], a BERT-based Transformer model for protein classification. With this packing method, PPTI can be realized using a combination of column packing and diagonal packing, without requiring conversions between different packing formats. This approach was further validated in MOAI [8], and it achieves the minimal number of ciphertext rotations, compared with BOLT [36], NEXUS [4], PowerFormer [7], and THOR [20]. ARION follows this strategy, so we first provide a brief overview of the underlying approach.

1) Column Packing and Diagonal Packing: Given a matrix $\boldsymbol{X} \in \mathbb{R}^{m\times d}$, the *column packing* of $\boldsymbol{X}$ is defined to be a set of d ciphertexts, denoted as $\mathsf{ct}.\boldsymbol{X}_{\text{col}} = \{\mathsf{ct}.\boldsymbol{x}_0^\top, \ldots, \mathsf{ct}.\boldsymbol{x}_{d-1}^\top\}$, where $\boldsymbol{x}_i^\top$ is the i-th column of $\boldsymbol{X}$. For a square matrix $\boldsymbol{X} \in \mathbb{R}^{m\times m}$, the *diagonal packing* of $\boldsymbol{X}$ is defined to be a set of m ciphertexts, denoted as $\mathsf{ct}.\boldsymbol{X}_{\text{diag}} = \{\mathsf{ct}.\boldsymbol{d}_0(\boldsymbol{X}), \ldots, \mathsf{ct}.\boldsymbol{d}_{m-1}(\boldsymbol{X})\}$, where $\boldsymbol{d}_i(\boldsymbol{X})$ is the i-th (upper) diagonal of $\boldsymbol{X}$.

2) C2C CPMM: We first consider ciphertext-plaintext matrix multiplication, where the input and output ciphertext matrices are all in column packing. We simply denote it by C2C CPMM. It will be used to compute $\boldsymbol{Q}$, $\boldsymbol{K}$, and $\boldsymbol{V}$ in the attention layer, as well as the linear transformations in the FFN layer. Given column packing $\mathsf{ct}.\boldsymbol{X}_{\text{col}}$ of a matrix $\boldsymbol{X} \in \mathbb{R}^{m\times d}$ and $\boldsymbol{W} = (w_{i,j}) \in \mathbb{R}^{d\times d_h}$, we have the column packing of $\boldsymbol{XW}$ is $\mathsf{ct}.(\boldsymbol{XW})_{\text{col}} = \{\mathsf{ct}.(\boldsymbol{XW})_j : 0 \le j < d_h\}$ with

$$\mathsf{ct}.(\boldsymbol{XW})_j \quad = \quad \mathsf{Add}_{i\in[d]}(\mathsf{CMul}(w_{i,j}, \mathsf{ct}.\boldsymbol{x}_i^\top)).$$

This process requires no ciphertext rotations, and applies to other CPMMs in ARION.

3) CC2D CCMM: CC2D CCMM is for ciphertext-ciphertext matrix multiplication, where the two input matrices are in column packing, while the resulting matrix is in diagonal packing. It happens when computing $\boldsymbol{QK}^\top$, where $\boldsymbol{Q}$ and $\boldsymbol{K}$ are all in column packing. Let $\boldsymbol{Q} = (\boldsymbol{q}_0^\top, \ldots, \boldsymbol{q}_{d_h-1}^\top) \in \mathbb{R}^{m\times d_h}$ and $\boldsymbol{K} = (\boldsymbol{k}_0^\top, \ldots, \boldsymbol{k}_{d_h-1}^\top) \in \mathbb{R}^{m\times d_h}$, i.e., $\boldsymbol{q}_i^\top$ and $\boldsymbol{k}_i^\top$ be the i-th column of $\boldsymbol{Q}$ and $\boldsymbol{K}$, respectively. Then for $i \in [m]$, $\mathsf{ct}.\boldsymbol{d}_i(\boldsymbol{QK}^\top) = \mathsf{Add}_{s\in[d_h]}(\mathsf{Mul}(\mathsf{ct}.\boldsymbol{q}_s^\top, \mathsf{Rot}_i(\mathsf{ct}.\boldsymbol{k}_s^\top)))$, which form the diagonal packing for $\boldsymbol{QK}^\top$.

4) DC2C CCMM: DC2C CCMM is for ciphertext-ciphertext matrix multiplication, where the two input matrices are in diagonal packing and column packing, respectively, while the resulting matrix is in column packing. This occurs at Eq. (5), where the result of the $\mathsf{softmax}(\cdot)$ operation yields a matrix $\boldsymbol{C} \in \mathbb{R}^{m\times m}$ that is multiplied with $\boldsymbol{V} \in \mathbb{R}^{m\times d_h}$. At this point, matrix $\boldsymbol{C}$ is in diagonal packing, while matrix $\boldsymbol{V}$ is in column packing. Now a ciphertext of the j-th column of $\boldsymbol{CV}$ can be computed through

$$\mathsf{ct}.(\boldsymbol{CV})_j = \mathsf{Add}_{i\in[m]}(\mathsf{Mul}(\mathsf{ct}.\boldsymbol{d}_i(\boldsymbol{C}), \mathsf{Rot}_i(\mathsf{ct}.\boldsymbol{v}_j^\top))), \qquad (1)$$

where $\boldsymbol{v}_j^\top$ is the j-th column of $\boldsymbol{V}$ and $j \in [d_h]$.

IV. System Overview

A. System Setup and Threat Model

As shown in Fig. 1, ARION involves two parties: the data owner (client) and the model owner (server). The client holds n sequences to be classified. The server holds parameters of a transformer model to supply inference service. Each sequence is processed via tokenization and positional embedding, resulting in a matrix $\boldsymbol{X} \in \mathbb{R}^{m\times d}$. Given the resulting tensor of dimension $n \times m \times d$ as input, ARION performs the following steps sequentially:

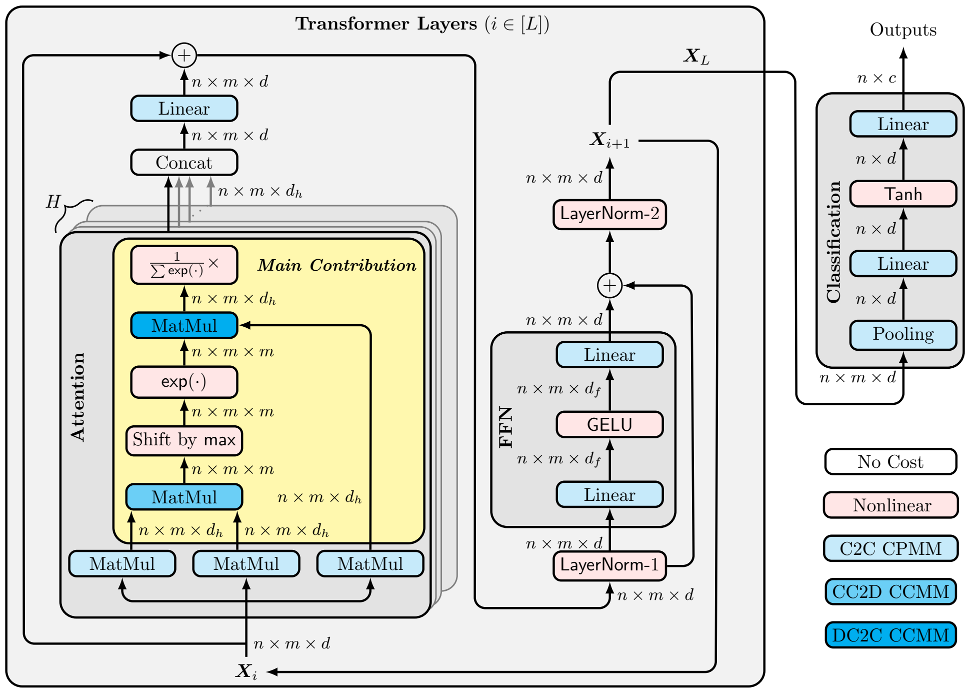


Fig. 1: The system architecture of ARION. The hyperparameters are given in Table II. C2C CPMM is for ciphertext-plaintext matrix multiplication, where the input and output ciphertext matrices are all in column packing. CC2D CCMM is for ciphertext-ciphertext matrix multiplication, where the two input matrices are in column packing, while the resulting matrix is in diagonal packing. DC2C CCMM is for ciphertext-ciphertext matrix multiplication, where the two input matrices are in diagonal packing and column packing, respectively, while the resulting matrix is in column packing. For both column-packed and diagonally-packed matrices, the packing format is preserved throughout all nonlinear modules, which simplifies ciphertext management and enables efficient downstream computation.

- The client generates a pair of keys $(\mathsf{sk}, \mathsf{pk})$ for CKKS, encrypts the input data using the public key pk, and sends the ciphertexts to the server.
- The server performs encrypted inference of the transformer model over the ciphertexts and returns the encrypted classification results to the client. Note that the model parameters are all in plaintext form.
- The client decrypts to obtain the results of classification.

Since FHE supports arbitrary computation on ciphertexts without decryption, Step IV-A) requires no interaction between the client and the server. Thus, ARION works in a fully non-interactive manner.

We assume that the client and the server are mutually untrusted but follow the *semi-honest* (*honest-but-curious*) adversarial model. That is, both parties faithfully execute the prescribed steps of the protocol but may attempt to infer additional information about the other party from the outputs or any intermediate results. On the one hand, a certain degree of trust is necessary, as the server is expected to provide inference services. On the other hand, achieving secure and efficient performance under the semi-honest model is significantly more practical than under the malicious model. Therefore, assuming a semi-honest security model remains a realistic and reasonable approach, as commonly adopted in prior works [4], [7], [8], [20], [36], [41]. Although these protocols do not provide security against malicious adversaries, data privacy can still be preserved.

B. System Architecture

Fig. 1 illustrates the system architecture of ARION. For a batch input $\boldsymbol{X} = \boldsymbol{X}_0 \in \mathbb{R}^{n\times m\times d}$, each $n \times m$ slice of $\boldsymbol{X}$ is packed into a single ciphertext, where n denotes the number of sequences in the batch. Specifically, the j-th ciphertext stores the j-th $n \times m$ slice of $\boldsymbol{X}$, with the first n slots holding the first column of this slice, and so on. This packing strategy was used for parallel matrix multiplication in [46, § 4.2], referred to as interleaved batching in MOAI.

After encryption, the input data are processed through L Transformer layers, as shown in Fig. 1, followed by pooling and classification to produce the final encrypted output. The main contributions of this work focus on optimizing the attention module within the Transformer (the yellow part in Fig. 1), which will be discussed in detail in §V. The proposed attention optimization works with any Transformer model that uses the standard attention mechanism and does not depend on specific architectural choices. The computation of other Transformer modules will be presented in §V-G.

V. Optimized Attention on Encrypted Data

The computation in Eq. (5) essentially evaluates the expression $\boldsymbol{A} = \mathsf{softmax}(\boldsymbol{Q}\boldsymbol{K}^\top)\boldsymbol{V}$, [3] where $\boldsymbol{Q}$, $\boldsymbol{K}$, and $\boldsymbol{V} \in \mathbb{R}^{m\times d_h}$

[3]For simplicity, we omit the constant factor $1/\sqrt{d_h}$ inside the $\mathsf{softmax}$. In practice, this constant can be folded into the weight matrix (e.g., $\boldsymbol{W}_Q$), and thus does not affect the correctness or structure of the computation.

are all encrypted with column packing. For ease of downstream computations, the resulting ciphertexts for $\boldsymbol{A}$ are also required to be in column packing.

In almost all existing FHE-based PPTI protocols, the matrix $\boldsymbol{A}$ is typically computed in the following three steps:

1) compute $\boldsymbol{B} = \boldsymbol{Q}\boldsymbol{K}^\top \in \mathbb{R}^{m \times m}$,
2) compute $\boldsymbol{C} = \mathsf{softmax}(\boldsymbol{B}) \in \mathbb{R}^{m \times m}$, and
3) compute $\boldsymbol{A} = \boldsymbol{C}\boldsymbol{V} \in \mathbb{R}^{m \times d_h}$.

To compute $\boldsymbol{A} = \boldsymbol{C}\boldsymbol{V}$, one can proceed column by column. Specifically, the j-th column of $\boldsymbol{A}$, denoted $\boldsymbol{a}_j^\top$, is given by $\boldsymbol{a}_j^\top = \boldsymbol{C}\boldsymbol{v}_j^\top$, where $\boldsymbol{v}_j^\top$ is the j-th column of matrix $\boldsymbol{V}$. Assume that the matrix $\boldsymbol{C}$ remains in diagonal packing. Following the method proposed by Halevi and Shoup [47] (see Eq. (1)), computing $\boldsymbol{a}_i^\top$ requires m ciphertext rotations. However, Halevi and Shoup also introduced a baby-step giant-step (BSGS) algorithm [48] that reduces the number of required ciphertext rotations to $O(\sqrt{m})$ if $\boldsymbol{C}$ is in plaintext form. The idea is based on the following formula (assuming $m = \ell \cdot k$)

$$\boldsymbol{C}\boldsymbol{v}^\top \;=\; \sum_{0 \le i < \ell} \rho_{ik}\left(\sum_{0 \le j < k} \rho_{-ik}\left(\boldsymbol{d}_{ik+j}(\boldsymbol{C})\right) \odot \rho_j(\boldsymbol{v}) \right),$$

where $\rho_j(\boldsymbol{v})$ denotes the plaintext counterpart of the rotation operation $\mathsf{Rot}_j(\cdot)$, i.e., it represents a left rotation of the vector $\boldsymbol{v}$ by j positions. In principle, the effectiveness of BSGS relies on the assumption that one can *pre-compute* $\mathsf{Rot}_{-ik}(\boldsymbol{d}_{ik+j}(\boldsymbol{C}))$ for all $i \in [\ell]$ and $j \in [k]$. In fact, if ciphertexts of $\mathsf{Rot}_{-ik}(\boldsymbol{d}_{ik+j}(\boldsymbol{C}))$ can be precomputed, they can be reused across all $\boldsymbol{C}\boldsymbol{v}_s$ computations for $s \in [d_h]$, potentially leading to a further reduction in the total number of required ciphertext rotations in attention computation. However, the resulting matrix $\boldsymbol{B}$ is encrypted in diagonal packing (see § III-C3), while the softmax function is applied row by row (see § B-A), which gives rise to the following challenges:

- Assuming that the softmax computation does not alter the packing format of $\boldsymbol{B}$, how to pre-compute all the required ciphertexts of $\mathsf{Rot}_{-ik}(\boldsymbol{d}_{ik+j}(\boldsymbol{B}))$ directly during the evaluation of $\boldsymbol{B} = \boldsymbol{Q}\boldsymbol{K}^\top$?
- How to design a high-precision (shifted by maximum of rows) and matrix-level implementation of softmax that preserves the rotated diagonal packing format of $\boldsymbol{B}$?

To address these challenges, we propose the double-BSGS algorithm. This method generalizes the standard BSGS algorithm for encrypted linear transformation [47] to optimize the triple matrix multiplication structure inherent in the attention module. We decompose the computation of $\boldsymbol{A} = \mathsf{softmax}(\boldsymbol{Q}\boldsymbol{K}^\top)\boldsymbol{V}$ into five steps (as in Fig. 1):

1) Compute rotated ciphertexts of $\boldsymbol{B} = \boldsymbol{Q}\boldsymbol{K}^\top \in \mathbb{R}^{m \times m}$: This step takes $\boldsymbol{Q}$ and $\boldsymbol{K}$ as ciphertexts in column packing and outputs ciphertexts of $\mathsf{Rot}_{-ik}(\boldsymbol{d}_{ik+j}(\boldsymbol{B}))$.
2) Shift the rotated ciphertexts of $\boldsymbol{B}$ by maximum: This step updates $b_{i,j} = b_{i,j} - \mathsf{max}(\boldsymbol{b}_i)$ and ciphertexts of $\mathsf{Rot}_{-ik}(\boldsymbol{d}_{ik+j}(\boldsymbol{B}))$, where $\boldsymbol{b}_i$ is the i-th row of $\boldsymbol{B}$.
3) Compute $\boldsymbol{C} = \mathsf{exp}(\boldsymbol{B})$: This step applies the exponential function component-wise to each element of $\boldsymbol{B}$ without altering its packing format, i.e., the outputs are ciphertexts of $\mathsf{Rot}_{-ik}(\boldsymbol{d}_{ik+j}(\boldsymbol{C}))$.
4) Compute $\tilde{\boldsymbol{A}} = \boldsymbol{C}\boldsymbol{V}$: For each column $\boldsymbol{v}_j$ of $\boldsymbol{V}$, we apply the BSGS algorithm to compute $\boldsymbol{C}\boldsymbol{v}_j$, yielding ciphertexts of $\tilde{\boldsymbol{A}}$ in column packing.
5) Normalization: Compute $\boldsymbol{A} = (\boldsymbol{s}^\top) \odot \tilde{\boldsymbol{A}}$, where $\boldsymbol{s} = (s_i) \in \mathbb{R}^m$ is a vector whose i-th component is given by $s_i = 1/\sum_j c_{i,j}$, with $\boldsymbol{C} = (c_{i,j}) = \mathsf{exp}(\boldsymbol{B})$.

After execution of the five steps above, the i-th row of the output matrix $\boldsymbol{A}$ can then be written as

$$\boldsymbol{a}_i = s_i \cdot \boldsymbol{c}_i \cdot \boldsymbol{V} = \frac{1}{\sum_j c_{i,j}} \boldsymbol{c}_i \cdot \boldsymbol{V} = \mathsf{softmax}((\boldsymbol{b})_i) \cdot \boldsymbol{V}, \quad (2)$$

where $\boldsymbol{c}_i$ and $\boldsymbol{b}_i$ are the i-th row of $\boldsymbol{C} = \mathsf{exp}(\boldsymbol{Q}\boldsymbol{K}^\top)$ and $\boldsymbol{B} = \boldsymbol{Q}\boldsymbol{K}^\top$, respectively. Therefore, the output is mathematically equivalent to $\boldsymbol{A} = \mathsf{softmax}(\boldsymbol{Q}\boldsymbol{K}^\top)\boldsymbol{V}$. In the following, we provide a detailed explanation of each step. (To aid understanding the CCMM part of the above procedure, we present an illustrative example in Fig. 2.) From now on, we assume that $m \le k \cdot \ell$ for some positive integers k and ℓ with $k \approx \ell \approx \sqrt{m}$.

A. Multiplying $\boldsymbol{Q}$ and $\boldsymbol{K}^\top$

Given that $\boldsymbol{Q} \in \mathbb{R}^{m \times d_h}$ and $\boldsymbol{K} \in \mathbb{R}^{m \times d_h}$ are both encrypted using column packing, Algorithm 1 is designed to compute the ciphertexts of $\rho_{-ik}(\boldsymbol{d}_{ik+j}(\boldsymbol{B}))$, where $\boldsymbol{B} = \boldsymbol{Q}\boldsymbol{K}^\top$, $i \in [\ell]$, and $j \in [k]$.

Algorithm 1 Rotated ciphertexts of diagonals of $\boldsymbol{B} = \boldsymbol{Q}\boldsymbol{K}^\top$

```
Input: $(\mathsf{ct}.\boldsymbol{q}_s)_{s\in[d_h]}$ and $(\mathsf{ct}.\boldsymbol{k}_s)_{s\in[d_h]}$, where $\boldsymbol{q}_s$ and $\boldsymbol{k}_s$ are the $s$-th
       column of $\boldsymbol{Q} \in \mathbb{R}^{m\times d_h}$ and $\boldsymbol{K} \in \mathbb{R}^{m\times d_h}$, respectively.
Output: $\mathsf{ct}.\rho_{-ik}(\boldsymbol{d}_{ik+j}(\boldsymbol{B}))$ for $i \in [\ell]$ and $j \in [k]$, where $\boldsymbol{B} =$
       $\boldsymbol{Q}\boldsymbol{K}^\top$ and $m \le k \cdot \ell$.
1: for $s \in [d_h]$ do
2:     for $j \in [k]$ do                                   // Only for baby steps
3:         Compute $\mathsf{ct}.\rho_j(\boldsymbol{k}_s) \leftarrow \mathsf{Rot}_j(\mathsf{ct}.\boldsymbol{k}_s)$.
4:     for $i \in [\ell]$ do                                // Only for giant steps
5:         Compute $\mathsf{ct}.\rho_{-ik}(\boldsymbol{q}_s) \leftarrow \mathsf{Rot}_{-ik}(\mathsf{ct}.\boldsymbol{q}_s)$.
6: for $i \in [\ell]$ do
7:     for $j \in [k]$ do
8:         Compute $\mathsf{ct}.\rho_{-ik}(\boldsymbol{d}_{ik+j}(\boldsymbol{B}))$ as
           $\mathsf{Add}_{s\in[d_h]}\,(\mathsf{Mul}\,(\mathsf{ct}.\rho_{-ik}(\boldsymbol{q}_s),\ \mathsf{ct}.\rho_{-j}(\boldsymbol{k}_s)))$.
```

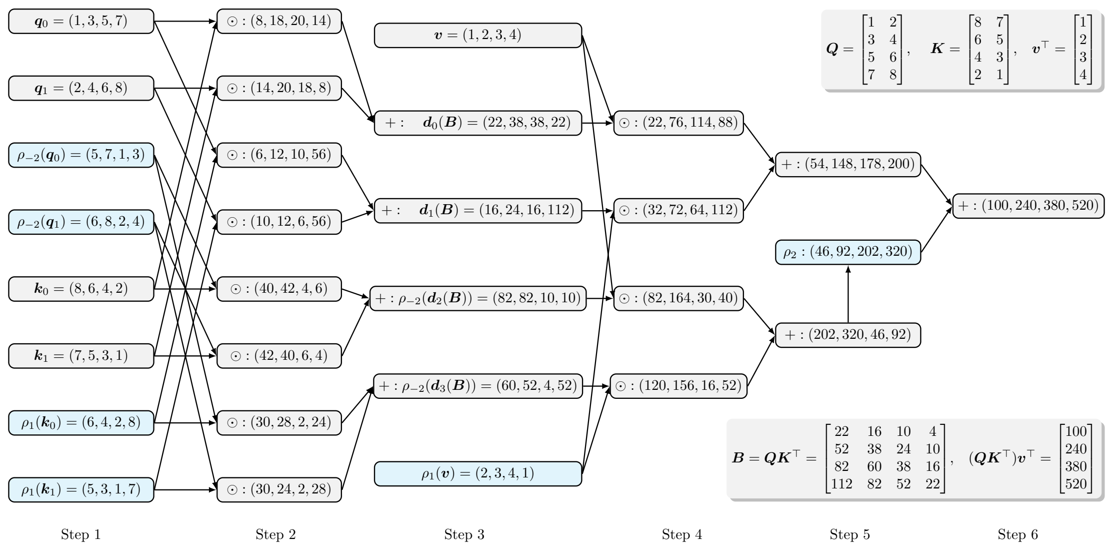
Proposition 1. *Algorithm 1 is correct and requires at most $(k + \ell - 2)d_h \approx 2(\sqrt{m} - 1)d_h$ ciphertext rotations, assuming $k \approx \ell \approx \sqrt{m}$.*

All proofs in this section can be found in Appendix C.

B. Shifting $\boldsymbol{B}$ by Maximum

To prevent overflow during the computation of softmax, it is common to use the shifted formulation $\mathsf{softmax}(\boldsymbol{x}) = \mathsf{softmax}(\boldsymbol{x} - x_{\mathsf{max}})$ with $x_{\mathsf{max}} = \mathsf{max}_{i\in[m]}(x_i)$. However, it is well known that computing the maximum in encrypted form is highly challenging. Therefore, both NEXUS [4] and MOAI [8] approximate x_{max} using a pre-defined constant c. If c is too small, overflow may still occur; if c is too large, the computed softmax values are nearly all close to zero.

In a very recent work [15], Lim et al. introduced a *statistical maximum estimation* technique into the PPTI setting. This



$q_0 = (1,3,5,7)$
$q_1 = (2,4,6,8)$
$\rho_{-2}(q_0) = (5,7,1,3)$
$\rho_{-2}(q_1) = (6,8,2,4)$
$k_0 = (8,6,4,2)$
$k_1 = (7,5,3,1)$
$\rho_1(k_0) = (6,4,2,8)$
$\rho_1(k_1) = (5,3,1,7)$
$\odot : (8,18,20,14)$
$\odot : (14,20,18,8)$
$\odot : (6,12,10,56)$
$\odot : (10,12,6,56)$
$\odot : (40,42,4,6)$
$\odot : (42,40,6,4)$
$\odot : (30,28,2,24)$
$\odot : (30,24,2,28)$
$v = (1,2,3,4)$
$+ : \ d_0(B) = (22,38,38,22)$
$+ : \ d_1(B) = (16,24,16,112)$
$+ : \rho_{-2}(d_2(B)) = (82,82,10,10)$
$+ : \rho_{-2}(d_3(B)) = (60,52,4,52)$
$\rho_1(v) = (2,3,4,1)$
$\odot : (22,76,114,88)$
$\odot : (32,72,64,112)$
$\odot : (82,164,30,40)$
$\odot : (120,156,16,52)$
$+ : (54,148,178,200)$
$\rho_2 : (46,92,202,320)$
$+ : (202,320,46,92)$
$+ : (100,240,380,520)$
$Q = \begin{bmatrix}1&2\\3&4\\5&6\\7&8\end{bmatrix}, \quad K = \begin{bmatrix}8&7\\6&5\\4&3\\2&1\end{bmatrix}, \quad v^\top = \begin{bmatrix}1\\2\\3\\4\end{bmatrix}$
$B = QK^\top = \begin{bmatrix}22&16&10&4\\52&38&24&10\\82&60&38&16\\112&82&52&22\end{bmatrix}, \quad (QK^\top)v^\top = \begin{bmatrix}100\\240\\380\\520\end{bmatrix}$
Step 1
Step 2
Step 3
Step 4
Step 5
Step 6

Fig. 2: An illustrative example for $(QK^\top)v$. $\rho_j(v)$ denotes a left rotation of the vector v by j positions. In this example, both Q and K are 4×2 matrices, and v is a four-dimensional row vector, i.e., $m = 4$ and $d_h = 2$. Given column-packed ciphertexts of Q and K, we first rotate these ciphertexts to with appropriate step sizes (Step 1), then construct the rotated diagonal vectors of the matrix $B = QK^\top$ (Step 3) that are necessary for the baby-step giant-step (BSGS) algorithm, and finally apply BSGS to compute the result of $(QK^\top)v$ (Steps 4–6). Ciphertext rotations are highlighted using blue boxes. In this example, the correct result can be obtained using only 6 ciphertext rotations; see §V-F for detailed analysis.

method allows for a fast and accurate estimation of the expected maximum. In particular, if $x_0, x_1, \ldots, x_{m-1}$ are σ^2-subgaussian random variables (not necessarily independent) with mean μ, then we have

$$\mathbf{E}[x_{\mathsf{max}}] \quad \leq \quad \mu + \sqrt{2\ln m} \cdot \sigma. \tag{3}$$

Furthermore, this upper bound is tight up to a constant; see, e.g., [49, page 50]. Lim et al. [15] utilized this bound to estimate the maximum of an encrypted vector. To avoid computing square roots, they use $\sigma \approx 1.15 + 0.1 \cdot \sigma^2$.[4]

However, the input vectors to the max function may not always satisfy a σ^2-subgaussian distribution. Therefore, we adjust the coefficient in Equation (3) and instead adopt the following formula:

$$\mathbf{E}[x_{\mathsf{max}}] \quad \leq \quad \mu + k_{\mathsf{max}}\sqrt{\ln m} \cdot \sigma, \tag{4}$$

Here, k_{max} is an empirical constant that depends on the dataset; for example, for the QNLI dataset, we observe $k_{\mathsf{max}} \approx 0.944$. In addition, we use $1.25 + 0.1 \cdot \sigma^2$ to estimate the standard deviation σ, which is the minimax polynomial of degree 1 for $\sqrt{X}$ in $(0, 100)$.

1) Row-wise Summation from Rotated Diagonal Encoding: Recall that after the execution of Algorithm 1, we obtain rotated ciphertexts of diagonals of B, i.e., $\mathsf{ct}.\rho_{-ik}(d_{ik+j}(B))$ for $i \in [\ell]$ and $j \in [k]$. However, computing $\mathsf{max}(b_i)$ using Equation (4) requires row-wise summation to obtain both the mean μ and the variance σ^2. We present the following algorithm to deal with this problem.

[4]In [15], the expression was given as $\sigma \approx 0.1 + 1.15 \cdot \sigma^2$; however, this appears to be an obvious typo.

Algorithm 2 Row-wise Summation from Rotated Diagonals

```
Input: $\mathsf{ct}.\rho_{-ik}(d_{ik+j}(B))$ for $i \in [\ell]$ and $j \in [k]$, where $B \in \mathbb{R}^{m \times m}$ and $m \leq k \cdot \ell$.
Output: $\mathsf{ct}.s$ with $s = (\sum_j b_{i,j})_{i \in [m]}$.
1: for $i < [\ell]$ do
2:     $\mathsf{ct}.t_i \leftarrow \mathsf{Add}_{j \in [k]}(\mathsf{ct}.\rho_{-ik}(d_{ik+j}(B)))$.
3: $\mathsf{ct}.s \leftarrow \mathsf{Add}_{i \in [\ell]}(\mathsf{Rot}_{ik}(\mathsf{ct}.t_i))$.
```

Proposition 2. *Algorithm 2 is correct and requires at most $\ell \approx \sqrt{m}$ ciphertext rotations, assuming $k \approx \ell \approx \sqrt{m}$.*

Algorithm 3 Shifting B by $\mathbf{E}(\mathsf{max}(B))$

```
Input: $\mathsf{ct}.\rho_{-ik}(d_{ik+j}(B))$ for $i \in [\ell]$ and $j \in [k]$, where $B \in \mathbb{R}^{m \times m}$ and $m \leq k \cdot \ell$.
Output: $\mathsf{ct}.\rho_{-ik}(d_{ik+j}(B'))$ for $i \in [\ell]$ and $j \in [k]$, where $B' = (b'_{i,j}) = (b_{i,j} - g_i)$, and $g_i$ is the upper bound on $\mathbf{E}(\mathsf{max}_j(b_{i,j}))$ from Eq. (4).
1: Calling Algorithm 2 with $\mathsf{ct}.\rho_{-ik}(d_{ik+j}(B))$ for $i \in [\ell]$ and $j \in [k]$ as inputs, returns $\mathsf{ct}.s$.
2: $\mathsf{ct}.\mu \leftarrow \mathsf{CMul}(\frac{1}{m}, \mathsf{ct}.s)$.        // $\mu$: the mean vector
3: for $i < [\ell]$ do
4:     $\mathsf{ct}.\mu_i \leftarrow \mathsf{CMul}(-1, \mathsf{Rot}_{-ik}(\mathsf{ct}.\mu))$.
5:     for $j \in [k]$ do
6:         $\mathsf{ct}.\rho_{-ik}(d_{ik+j}(T)) \leftarrow \mathsf{Add}(\mathsf{ct}.\rho_{-ik}(d_{ik+j}(B)), \mathsf{ct}.\mu_i)$.
7:         Update $\mathsf{ct}.\rho_{-ik}(d_{ik+j}(T))$ to its square.
8: Calling Algorithm 2 with $\mathsf{ct}.\rho_{-ik}(d_{ik+j}(T))$ for $i \in [\ell]$ and $j \in [k]$ as inputs, returns $\mathsf{ct}.(m\sigma^2)$.
9: $\mathsf{ct}.\sigma^2 \leftarrow \mathsf{CMul}(\frac{1}{m}, \mathsf{ct}.(m\sigma^2))$.        // $\sigma^2$: the variance vector
10: $\mathsf{ct}.\sigma \leftarrow \mathsf{Add}(1.25, \mathsf{CMul}(0.1, \mathsf{ct}.\sigma^2))$.
11: $\mathsf{ct}.g \leftarrow \mathsf{Add}(\mathsf{ct}.\mu, \mathsf{CMul}(\sqrt{2\ln m}, \mathsf{ct}.\sigma))$.
12: for $i < [\ell]$ do
13:     $\mathsf{ct}.g_i \leftarrow \mathsf{CMul}(-1, \mathsf{Rot}_{-ik}(\mathsf{ct}.g))$.
14:     for $j \in [k]$ do
15:         Compute $\mathsf{ct}.\rho_{-ik}(d_{ik+j}(B'))$ as
            $\mathsf{Add}(\mathsf{ct}.\rho_{-ik}(d_{ik+j}(B)), \mathsf{ct}.g_i)$.
```

2) Shifting $\boldsymbol{B}$ by Statistical Maximum Estimation: We now consider using Equation (4) as an approximation of the vector maximum. Under the constraint that both the rotation step sizes and the diagonal packing format remain unchanged, we subtract the corresponding approximate maximum vector from each rotated diagonal vector of matrix $\boldsymbol{B}$ to prepare the inputs for the subsequent exponential function evaluation.

Proposition 3. *Algorithm 3 is correct and requires at most $4\ell \approx 4\sqrt{m}$ ciphertext rotations, assuming $k \approx \ell \approx \sqrt{m}$.*

C. Exponential Evaluation

After shift $\boldsymbol{B}$ by the statistical maximum, the input domain of the function $\mathsf{exp}(x)$ is contained within the interval $\mathbb{I}_{\mathsf{exp}}$, say $\mathbb{I}_{\mathsf{exp}} = [-60, 15]$. [5] Therefore, we focus on approximating $\mathsf{exp}(x)$ over the interval $\mathbb{I}_{\mathsf{exp}}$.

Algorithm 4 Rotated ciphertexts of diagonals of $\boldsymbol{C} = \mathsf{exp}(\boldsymbol{B})$

```
Input: $\mathsf{ct}.\rho_{-ik}(\boldsymbol{d}_{ik+j}(\boldsymbol{B}))$ for $i \in [\ell]$ and $j \in [k]$, where $\boldsymbol{B} \in \mathbb{R}^{m\times m}$ and $m \le k \cdot \ell$; a parameter $r$.
Output: $\mathsf{ct}.\rho_{-ik}(\boldsymbol{d}_{ik+j}(\boldsymbol{C}))$ for $i \in [\ell]$ and $j \in [k]$, where $\boldsymbol{C} = \mathsf{exp}(\boldsymbol{B})$.
1: for $i \in [\ell]$ do
2:     for $j \in [k]$ do
3:         Invoking the algorithm presented in [50, §4.3] to evaluate the Chebyshev-based polynomial for exp with input as $\mathsf{ct}.\rho_{-ik}(\boldsymbol{d}_{ik+j}(\boldsymbol{B}))$ returns $\mathsf{ct}.\rho_{-ik}(\boldsymbol{d}_{ik+j}(\boldsymbol{C}))$.
```

Proposition 4. *Algorithm 4 is correct and requires no ciphertext rotations.*

Approximate $\mathsf{exp}(x)$ by $(1+x/2^r)^{2^r}$ with $r = 7$ or 8, which costs $r+1$ multiplicative depths (one CMul for computing $x/2^r$, followed by r Muls to raise the result to the 2^r-th power). This method has been used in many FHE-based PPTI protocols, e.g., NEXUS [4], MOAI [8] and Tricycle [15]. Over the interval $\mathbb{I}_{\mathsf{exp}}$, the maximum absolute error (MAE) can exceed 1.12×10^6.

To obtain a numerically more stable approximation for computation, one may adopt Chebyshev basis interpolation for the exponential function. This approach has been widely used in homomorphic encryption. For example, Chen, Chillotti, and Song [50] applied this technique in the context of CKKS bootstrapping and proposed a Paterson–Stockmeyer-style [51] method to efficiently evaluate Chebyshev-based polynomials over ciphertexts. In particular, we use a polynomial of degree 44 with Chebyshev basis to approximate $\mathsf{exp}(x)$ over $\mathbb{I}_{\mathsf{exp}}$. The resulting MAE is about 1.05×10^{-5}. Evaluating this polynomial requires 7 multiplicative depths. We therefore adopt it to approximate the function exp.

D. Compute $\boldsymbol{CV}$

At this point, we have completed the construction of the ciphertexts of $\rho_{-ik}(\boldsymbol{d}_{ik+j}(\boldsymbol{C}))$, where $\boldsymbol{C} = \mathsf{exp}(\boldsymbol{B}) \in \mathbb{R}^{m\times m}$. Recall that the matrix $\boldsymbol{V} \in \mathbb{R}^{m\times d_h}$. Algorithm 5 computes the ciphertexts of the product $\boldsymbol{CV}$ in column packing.

[5]Equation (4) provides only a statistical estimate of the maximum and therefore does not guarantee that the entries of $\boldsymbol{B} - \mathbf{E}(\mathsf{max}(\boldsymbol{B}))$, i.e., the inputs to $\mathsf{exp}(x)$, are bounded from above by zero.

Algorithm 5 Ciphertexts of $\tilde{\boldsymbol{A}} = \boldsymbol{CV}$ in column packing

```
Input: $\mathsf{ct}.\rho_{-ik}(\boldsymbol{d}_{ik+j}(\boldsymbol{C}))$, where $\boldsymbol{C} \in \mathbb{R}^{m\times m}$, $m \le k \cdot \ell$, $i \in [\ell]$, and $j \in [k]$; column-packed ciphertexts of $\boldsymbol{V} \in \mathbb{R}^{m\times d_h}$: $\{\mathsf{ct}.\boldsymbol{v}_0^\top, \ldots, \mathsf{ct}.\boldsymbol{v}_{d_h-1}^\top\}$.
Output: $\{\mathsf{ct}.\tilde{\boldsymbol{a}}_0^\top, \ldots, \mathsf{ct}.\tilde{\boldsymbol{a}}_{d_h-1}^\top\}$, i.e., column-packed ciphertexts of $\tilde{\boldsymbol{A}} = \boldsymbol{CV}$.
1: for $s \in [d_h]$ do
2:     for $j \in [k]$ do
3:         $\mathsf{ct}.\rho_j(\boldsymbol{v}_s) \leftarrow \mathsf{Rot}_j(\mathsf{ct}.\boldsymbol{v}_s)$.
4:     for $i \in [\ell]$ do
5:         Compute $\mathsf{ct}.\boldsymbol{t}_i$ as:
           $\mathsf{Add}_{j\in[k]}(\mathsf{Mul}(\mathsf{ct}.\rho_{-ik}(\boldsymbol{d}_{ik+j}(\boldsymbol{C})), \mathsf{ct}.\rho_j(\boldsymbol{v}_s)))$.
6:     $\mathsf{ct}.\tilde{\boldsymbol{a}}_s^\top \leftarrow \mathsf{Add}_{i\in[\ell]}(\mathsf{Rot}_{i\cdot k}(\mathsf{ct}.\boldsymbol{t}_i))$
```

Proposition 5. *Algorithm 5 is correct and requires at most $(k+\ell-2)d_h \approx 2(\sqrt{m}-1)d_h$ ciphertext rotations, assuming $k \approx \ell \approx \sqrt{m}$.*

E. Normalization

At this point, one final step remains to complete the computation of $\boldsymbol{A} = \mathsf{softmax}(\boldsymbol{QK}^\top)\boldsymbol{V}$. As shown in Eq. (2), this step essentially multiplies each row of $\tilde{\boldsymbol{A}}$ by the ciphertext of the scalar $\frac{1}{\sum_j c_{i,j}}$, where $\boldsymbol{C} = (c_{i,j}) = \mathsf{exp}(\boldsymbol{QK}^\top)$. This functionality is implemented by Algorithm 6.

Algorithm 6 Normalization

```
Input: Ciphertexts of $\tilde{\boldsymbol{A}} = \mathsf{exp}(\boldsymbol{QK}^\top)\boldsymbol{V} \in \mathbb{R}^{m\times d_h}$ in column packing: $\{\mathsf{ct}.\tilde{\boldsymbol{a}}_0^\top, \ldots, \mathsf{ct}.\tilde{\boldsymbol{a}}_{d_h-1}^\top\}$; the outputs of Algorithm 4, i.e., $\mathsf{ct}.\rho_{-ik}(\boldsymbol{d}_{ik+j}(\boldsymbol{C}))$ with $\boldsymbol{C} = \mathsf{exp}(\boldsymbol{QK}^\top)$ for $i \in [\ell]$ and $j \in [k]$, where $m \le k \cdot \ell$.
Output: Ciphertexts of $\boldsymbol{A} = \mathsf{softmax}(\boldsymbol{QK}^\top)\boldsymbol{V} \in \mathbb{R}^{m\times d_h}$ in column packing: $\{\mathsf{ct}.\boldsymbol{a}_0^\top, \ldots, \mathsf{ct}.\boldsymbol{a}_{d_h-1}^\top\}$.
1: Calling Algorithm 2 with $\mathsf{ct}.\rho_{-ik}(\boldsymbol{d}_{ik+j}(\boldsymbol{C}))$ for $i \in [\ell]$ and $j \in [k]$ as inputs, returns $\mathsf{ct}.\boldsymbol{s}$.    // $\boldsymbol{s} = (\sum_j c_{i,j})_{i\in[m]}$
2: Computing the inverse of each component of $\boldsymbol{s}$ using a Chebyshev polynomial approximation followed by two iterations of the Goldschmidt algorithm [52] yields the ciphertext $\mathsf{ct}.\boldsymbol{s} \leftarrow \mathsf{ct}.\big((1/s_i)_i\big)$.
3: for $i \in [d_h]$ do
4:     $\mathsf{ct}.\boldsymbol{a}_i^\top \leftarrow \mathsf{Mul}(\mathsf{ct}.\boldsymbol{s}, \mathsf{ct}.\tilde{\boldsymbol{a}}_i^\top)$.
```

Proposition 6. *Algorithm 6 is correct, and requires $\ell - 1 \approx \sqrt{m} - 1$ ciphertext rotations, assuming $\ell \approx \sqrt{m}$.*

We remark that Steps 1–2 in Algorithm 6 could, in principle, be moved immediately after Algorithm 4, since their computations depend only on the outputs $\mathsf{ct}.\rho_{-ik}(\boldsymbol{d}_{ik+j}(\boldsymbol{C}))$ returned by Algorithm 4. However, such reordering is not recommended for Steps 3-4. The reason is that the outputs of Algorithm 4 are ciphertexts of rotated diagonals $\boldsymbol{d}_{ik+j}(\boldsymbol{C})$, where the rotation step sizes vary with (i, j). As a result, multiplying these ciphertexts directly with $\mathsf{ct}.\boldsymbol{s}$ would not yield the correct normalization result. For example, when $i = 1$ and $j = 0$, Algorithm 4 produces the ciphertext $\mathsf{ct}.\rho_{-k}(\boldsymbol{d}_k(\boldsymbol{C}))$, which must be multiplied with $\mathsf{Rot}_{-k}(\mathsf{ct}.\boldsymbol{s})$ to ensure correctness. This implies that an additional $\ell - 1$ rotations must be applied to $\mathsf{ct}.\boldsymbol{s}$. Furthermore, completing the normalization would then require $m = 128$ ciphertext multiplications. In contrast, by following Algorithm 6 as designed—delaying normalization until after the $\boldsymbol{CV}$ computation—only $d_h = 64$ ciphertext multiplications are needed. This optimization justifies our decision to defer normalization.

F. Analysis of Our Solution for Attention

By cascading Algorithm 1 through Algorithm 6, we obtain our complete solution for encrypted attention computation. Given column-packed ciphertexts of $\boldsymbol{Q}$, $\boldsymbol{K}$, and $\boldsymbol{V}$ as input, the module outputs column-packed ciphertexts of $\boldsymbol{A} = \mathsf{softmax}(\boldsymbol{QK}^\top)\boldsymbol{V}$. This computation is repeated H times within each Transformer layer. We now count the total number of ciphertext rotations required to perform H computation of this module over n input sequences, each of length m and embedding dimension d in Table III.

TABLE III: Comparison of the number of ciphertext rotations required by CCMM in the computation of $\mathsf{softmax}(\boldsymbol{QK}^\top)\boldsymbol{V}$ for all H heads in one Transformer layer. The concrete numbers are computed with the hyperparameters of BERT-Base given in Table II. $N = 2^{16}$ is the ring dimension for CKKS, $m = 128$ is the length of each sequence, and $n = N/(2m)$ is the number of sequences ($\sqrt{m} \approx 12$).

	MOAI [8]		ARION (Ours)	
	Equation	Number	Equation	Number
$\boldsymbol{B} = \boldsymbol{QK}^\top$	md_hH	98 304	$2\sqrt{m}d_hH$	18 432
$\boldsymbol{B} - \mathsf{max}(\boldsymbol{B})$	$-^*$	–	$4\sqrt{m}H$	576
$\boldsymbol{C} = \mathsf{exp}(\boldsymbol{B})$	0	0	0	0
$\tilde{\boldsymbol{A}} = \boldsymbol{CV}$	$(m + 2\sqrt{m})d_hH^\dagger$	116 736	$2\sqrt{m}d_hH$	18 432
Normalization	0	0	$\sqrt{m}H$	144
Total (H heads)	$2(m + \sqrt{m})d_hH$	215 040	$(4d_h + 5)\sqrt{m}H$	**37 584**
Per Sequence	$\frac{4(m+\sqrt{m})d_hH}{N/m}$	840	$\frac{(8d_hH+10)\sqrt{m}}{N/m}$	**147**

* Since MOAI [8] uses a pre-defined constant as the estimation of $\mathsf{max}(\boldsymbol{B})$, no ciphertext rotations are needed. † In [8], this formula was written as $(m + \sqrt{m})d_hH$. Here, we correct the term of $\sqrt{m}$ to $2\sqrt{m}$.

As shown in Table III, the complexity of MOAI is dominated by linear terms of the sequence length m, resulting in $O(md_hH)$ rotations. In contrast, ARION leverages the double-BSGS algorithm to reduce this complexity to $O(\sqrt{m}d_hH)$. Specifically, for BERT-Base under a CKKS configuration with ring dimension $N = 2^{16}$, ARION reduces the number of ciphertext rotations by approximately 82.5% compared with MOAI.

To contextualize this improvement within the broader landscape, we benchmark ARION against prominent protocols including BOLT [36], NEXUS [4], PowerFormer [7], and THOR [20]. Table IV summarizes the total rotation overhead across all 12 layers of BERT-Base. Note that BOLT and PowerFormer utilize high-order approximations. Furthermore, unlike THOR and PowerFormer which reflect single-sequence costs, the remaining protocols (including ARION) leverage batch inference to report amortized overheads.

BOLT and NEXUS incur high overheads. PowerFormer and THOR target single-sequence inference, exhibiting the number of required ciphertext rotations quadratic and linear with m, respectively. ARION requires essentially fewer rotations than all these methods for encrypted attention computation by saving at least a factor of $\sqrt{m}$ asymptotically.

TABLE IV: Total required ciphertext rotations for attention computation (L layers, H heads) on BERT-Base.

Protocols	Equation	Number (Total)
BOLT† [36]	$(\frac{188md}{N} + 4mH\log m + 2mH)L$	$> 556\ 334$
NEXUS‡ [4]	$\frac{4m}{N}(md + md_hH + 2d^2)L$	786 432
PowerFormer† [7]	$(\frac{3dH(\sqrt{3m}+d_h)}{2m} + \frac{H^2\sqrt{m}}{2} + \frac{13mH}{4})L$	$> 178\ 020$
THOR [20]	$(\frac{1}{32}m^2 + \frac{H+59}{16}m + \frac{3H+5}{16}d_h)L$	14 926
MOAI‡ [8]	$\frac{4(m+\sqrt{m})d_hH}{N/m}L$	10 080
ARION (Ours)‡	$\frac{(8d_hH+10)\sqrt{m}}{N/m}L$	**1 764**

The formulas are based on [20, Tab. 4] and [8]. The parameters are identical to those in Table III. † The equation is the high-order approximations. ‡ Costs are amortized over the batch.

G. Feed-Forward and Classification Layers

After the attention module, since each ciphertext is column-packed, the concatenation step requires no additional operation (as shown in Fig. 1); all concatenated data can be accessed directly by indexing the ciphertexts. Moreover, outside the attention module, all linear operations are ciphertext-plaintext matrix multiplications (CPMM), with both input and output ciphertexts in column packing. Therefore, the C2C CPMM method described in §III-C2 can be directly applied, *without* any ciphertext rotations.

Regarding the non-linear operations involved in these layers (including $1/\sqrt{x}$ in $\mathsf{LayerNorm}$, GELU in FFN, and Tanh in Classification), we employ high-precision Chebyshev polynomial approximations combined with Newton iterations. The detailed implementation strategies and specific approximation parameters are provided in Appendix D.

At this point, every computational module involved in Fig. 1 has been instantiated with a concrete solution, and we refer to the resulting PPTI protocol as *ARION*.

VI. Security Analysis

We analyze the security of ARION under the semi-honest adversary model. The protocol relies on the underlying CKKS scheme, which offers provable IND-CPA security under the RLWE assumption [44]. Since the server performs all inference operations—including matrix multiplications and non-linear approximations—exclusively on ciphertexts without access to the client's secret key, its view remains computationally indistinguishable from random noise. We provide a rigorous formal proof based on the simulation paradigm in Appendix F.

VII. Implementation

We implemented ARION using the CKKS scheme in the homomorphic encryption library Lattigo [22], covering all modules in Fig. 1 and all features described above. To the best of our knowledge, this is the first PPTI protocol based on Lattigo, which supports end-to-end, non-interactive encrypted inference for both BERT-Tiny and BERT-Base.

A. Additional Features

To maximize performance and versatility, our implementation incorporates several additional features. These include minimizing multiplicative depth through mathematical folding of constants, accelerating ciphertext rotations via the Hoisting technique [53], and employing task-level parallelization to mask latency in the normalization step. Furthermore, we designed a flexible batching strategy that adapts to different token lengths. Detailed descriptions of these features are provided in Appendix E.

B. Bootstrapping Placement

In each Transformer layer, we use five bootstrapping operations in total: one within the attention module, one after the attention module, and three surrounding the two LayerNorm operations (after LayerNorm-1, before and after LayerNorm-2). Compared to MOAI, we use one additional bootstrapping operation. This is mainly because we employ statistical maximum estimation instead of a predefined constant, resulting in a wider input range for the exp function, which in turn requires higher-degree polynomial approximations for both exp and $1/x$. While this incurs additional computational cost, it preserves information of actual data distribution (e.g., mean and standard deviation) and enhances robustness under dynamic input data. As will be shown in the next section, despite this extra cost, ARION is still faster than MOAI.

VIII. Evaluation

A. Setup

All experiments in this section were conducted on a workstation equipped with Intel Xeon 6530 CPU (32 × 2 cores @ 2.1 GHz). For the CKKS parameters, we fixed the ring dimension at $N = 2^{16}$, set the remaining number of levels after bootstrapping to 14, and configured the ciphertext modulus q such that $\log q = 58$ (last modulus) $+45 \cdot 14$ (for 14 levels) $+60 \cdot 4$ (for relinearization). The scaling factor Δ satisfies $\log \Delta \approx 45$. Under this setting, the modulus at bootstrapping satisfies $\log q = 1749$. According to the latest Lattice Estimator [54], these parameters achieve a 128-bit security level.

B. Micro-Benchmarks

1) Double-BSGS for Three Ciphertexts Matrix Multiplication (TCMM): To evaluate the performance improvement of the double-BSGS algorithm, we consider three ciphertexts matrix multiplication (TCMM). Let $\boldsymbol{Q}$, $\boldsymbol{K}$, and $\boldsymbol{V}$ be matrices of the same dimension $m \times d$. Given their ciphertexts in column-packing, the goal is to compute the ciphertexts in column-packing of $\boldsymbol{Q}\boldsymbol{K}^\top\boldsymbol{V}$. In Table V, we give a comparison between the method without (used in MOAI [8]) and with (used in ARION) double-BSGS for TCMM, where d is fixed to 64. All experiments are run on a single-thread.

As shown in Table V, for the "without" method, both the time consumed by Rot operations and the total runtime exhibit a clear linear relationship with respect to m. In contrast, for the "with" method, although the total runtime still increases linearly with m, the growth in Rot time is much slower, resulting in the performance gap between the two methods also growing linearly with m. Notably, when $m = 256$, the double-BSGS algorithm saves approximately 88.5% in ciphertext rotation time and 44.3% in total runtime.

TABLE V: TCMM w./w.o. Double-BSGS

(m, d)	Double-BSGS	Rot Time (s)	Total Time (s)
$(32, 64)$	without	7.26	13.59
	with	2.28	8.75
$(64, 64)$	without	14.79	27.33
	with	3.21	16.61
$(128, 64)$	without	29.85	54.43
	with	4.81	31.44
$(256, 64)$	without	59.13	109.88
	with	6.82	61.22

Using CKKS in Lattigo with $\log N = 13$, $\log q = 190$, and $\log \Delta = 40$.

TABLE VI: Runtime of Matrix Operations in One Encrypted Attention Layer in a Multi-Threaded Environment

Model	Without D-BSGS (s)	With D-BSGS (s)	Speedup
BERT-Tiny	365.5	196.8	**1.86**×
BERT-Base	2239.3	1184.4	**1.89**×
LLaMA-3-8B†	10051.2	5694.1	**1.76**×

The input sequence length is fixed at $m = 128$ for all experiments. The reported runtimes represent the total time to process a batch of $n = 256$ inputs using 64 threads. † Refers to the LLaMA-3-8B architecture with hidden size $d = 4096$ and heads $H = 32$.

2) Performance on Matrix Operations in Attention Layer in Multi-Threaded Environment: To assess performance in a practical setting, we measured the total runtime of matrix operations for a single encrypted attention layer using multi-threaded execution. The evaluation covers three typical LLMs: BERT-Tiny, BERT-Base, and LLaMA-3-8B.

As summarized in Table VI, the double-BSGS optimization delivers a consistent speedup ranging from 1.76x to 1.89x across all model scales. Notably, for the large-scale LLaMA-3-8B model, our algorithm reduces the computation time by over 4350 seconds. This reduction in absolute runtime demonstrates that our optimizations can be directly applied to larger LLMs, such as LLaMA.

C. End-to-End Performance

We evaluate encrypted inference accuracy on three representative classification tasks: QNLI, which determines whether a context sentence answers a given question; RTE, which tests whether a premise entails a hypothesis; and SST-2, which assesses the sentiment polarity of movie reviews. These benchmarks enable reliable assessment of inference correctness under fine-tuned model conditions.

1) Accuracy: For BERT-Tiny, we evaluate ARION on both the pretrained model and variants fine-tuned on QNLI, RTE, and SST-2. Encrypted computation is carried out up to the pooler layer, with the classification head executed in plaintext to verify prediction correctness. For BERT-Base, we assess numerical accuracy layer by layer in direct comparison with MOAI without fine-tuning.

a) BERT-Tiny: As shown in Figure 3, the absolute mean errors across Layer 0, Layer 1, and the pooler remain consistently low for all variants, with most values below 5×10^{-4}. Although the QNLI-finetuned model exhibits slightly higher errors, they remain within a negligible range. These results confirm that ARION maintains high numerical fidelity

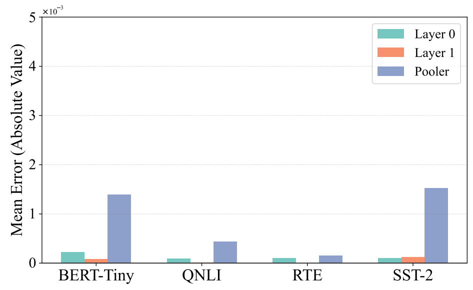


Fig. 3: Absolute mean error of Layer 0, Layer 1, and the Pooler across BERT-Tiny models.

throughout the encrypted inference process. Consequently, the encrypted logits deviate only marginally from their plaintext counterparts, yielding identical classification outcomes. We provide concrete examples of input sentences and their corresponding logits in Appendix H.

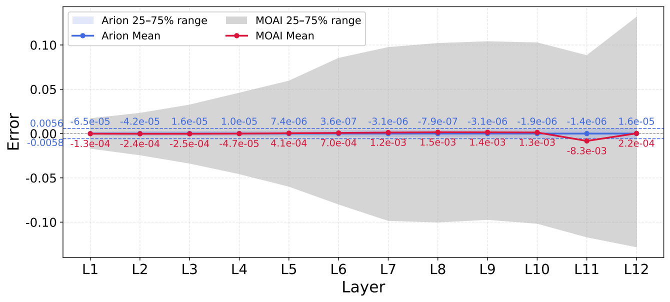


Fig. 4: Full layer-wise error propagation in BERT-Base.

b) BERT-Base: Figure 4 compares the numerical stability of ARION and MOAI across all 12 layers. ARION demonstrates superior stability, maintaining mean errors close to zero with a tight interquartile range. Even after the 12th layer, the error remains bounded within approximately ± 0.005. In contrast, MOAI exhibits significant error accumulation, with its error range expanding to nearly ± 0.1 after the seventh layer. This comparison highlights ARION's ability to preserve high precision during deep encrypted inference.

2) Runtime: Table VII provides a detailed breakdown of component-wise latency for encrypted inference on both BERT-Tiny and BERT-Base models under the ARION framework, as well as direct comparisons with state-of-the-art protocols. All experiments are conducted using 64 threads.

For BERT-Tiny, ARION achieves a remarkable 34.6x speedup in amortized per-sample inference time compared to the most recent SoTA result reported by Lim et al. [15] (Tricycle). For BERT-Base, ARION delivers a 2.5x speedup over MOAI, further demonstrating its efficiency at scale.

Since MOAI is implemented using the SEAL library, whereas ARION is built on top of Lattigo, we provide in Appendix G a detailed normalization method to assess the impact of different FHE libraries on the reported runtimes. Here, we highlight two additional critical distinctions: (1) Similar to NEXUS, MOAI estimates the attention maximum using a fixed, whereas ARION employs statistical maximum estimation, thus handling a more complex and computationally intensive attention step; (2) MOAI's runtime was measured on 112 threads, while ARION achieves superior performance using only 64 threads. Taking all these factors into account, the 2.5x speedup can be viewed as a relatively conservative estimate.

D. Token-Length Flexibility

To further evaluate scalability across different input sizes, Table VIII reports the average per-input latency of BERT-Tiny inference under varying maximum token lengths, ranging from 16 to 128. It is important to note that the latency in each configuration reflects the cost associated with the maximum token length, regardless of the actual number of tokens in a given input. As long as the input sequence does not exceed this maximum, the per-input latency remains constant. This configuration yields predictable runtime behavior and enables efficient handling of variable-length inputs within a bounded computational budget. These characteristics validate the practicality of ARION in real-world scenarios where input lengths may vary.

E. Apply to the iDASH 2024 Dataset

To demonstrate the versatility of ARION beyond standard NLP tasks, we deployed it for protein sequence classification in the iDASH 2024 Secure Genome Analysis Competition. Targeting the DASHformer model ($L = 1, d = 128, m = 50$) [23], our solution, which implements the proposed double-BSGS algorithm and a more aggressive activation approximation, was awarded one of the two first places in Track 1.

As detailed in Table IX, our solution completes the inference for 163 encrypted protein sequences of length 50 in 165 seconds (amortized about 1 second per sequence), achieving a 3.78x speedup over the Bossuat solution [55]. This significant performance gain stems from our algorithmic optimizations, which reduced the multiplicative depth to 10, thereby allowing for leveled homomorphic encryption without expensive bootstrapping. The dramatic reduction in runtime validates the practical efficiency of ARION in latency-sensitive scenarios.

IX. Conclusion

We present ARION, an efficient, non-interactive, FHE-based PPTI protocol, featured by its highly efficient and accurate attention module. Together with our thread-level parallel implementation, our open-sourced implementation achieves significant performance improvements, accelerating encrypted inference by 34.6x on BERT-Tiny and 2.5x on BERT-Base, respectively, compared to state-of-the-art PPTI protocols.

References

[1] J. Devlin, M. Chang, K. Lee *et al.*, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *NAACL '19*, J. Burstein, C. Doran, and T. Solorio, Eds. ACL, 2019, pp. 4171–4186, https://doi.org/10.18653/v1/N19-1423.

[2] T. Xu, W. 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 2025, Seattle, WA, USA, August 13-15, 2025.* USENIX Association, 2025, pp. 2653–2672, https://ia.cr/2025/1532.

TABLE VII: BERT encrypted inference on $n = 256$ inputs with $m = 128$ tokens (amortized over n inputs in seconds)

Module	Operations	Data Dimension · Number of Executions	BERT-Tiny		BERT-Base	
			Tricycle [15] *	ARION (Ours)	MOAI †[8]	ARION (Ours)
Attention	CPMM	$(\mathbb{R}^{m\times d} \times \mathbb{R}^{d\times d_h}) \cdot 3HL$	14.5	0.07	37.4	13.35
	CCMM	$(\mathbb{R}^{m\times d} \times \mathbb{R}^{d\times m}) \cdot HL$	10.7	1.16	40.3	24.28
	max	$(\mathbb{R}^{m\times m}) \cdot HL$		0.03		0.57
	exp	$(\mathbb{R}^{m\times m}) \cdot HL$	14.2	0.16	54.7	3.29
	CCMM+Bts+Reciprocal	$(\mathbb{R}^{m\times m} \times \mathbb{R}^{m\times d_h}) \cdot HL$		0.25		5.01
	Bts	$(\mathbb{R}^{m\times d}) \cdot L$		1.76	95.4	32.89
Self Output	CPMM	$(\mathbb{R}^{m\times d} \times \mathbb{R}^{d\times d}) \cdot L$	2.1	0.03	1.7	5.20
	LayerNorm-1	$(\mathbb{R}^{m\times d}) \cdot L$	8.2	0.06	0.6	0.43
	Bts	$(\mathbb{R}^{m\times d}) \cdot L$		1.67	95.8	33.29
FFN	CPMM	$(\mathbb{R}^{m\times d} \times \mathbb{R}^{d\times d_f}) \cdot L$		0.08	44.1	19.80
	GELU	$(\mathbb{R}^{m\times d_f}) \cdot L$	13.2	0.47	3.3	14.70
	CPMM	$(\mathbb{R}^{m\times d_f} \times \mathbb{R}^{d_f\times d}) \cdot L$		0.02	7.1	4.89
	Bts	$(\mathbb{R}^{m\times d}) \cdot L$		1.76	98.8	33.49
Final	LayerNorm-2	$(\mathbb{R}^{m\times d}) \cdot L$	9.0	0.04	0.6	0.41
	Bts	$(\mathbb{R}^{m\times d}) \cdot L$	234.6	1.74	94.8	33.13
Total			322.0	**9.30**	574.6	**224.91**

Parameters apart from n and m can be found in Table II. $*$ The data for Tricycle are borrowed from [15], where the last Bts is the sum of all bootstrapping time. In addition, the time is not amortized since Tricycle does not support batch inference. † The data for MOAI are taken from [8], where the time 54.7s include softmax and CCMM. Additionally, the first Bts for MOAI happens just before LayerNorm-1, different from ARION.

TABLE VIII: Latency of ARION-encrypted BERT-Tiny inference under varying maximum token lengths

Max Token Length	Batch Size	Amortized Runtime (s)
16	2048	1.06
32	1024	2.21
64	512	4.46
128	256	9.30

TABLE IX: Performance encrypted DASHformer inference

Solution	Bts	Memory(GB)	Total Time(s)	Amortized Time(s)
Bossuat-I [55]	76	58	890	5.46
Bossuat-II [55]	58	56	624	3.83
Ours	**0**	**29**	**165**	**1.01**

[3] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *STOC '09*, M. Mitzenmacher, Ed. New York: ACM, 2009, pp. 169–178, https://doi.org/10.1145/1536414.1536440.

[4] J. Zhang, X. Yang, L. He, K. Chen, W. jie Lu, Y. Wang, X. Hou, J. Liu, K. Ren, and X. Yang, "Secure transformer inference made non-interactive," in *NDSS '25*. Fredericksburg: The Internet Society, 2025, https://dx.doi.org/10.14722/ndss.2025.230868.

[5] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *NIPS '17*. New York: Curran Associates, Inc., 2017, https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[6] J. Moon, Z. Omarov, D. Yoo, Y. An, and H. Chung, "Adaptive successive over-relaxation method for a faster iterative approximation of homomorphic operations," 2024, https://ia.cr/2024/1366.

[7] D. Park, E. Lee, and J.-W. Lee, "Powerformer: Efficient privacy-preserving transformer with batch rectifier-power max function and optimized homomorphic attention," in *ACL '25*. ACL, 2025, pp. 11 090–11 111, https://aclanthology.org/2025.acl-long.543.pdf.

[8] L. Zhang, X. Wang, J. J. Sim, Z. Huang, J. Zhong, H. Wang, P. Duan, and K. Y. Lam, "MOAI: Module-optimizing architecture for non-interactive secure transformer inference," 2025, https://ia.cr/2025/991.

[9] A. Radford, K. Narasimhan, T. Salimans *et al.*, "Improving language understanding by generative pre-training," 2018, https://openai.com/research/language-unsupervised.

[10] A. Grattafiori, A. Dubey, A. Jauhri *et al.*, "The Llama 3 herd of models," 2024, https://arxiv.org/pdf/2407.21783.

[11] I. Zimerman, M. Baruch, N. Drucker, G. Ezov, O. Soceanu, and L. Wolf, "Converting transformers to polynomial form for secure inference over homomorphic encryption," in *ICML '24*. PMLR, 2024, pp. 62 803–62 814, https://proceedings.mlr.press/v235/zimerman24a.html.

[12] E. Aharoni, A. Adir, M. Baruch, N. Drucker, G. Ezov, A. Farkash, L. Greenberg, R. Masalha, G. Moshkowich, D. Murik, H. Shaul, and O. Soceanu, "HeLayers: A tile tensors framework for large neural networks on encrypted data," *PoPETs*, vol. 2023, no. 1, pp. 325–342, 2023, https://doi.org/10.56553/popets-2023-0020.

[13] I. Zimerman, A. Adir, E. Aharoni, M. Avitan, M. Baruch, N. Drucker, J. Lerner, R. Masalha, R. Moshe, and O. Soceanu, "Powersoftmax: Towards secure LLM inference over encrypted data," 2024, https://arxiv.org/abs/2410.09457.

[14] L. Rovida and A. Leporati, "Transformer-based language models and homomorphic encryption: An intersection with BERT-tiny," in *IWSPA '24*. New York: ACM, 2024, pp. 3–13, https://doi.org/10.1145/3643651.3659893.

[15] L. Lim, V. Kalagi, D. Agrawal, and A. E. Abbadi, "Tricycle: Private transformer inference with tricyclic encodings," 2025, https://ia.cr/2025/1200.

[16] A. Al Badawi *et al.*, "OpenFHE: Open-source fully homomorphic encryption library," in *Proceedings of the 10th Workshop on Encrypted Computing & Applied Homomorphic Cryptography (Los Angeles, USA, 7 November 2022)*, M. Brenner, A. Costache, and K. Rohloff, Eds. New York: ACM, 2022, pp. 53–63, https://doi.org/10.1145/3560827.3563379.

[17] Microsoft, "Microsoft SEAL (release 4.1.1)," Accessed in July, 2023, https://github.com/microsoft/SEAL.

[18] 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 *ICML '22*. PMLR, 2022, pp. 12 403–12 422, https://proceedings.mlr.press/v162/lee22e.html.

[19] H. Yang, S. Shen, W. Dai, L. Zhou, Z. Liu, and Y. Zhao, "Phantom: A CUDA-accelerated word-wise homomorphic encryption library," *IEEE Transactions on Dependable and Secure Computing*, vol. 21, no. 5, pp. 4895–4906, 2024, https://doi.org/10.1109/TDSC.2024.3363900.

[20] J. Moon, D. Yoo, X. Jiang, and M. Kim, "THOR: Secure transformer inference with homomorphic encryption," in *CCS '25*. New York: ACM, 2025, pp. 3765 – 3779, https://doi.org/10.1145/3719027.3765150.

[21] DESILO, "Liberate.FHE v0.9.0," https://github.com/Desilo/liberate-fhe, 2023.

[22] Tune-Insight, "Lattigo v6.1.1," https://github.com/tuneinsight/lattigo, 2025.

[23] A. Harmanci, L. Chen, M. Kim *et al.*, "Descriptor: Benchmarking secure neural network evaluation methods for protein sequence classification (iDASH24)," *IEEE Data Descriptions*, 2024, https://doi.org/10.1109/IEEEDATA.2024.3482283.

[24] L. K. L. Ng and S. S. M. Chow, "SoK: Cryptographic neural-network computation," in *S& P '23*. Los Alamitos: IEEE, 2023, pp. 497–514.

[25] W. Zeng, T. Xu, Y. Chen, Y. Zhou, M. Zhang, J. Tan, C. Hong, and M. Li,

"Towards efficient privacy-preserving machine learning: A systematic review from protocol, model, and system perspectives," 2025, https://arxiv.org/abs/2507.14519.

[26] D. Li, H. Wang, R. Shao, H. Guo, E. P. Xing, and H. Zhang, "MPCFORMER: fast, performant and provate transformer inference with MPC," in *ICLR 2023*. OpenReview.net, 2023, https://openreview.net/forum?id=CWmvjOEhgH-.

[27] Y. Dong, W. jie Lu, Y. Zheng, H. Wu, D. Zhao, J. Tan, Z. Huang, C. Hong, T. Wei, and W. Chen, "PUMA: Secure inference of LLaMA-7B in five minutes," 2023, https://arxiv.org/abs/2307.12533.

[28] W. Zeng, M. Li, W. Xiong, T. Tong, W.-J. Lu, J. Tan, R. Wang, and R. Huang, "MPCViT: Searching for accurate and efficient MPC-friendly vision transformer with heterogeneous attention," in *ICCV '23*. Los Alamitos: IEEE Computer Society, 2023, pp. 5029–5040, https://doi.org/10.1109/ICCV51070.2023.00466.

[29] M. Zheng, Q. Lou, and L. Jiang, "Primer: Fast private transformer inference on encrypted data," in *DAC '23*, 2023, pp. 1–6, https://doi.org/10.1109/DAC56929.2023.10247719.

[30] Y. Zhang, D. Chen, S. Kundu, C. Li, and P. A. Beerel, " SAL-ViT: Towards latency efficient private inference on ViT using selective attention search with a learnable softmax approximation," in *ICCV '23*. Los Alamitos: IEEE Computer Society, 2023, pp. 5093–5102, https://doi.org/10.1109/ICCV51070.2023.00472.

[31] D. Chen, Y. Zhang, S. Kundu, C. Li, and P. A. Beerel, "RNA-ViT: Reduced-dimension approximate normalized attention vision transformers for latency efficient private inference," in *ICCAD '23*. Piscataway: IEEE, 2023, pp. 1–9, https://doi.org/10.1109/ICCAD57390.2023.10323702.

[32] Y. Chen, X. Meng, Z. Shi, Z. Ning, and J. Lin, "SecureTLM: Private inference for transformer-based large model with MPC," *Information Sciences*, vol. 667, p. 120429, 2024, https://doi.org/10.1016/j.ins.2024.120429.

[33] K. Gupta, N. Jawalkar, A. Mukherjee, N. Chandran, D. Gupta, A. Panwar, and R. Sharma, "SIGMA: Secure GPT inference with function secret sharing," *PoPETs*, vol. 2024, no. 4, pp. 61–79, 2024, https://doi.org/10.56553/popets-2024-0107.

[34] J. Luo, Y. Zhang, Z. Zhang, J. Zhang, X. Mu, H. Wang, Y. Yu, and Z. Xu, "SecFormer: Fast and accurate privacy-preserving inference for transformer models via SMPC," in *ACL '24*. Bangkok, Thailand: ACL, 2024, pp. 13 333–13 348, https://doi.org/10.18653/v1/2024.findings-acl.790.

[35] M. Hao, H. Li, H. Chen *et al.*, "Iron: Private inference on transformers," in *NeurIPS '22*. Curran Associates, 2022, pp. 15 718–15 731, https://proceedings.neurips.cc/paper_files/paper/2022/file/64e2449d74f84e5b1a5c96ba7b3d308e-Paper-Conference.pdf.

[36] Q. Pang, J. Zhu, H. Möllering, W. Zheng, and T. Schneider, "BOLT: Privacy-preserving, accurate and efficient inference for transformers," in *IEEE S & P '24*. Los Alamitos: IEEE, 2024, pp. 133–133, https://doi.org/10.1109/SP54263.2024.00130.

[37] W.-J. Lu, Z. Huang, Z. Gu, J. Li, J. Liu, C. Hong, K. Ren, T. Wei, and W. Chen, "BumbleBee: Secure two-party inference framework for large transformers," in *NDSS '25*, 2025, https://doi.org/10.14722/ndss.2025.230057.

[38] K. Cheng, Y. Xia, A. Song, J. Fu, W. Qu, Y. Shen, and J. Zhang, "Mosformer: Maliciously secure three-party inference framework for large transformers," in *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security, CCS 2025, Taipei, Taiwan, October 13-17, 2025*. ACM, 2025, pp. 4124–4138, https://doi.org/10.1145/3719027.3765028.

[39] A. Y. L. Kei and S. S. M. Chow, "SHAFT: Secure, handy, accurate and fast transformer inference," in *NDSS '25*. The Internet Society, 2025, https://doi.org/10.14722/ndss.2025.242287.

[40] T. Chen, H. Bao, S. Huang *et al.*, "THE-X: Privacy-preserving transformer inference with homomorphic encryption," in *ACL '22*. Stroudsburg: ACL, 2022, pp. 3510–3520, https://doi.org/10.18653/v1/2022.findings-acl.277.

[41] J. Chen, L. Yang, C. Yang, S. Wang, R. Li, W. Miao, W. Wu, L. Yang, K. Wu, and L. Dai, "Secure transformer-based neural network inference for protein sequence classification," 2024, https://ia.cr/2024/1851.

[42] Y. Li, X. Zhou, Y. Wang, L. Qian, and J. Zhao, "Private transformer inference in MLaaS: A survey," 2025, https://arxiv.org/abs/2505.10315.

[43] J. Chen, L. Yang, W. Wu, Y. Liu, and Y. Feng, "Homomorphic matrix operations under bicyclic encoding," *IEEE Trans. Inf. Forensics Secur.*, vol. 20, pp. 1390–1404, 2025, https://doi.org/10.1109/TIFS.2024.3490862, https://ia.cr/2024/1762.

[44] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *ASIACRYPT 2017*, ser. LNCS. Heidelberg: Springer, 2017, vol. 10624, pp. 409–437, https://doi.org/10.1007/978-3-319-70694-8_15.

[45] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," *Journal of ACM*, vol. 60, no. 6, pp. 43:1–35, 2013, https://doi.org/10.1145/2535925.

[46] X. Jiang, M. Kim, K. Lauter, and Y. Song, "Secure outsourced matrix computation and application to neural networks," in *CCS '18*. New York: ACM, 2018, pp. 1209–1222, https://doi.org/10.1145/3243734.3243837.

[47] S. Halevi and V. Shoup, "Algorithms in HElib," in *CRYPTO '14*, ser. LNCS. Heidelberg: Springer, 2014, vol. 8616, pp. 554–571, https://doi.org/10.1007/978-3-662-44371-2_31.

[48] ——, "Bootstrapping for HElib," in *EUROCRYPT '15*, ser. LNCS. Heidelberg: Springer, 2015, vol. 9056, pp. 641–670, https://doi.org/10.1007/978-3-662-46800-5_25.

[49] R. van Handel, "Probability in high dimension," 2016, https://web.math.princeton.edu/~rvan/APC550.pdf.

[50] H. Chen, I. Chillotti, and Y. Song, "Improved bootstrapping for approximate homomorphic encryption," in *EUROCRYPT 2019*, ser. LNCS. Cham: Springer, 2019, vol. 11477, pp. 34–54, https://doi.org/10.1007/978-3-030-17656-3_2.

[51] M. S. Paterson and L. J. Stockmeyer, "On the number of nonscalar multiplications necessary to evaluate polynomials," *SIAM Journal on Computing*, vol. 2, no. 1, pp. 60–66, 1973, https://doi.org/10.1137/0202007.

[52] R. E. Goldschmidt, "Applications of division by convergence," Master's thesis, MIT, 1964, https://dspace.mit.edu/bitstream/handle/1721.1/11113/34136725-MIT.pdf.

[53] S. Halevi and V. Shoup, "Faster homomorphic linear transformations in HElib," in *CRYPTO '18*, ser. LNCS. Cham: Springer, 2018, vol. 10991, pp. 93–120, https://doi.org/10.1007/978-3-319-96884-1_4.

[54] M. R. Albrecht, R. Player, and S. Scott, "On the concrete hardness of learning with errors," *Journal of Mathematical Cryptology*, vol. 9, no. 3, pp. 169–203, 2015, https://doi.org/10.1515/jmc-2015-0016. Lattice Estimator: https://github.com/malb/lattice-estimator.

[55] J.-P. Bossuat, "A solution for iDASH 2024 - HE track," 2024, https://github.com/gausslabs/idash-2024-solution.

Appendix A
The CKKS scheme

The Cheon-Kim-Kim-Song scheme (CKKS) [44] is one of the most popular fully homomorphic encryption (FHE) schemes for PPML since it supports approximate arithmetic operations. In CKKS, the *plaintext space* is a subset of $R = \mathbb{Z}[X]/\langle X^N + 1\rangle$ while *messages* are complex vectors in $\mathbb{C}^{N/2}$, where N is a power-of-two integer. The *ciphertext space* of CKKS is R/qR with a large integer q, called *ciphertext modulus*. CKKS naturally supports single-instruction-multiple-data (SIMD) operations, i.e., performing an operation on a ciphertext corresponds to performing the same operation on $N/2$ *slots* of $\boldsymbol{m}$ in parallel. For $\boldsymbol{x} = (x_i)_{0\le i<N/2}$ and $\boldsymbol{y} = (y_i)_{0\le i<N/2}$, let $\mathsf{ct}.\boldsymbol{x}$ and $\mathsf{ct}.\boldsymbol{y}$ be the ciphertext encrypted by CKKS under the same public key. CKKS supports the following basic operations:

- $\mathsf{Add}(\mathsf{ct}.\boldsymbol{x}, \mathsf{ct}.\boldsymbol{y})$: $\mathsf{Dec}(\mathsf{Add}(\mathsf{ct}.\boldsymbol{x}, \mathsf{ct}.\boldsymbol{y})) \approx \boldsymbol{x} + \boldsymbol{y}$, where Dec is the decryption of CKKS.
- $\mathsf{Mul}(\mathsf{ct}.\boldsymbol{x}, \mathsf{ct}.\boldsymbol{y})$: $\mathsf{Dec}(\mathsf{Mul}(\mathsf{ct}.\boldsymbol{x}, \mathsf{ct}.\boldsymbol{y})) \approx \boldsymbol{x} \odot \boldsymbol{y}$, where $\odot$ is for component-wise multiplication.
- $\mathsf{CMul}(\boldsymbol{m}, \mathsf{ct}.\boldsymbol{x})$: $\mathsf{Dec}(\mathsf{CMul}(\boldsymbol{m}, \mathsf{ct}.\boldsymbol{x})) \approx \boldsymbol{m} \odot \boldsymbol{x}$, where $\boldsymbol{m}$ is a message in $\mathbb{C}^{\ell}$.
- $\mathsf{Rot}_k(\mathsf{ct}.\boldsymbol{x})$ converts $\mathsf{ct}.\boldsymbol{x} = \mathsf{Enc}(x_0, \ldots, x_{N/2-1})$ into a new ciphertext $\mathsf{Enc}(x_k, \ldots, x_{N/2-1}, x_0, \ldots, x_{k-1})$. Rot costs no multiplicative depths, but requires key-switching.
- $\mathsf{Bts}(\mathsf{ct}.\boldsymbol{x})$: The noise in CKKS ciphertexts increases with the computational depth, especially the multiplicative depth. Once the noise exceeds a certain threshold, correct decryption is no longer possible. Therefore, a bootstrapping operation $\mathsf{Bts}(\mathsf{ct}.\boldsymbol{x})$ must be performed to reduce the noise before this threshold is reached. It returns a ciphertext of the same plaintext $\boldsymbol{x}$, which is refreshed to support further operations.

CKKS is IND-CPA secure under the RLWE assumption [45]. In practical applications, how to minimize the required multiplicative depth (less Bts) and reduce the required Rot (less key-switching) are usually the central problems.

Appendix B
Workflow of BERT

Assume that the input data is $\boldsymbol{X}_0 \in \mathbb{R}^{m\times d}$ after embedding and positional encoding, where m is the *length* of tokens and d is the *hidden size* (or *model dimension*). Let L be the number of transformer layers and H be the number of heads. We now sketch the workflow of BERT [1].

A. Attention

Let $\boldsymbol{X}_i \in \mathbb{R}^{m\times d}$ be the input of the i-th transformer layer. Then the queries, keys, and values are computed as follows:

$$\begin{aligned}
\boldsymbol{Q}_i^{(h)} &= \boldsymbol{X}_i \boldsymbol{W}_Q^{(h)} + \mathbf{1}^\top \cdot \boldsymbol{b}_Q^{(h)} \in \mathbb{R}^{m\times d_h},\\
\boldsymbol{K}_i^{(h)} &= \boldsymbol{X}_i \boldsymbol{W}_K^{(h)} + \mathbf{1}^\top \cdot \boldsymbol{b}_K^{(h)} \in \mathbb{R}^{m\times d_h},\\
\boldsymbol{V}_i^{(h)} &= \boldsymbol{X}_i \boldsymbol{W}_V^{(h)} + \mathbf{1}^\top \cdot \boldsymbol{b}_V^{(h)} \in \mathbb{R}^{m\times d_h},
\end{aligned}$$

where $\boldsymbol{W}_Q^{(h)}, \boldsymbol{W}_K^{(h)}, \boldsymbol{W}_V^{(h)} \in \mathbb{R}^{d\times d_h}$ and $\boldsymbol{b}_Q^{(h)}, \boldsymbol{b}_K^{(h)}, \boldsymbol{b}_V^{(h)} \in \mathbb{R}^{d_h}$. Note that here $h \in [H]$ and $d_h = d/H$. Now the attention for the h-th head can be computed in parallel:

$$\boldsymbol{A}_i^{(h)} = \mathsf{softmax}\left(\frac{\boldsymbol{Q}_i^{(h)} \boldsymbol{K}_i^{(h)\top}}{\sqrt{d_h}}\right) \boldsymbol{V}_i^{(h)} \in \mathbb{R}^{m\times d_h}, \quad (5)$$

where $\mathsf{softmax}$ applies to each row of a matrix. For a vector $\boldsymbol{x} = (x_i)_i$, $\mathsf{softmax}$ is defined to be

$$\mathsf{softmax}(\boldsymbol{x}) = \left(\frac{\exp(x_i)}{\sum_i \exp(x_i)}\right)_i. \quad (6)$$

Then $(\boldsymbol{A}_i^{(h)})_h$ will be concatenated as a matrix

$$\boldsymbol{A}_i = \left(\boldsymbol{A}_i^{(0)}, \boldsymbol{A}_i^{(1)}, \ldots, \boldsymbol{A}_i^{(H-1)}\right) \in \mathbb{R}^{m\times d},$$

to which a linear transformation will be applied:

$$\boldsymbol{Y}_i = \boldsymbol{A}_i \boldsymbol{W}_0 + \mathbf{1}^\top \boldsymbol{b}_0 \in \mathbb{R}^{m\times d}$$

with $\boldsymbol{W}_0 \in \mathbb{R}^{d\times d}$ and $\boldsymbol{b}_0 \in \mathbb{R}^d$.

B. Feed-Forward Network

The input of the Feed-Forward Network (FFN) is the updated

$$\boldsymbol{Y}_i := \mathsf{LayerNorm}_1(\boldsymbol{X}_i + \boldsymbol{Y}_i) \in \mathbb{R}^{m\times d}$$

where $\boldsymbol{X}_i$ and $\boldsymbol{Y}_i$ are the input and output of the attention layer, respectively. Given a vector $\boldsymbol{x} = (x_i)_i$,

$$\mathsf{LayerNorm}(\boldsymbol{x}) = \left(\gamma \cdot \frac{x_i - \mu}{\sqrt{\sigma^2}} + \beta\right)_i \quad (7)$$

with μ and σ^2 the mean and variance of $\boldsymbol{x}$. The $\mathsf{LayerNorm}$ function applies to all the rows of the matrix. Now update $\boldsymbol{X}_i := \boldsymbol{Y}_i$ for later use and compute

$$\boldsymbol{Y}_i := \mathsf{GELU}(\boldsymbol{Y}_i \boldsymbol{W}_1 + \mathbf{1}^\top \boldsymbol{b}_1)\boldsymbol{W}_2 + \mathbf{1}^\top \boldsymbol{b}_2 \in \mathbb{R}^{m\times d}$$

with $\boldsymbol{W}_1 \in \mathbb{R}^{d\times d_f}$, $\boldsymbol{W}_2 \in \mathbb{R}^{d_f\times d}$, $\boldsymbol{b}_1 \in \mathbb{R}^{d_f}$, $\boldsymbol{b}_2 \in \mathbb{R}^d$,

$$\mathsf{GELU}(x) = \frac{1}{2}x \cdot \left(1 + \operatorname{erf}\left(\frac{x}{\sqrt{2}}\right)\right) \quad (8)$$

where $\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}}\int_0^x \mathrm{e}^{-t^2}dt$ is the Gaussian error function. Note that $d_f = 4d$. Then compute

$$\boldsymbol{X}_{i+1} = \mathsf{LayerNorm}_2(\boldsymbol{X}_i + \boldsymbol{Y}_i) \in \mathbb{R}^{m\times d}.$$

C. Pooling and Classification

The output of the L transformer layers is $\boldsymbol{X}_L$, whose first row $\boldsymbol{x}_1$ is the input of the classifier layer with a typical activation function, e.g., $\mathsf{Tanh}(x) = \frac{\mathrm{e}^x - \mathrm{e}^{-x}}{\mathrm{e}^x + \mathrm{e}^{-x}}$: $\boldsymbol{y} = \mathsf{Tanh}(\boldsymbol{x}_1 \boldsymbol{W}_3 + \boldsymbol{b}_3) \in \mathbb{R}^d$, where $\boldsymbol{W}_3 \in \mathbb{R}^{d\times d}$ and $\boldsymbol{b}_3 \in \mathbb{R}^d$. Here $\boldsymbol{y}$ is the *pooled output* of BERT. The final classification result is then given by $\boldsymbol{y} = \boldsymbol{y}\boldsymbol{W}_4 + \boldsymbol{b}_4 \in \mathbb{R}^c$, where $\boldsymbol{W}_4 \in \mathbb{R}^{d\times c}$ and $\boldsymbol{b}_4 \in \mathbb{R}^c$ with c the number of classes.

Appendix C
Missing Proofs

In this appendix, we provide the formal proofs for the propositions stated in Section V.

A. *Proof of Proposition 1*

Proof. Let $\boldsymbol{q}_s$ and $\boldsymbol{k}_s$ be the s-th column of $\boldsymbol{Q} \in \mathbb{R}^{m\times d_h}$ and $\boldsymbol{K} \in \mathbb{R}^{m\times d_h}$, respectively, and let $\boldsymbol{B} = \boldsymbol{Q}\boldsymbol{K}^\top \in \mathbb{R}^{m\times m}$. Then for $i' \in [m]$, the i'-th diagonal vector of $\boldsymbol{B}$ is

$$\boldsymbol{d}_{i'}(\boldsymbol{B}) = \sum_{s\in[d_h]} \boldsymbol{q}_s \odot \rho_{i'}(\boldsymbol{k}_s). \tag{9}$$

Assume that $m \le k\cdot\ell$ for some integers k and ℓ. Each index $i' \in [m]$ can be uniquely represented as $i' = i\cdot k + j$ for some $i \in [\ell]$ and $j \in [k]$. Hence, Eq. (9) can be rewritten as:

$$\boldsymbol{d}_{i\cdot k+j} = \sum_{s\in[d_h]} \boldsymbol{q}_s \odot \rho_{i\cdot k+j}(\boldsymbol{k}_s),$$

which leads to:

$$\rho_{-ik}(\boldsymbol{d}_{ik+j}(\boldsymbol{B})) = \sum_{s\in[d_h]} \rho_{-ik}(\boldsymbol{q}_s) \odot \rho_j(\boldsymbol{k}_s). \tag{10}$$

Algorithm 1 strictly follows Eq. (10) to evaluate $\mathsf{ct}.\rho_{-ik}(\boldsymbol{d}_{ik+j}(\boldsymbol{B}))$, ensuring correctness. Regarding complexity, ciphertext rotations occur only in Step 2 (baby steps, $k-1$ rotations per column) and Step 4 (giant steps, $\ell-1$ rotations per column). Thus, the total number of rotations is $(k+\ell-2)d_h$. □

B. *Proof of Proposition 2*

Proof. Algorithm 2 computes:

$$\begin{aligned}\boldsymbol{s} &= \sum_{i\in[\ell]} \rho_{ik}\Bigg(\sum_{j\in[k]} \rho_{-k}(\boldsymbol{d}_{ik+j}(\boldsymbol{B}))\Bigg)\\ &= \sum_{i\in[\ell]} \rho_{ik}\Bigg(\sum_{j\in[k]} \rho_{-k}(\boldsymbol{d}_{ik+j}(\boldsymbol{B})) \odot \boldsymbol{1}^\top\Bigg)\\ &= \sum_{i\in[m]} \boldsymbol{d}_i(\boldsymbol{B}) \odot \boldsymbol{1}^\top = \boldsymbol{B}\cdot\boldsymbol{1}^\top.\end{aligned}$$

This implies the i-th entry of $\boldsymbol{s}$ is the row sum $s_i = \sum_{j\in[m]} b_{i,j}$. Rotations only occur in Step 3, requiring at most $\ell-1$ rotations. □

C. *Proof of Proposition 3*

Proof. In Algorithm 3, $\mathsf{ct}.\boldsymbol{\mu}$ (Step 2) correctly stores the row-wise mean. The computation of $\boldsymbol{T}$ ensures that $\mathsf{ct}.\boldsymbol{\sigma}^2$ (Step 9) represents the row-wise variance. Steps 10-11 implement the statistical maximum estimation $\boldsymbol{g}$. Finally, the loops update $\boldsymbol{B}$ by subtracting $\boldsymbol{g}$. Rotations occur in Steps 1, 4, 8, and 13, each requiring $\le \ell$ rotations, totaling at most 4ℓ. □

D. *Proof of Proposition 5*

Proof. The plaintext counterpart of Algorithm 5 computes:

$$\begin{aligned}\tilde{\boldsymbol{a}}_s^\top &= \sum_{i\in[\ell]} \rho_{ik}\Bigg(\sum_{j\in[k]} \rho_{-ik}(\boldsymbol{d}_{ik+j}(\boldsymbol{C})) \odot \rho_j(\boldsymbol{v}_s)\Bigg)\\ &= \sum_{i\in[\ell]}\sum_{j\in[k]} \big(\boldsymbol{d}_{ik+j}(\boldsymbol{C}) \odot \rho_{ik+j}(\boldsymbol{v}_s)\big)\\ &= \sum_{i\in[m]} \boldsymbol{d}_i(\boldsymbol{C}) \odot \rho_i(\boldsymbol{v}_s) = \boldsymbol{C}\boldsymbol{v}_s^\top.\end{aligned}$$

The final equality holds based on the Halevi-Shoup algorithm [47]. Since the rotated diagonals of $\boldsymbol{C}$ are precomputed inputs, the algorithm requires only $k+\ell-2$ rotations for each column $s \in [d_h]$. □

E. *Proof of Proposition 6*

Proof. Based on Proposition 2, the summation vector $\boldsymbol{s}$ is correctly computed. Algorithm 6 then performs the division and element-wise multiplication as required by Eq. (2). Rotations are only needed for the summation step, requiring at most $\ell-1$ operations. □

Appendix D
Implementation Details of Non-Linear Operations

In the computation of the attention mechanism, the softmax operation involves max, exp, and $1/x$. Here, we detail the implementation of non-linear functions required in subsequent operations: $1/\sqrt{x}$ in LayerNorm, GELU in the Feed-Forward Network (FFN), and Tanh in the Pooler.

A. *Square Root Reciprocal*

The definition of Layer Normalization is given in Eq. (7). The primary challenge lies in evaluating the $1/\sqrt{x}$ function homomorphically. Our approach is to first approximate $1/\sqrt{x}$ using a Chebyshev polynomial to obtain a suitable initial value, and then apply Newton iteration to improve the accuracy. Since each Transformer layer contains two LayerNorm operations, we employ two distinct Chebyshev polynomials for the approximations; see Table X.

1) GELU *and* Tanh*:* The definition of GELU is provided in Equation (8). In practice, it can also be approximated by the following formula,

$$\mathsf{GELU}(x) \quad = \quad 0.5x(1+\mathsf{Tanh}(\sqrt{2/\pi}(x+0.044715x^3))),$$

which reduces the problem to approximating only the Tanh function. In our work, we adopt the standard definition of GELU; however, since the pooling layer also includes a Tanh operation, we address both the GELU and Tanh functions in our implementation. Specifically, we directly use Chebyshev polynomial approximations for both functions, following the approach in [14], [15]; see Table X for parameters we use.

TABLE X: Approximation parameters for $1/\sqrt{x}$, GELU and Tanh.

Function	Placement	Deg Cheby.	#Newton Itr.
$1/\sqrt{x}$	LayerNorm-1	31	2
	LayerNorm-2	31	2
$\mathsf{GELU}(x)$	FFN	255	–
$\mathsf{Tanh}(x)$	Classification	63	–

Appendix E
Additional Features of Our Implementation

In addition to the core algorithmic contributions, ARION integrates several features to further reduce computational overhead and enhance system flexibility.

A. Optimization on Multiplication

We strive to minimize the required multiplicative depth and computational cost by leveraging mathematically equivalent formulations. For example, when computing $\boldsymbol{Q}\boldsymbol{K}^\top/\sqrt{d_h}$, we fold the constant $1/\sqrt{d_h}$ into the plaintext weight matrix $\boldsymbol{W}_Q$ (or $\boldsymbol{W}_K$) during the encoding phase. This eliminates one level of ciphertext-plaintext multiplication (CMul). Additionally, the expansion of $\boldsymbol{Q}\boldsymbol{K}^\top$ yields a term $\boldsymbol{X}(\boldsymbol{W}_Q\boldsymbol{W}_K^\top)\boldsymbol{X}^\top$. This structure allows for pre-multiplying the two plaintext matrices $\boldsymbol{W}_Q$ and $\boldsymbol{W}_K^\top$ before performing the homomorphic computation, effectively reducing the operation count by one full matrix multiplication.

B. Acceleration with Hoisting

The Double-BSGS algorithm (specifically Steps 2 and 4 in Algorithm 1) requires performing multiple ciphertext rotations on the same input ciphertext but with different rotation indices. This pattern is ideal for the Hoisting technique introduced by Halevi and Shoup [53]. By pre-computing and reusing the modulus switching and key-switching decomposition parts of the ciphertext, Hoisting significantly reduces the amortized cost of rotations.

C. Flexibility of Token Length

ARION allows users to configure the maximum token length for inference. Specifically, with a fixed CKKS ring dimension N, a single execution can perform batched encrypted inference for up to $2^{N/2-\ell}$ sequences, each with a maximum token length of 2^ℓ. This flexibility enables the protocol to achieve higher throughput (amortized performance) when processing shorter sequences, rather than being locked into a fixed worst-case configuration.

D. Task-Level Parallelization

We optimized the normalization step (Algorithm 6) using a task-level parallelization strategy. The reciprocal computation involves a complex chain of operations but depends on only a single ciphertext (the sum vector). Conversely, the multiplication of the exp result with the value matrix $\boldsymbol{V}$ involves simpler operations but on a larger volume of data. In our implementation, a dedicated thread handles the reciprocal computation, while the remaining threads process the $\boldsymbol{V}$ multiplication in parallel. This design allows for efficient overlapping of heavy computations and minimizes thread idle time.

Appendix F
Formal Security Proof

We rigorously analyze the security of ARION under the semi-honest model. During the execution of the protocol, the views of the Client (P_1) and the Server (P_2) are defined. The view structure is formalized as a tuple: (Private Inputs; Received Messages; Sent Messages).

$$View_C = (pk, sk, \boldsymbol{X}; \emptyset; \mathsf{ct}.\boldsymbol{X})$$
$$View_S = (pk, evk, \mathcal{W}; \mathsf{ct}.\boldsymbol{X}; \mathsf{ct}.\boldsymbol{Y})$$

Here, $\boldsymbol{X}$ represents the private input sequences from the Client, $\mathcal{W}$ denotes the private model parameters of the Server, $\mathsf{ct}.\boldsymbol{X}$ is the encrypted input sent to the Server, and $\mathsf{ct}.\boldsymbol{Y}$ is the encrypted inference result returned to the Client. The symbol $\emptyset$ denotes an empty set, indicating that the Client receives no intermediate messages during the computation.

Simulation of Client's View

The client view $View_C$ consists of its own inputs $(pk, sk, \boldsymbol{X})$ and the result message $\mathsf{ct}.\boldsymbol{Y}$. In the semi-honest model, the client is entitled to receive the computational result $f(\boldsymbol{X}, \mathcal{W})$. Since the server performs deterministic homomorphic evaluations, the output ciphertext $\mathsf{ct}.\boldsymbol{Y}$ is a function of the inputs. A simulator S_1, given the output $y = f(\boldsymbol{X}, \mathcal{W})$ and the public key, can generate a valid encryption $\mathsf{ct}.\boldsymbol{Y}'$ that is computationally indistinguishable from the real execution's output within the context of the protocol flow. Thus, $View_C$ can be simulated by S_1, implying $View_C \equiv S_1$.

Simulation of Server's View

The server view $View_S$ contains the public key pk, evaluation keys evk, the private model parameters $\mathcal{W}$, and the received ciphertext $\mathsf{ct}.\boldsymbol{X}$. The core security requirement is that the server learns nothing about $\boldsymbol{X}$.

We construct a simulator S_2 to simulate the server's view. S_2 does not have access to the real input $\boldsymbol{X}$. Instead, it generates a dummy input $\boldsymbol{X}'$ (e.g., a zero matrix of the same dimension as $\boldsymbol{X}$) and uses the public key pk to generate a simulated ciphertext $\mathsf{ct}.\boldsymbol{X}' = \mathsf{Enc}_{pk}(\boldsymbol{X}')$. Therefore, the simulated view is:

$$S_2 = (pk, evk, \mathcal{W}; \mathsf{ct}.\boldsymbol{X}'; \mathsf{ct}.\boldsymbol{Y}').$$

Based on the semantic security (IND-CPA) of the underlying CKKS encryption scheme and the hardness of the RLWE problem, an encryption of the real input $\mathsf{ct}.\boldsymbol{X}$ is computationally indistinguishable from an encryption of the dummy input $\mathsf{ct}.\boldsymbol{X}'$. That is, for any probabilistic polynomial-time adversary $\mathcal{A}$:

$$\{View_S\} \overset{c}{\equiv} \{S_2\}$$

Since the server's view in the real execution is indistinguishable from the simulated view where the input is replaced by garbage data, the server learns no information about the client's private input $\boldsymbol{X}$.

In summary, ARION is secure in the semi-honest model.

Appendix G
Analysis of Algorithmic Speedups vs. FHE Library Differences

A critical question in evaluating ARION is distinguishing the performance gains attributable to our algorithmic contributions (e.g., Double-BSGS, linear-nonlinear fusion) from those resulting from the underlying FHE library. Specifically, ARION is implemented using Lattigo [22], whereas the baseline protocol MOAI [8] utilizes Microsoft SEAL [17]. To ensure a fair comparison, we conduct a micro-benchmark analysis and a normalized runtime projection.

A. Micro-Benchmarks: Lattigo vs. SEAL

We benchmarked the primitive operations of the CKKS scheme in both libraries under identical cryptographic parameters (as detailed in §VIII-A). The results, averaged over 100 runs (5 runs for Bootstrapping), are presented in Table XI.

TABLE XI: Micro-benchmark of Basic Operations (ms). Comparison between SEAL and Lattigo under single-threaded execution.

Library	**CMul**	**Mul**	**Rot**	**Bts**
SEAL [17]	11	116	103	49,152
Lattigo [22]	34	212	186	27,184
Ratio (Lattigo/SEAL)	**3.09**$\times$	**1.83**$\times$	**1.81**$\times$	**0.55**$\times$

As shown in Table XI, Lattigo exhibits a distinct performance profile compared to SEAL:

- **Bootstrapping (**Bts**):** Lattigo is approximately $1.8\times$ faster than SEAL.
- **Basic Operations:** Conversely, for non-bootstrapping operations (CMul, Mul, Rot), Lattigo is significantly slower, ranging from $1.8\times$ to $3.1\times$ slower than SEAL.

B. Normalized Runtime Projection

The mixed performance profile suggests that Lattigo does not provide a universal speedup. To quantify the impact of the library choice on the end-to-end runtime of MOAI (BERT-Base), we decompose MOAI's reported runtime (574.6s) into bootstrapping and non-bootstrapping components based on the data in [8]:

$$T_{\text{MOAI}}^{\text{SEAL}} \approx T_{\text{Bts}} + T_{\text{Ops}} = 384.8\text{s} + 189.8\text{s} = 574.6\text{s}. \quad (11)$$

We then project the hypothetical runtime of MOAI if it were implemented in Lattigo ($T_{\text{MOAI}}^{\text{Lattigo}}$). We apply the speedup factor of $1.8\times$ for the bootstrapping component and a conservative average slowdown factor of $2.45\times$ (average of basic operations ratios) for the non-bootstrapping component:

$$\begin{aligned} T_{\text{MOAI}}^{\text{Lattigo}} &\approx \frac{384.8}{1.8} + (189.8 \times 2.45) \\ &\approx 213.8\text{s} + 465.0\text{s} = 678.8\text{s}. \end{aligned} \quad (12)$$

C. Discussion

The projection above yields several key insights:

1) **Algorithmic Dominance:** The projected runtime of MOAI on Lattigo (678.8s) is approximately $3\times$ slower than ARION's actual runtime (225s). This indicates that even if MOAI utilized the faster bootstrapping of Lattigo, its heavy reliance on basic matrix operations (which are slower in Lattigo) would hinder its performance. ARION's speedup is thus predominantly driven by our Double-BSGS optimization and Linear-Nonlinear fusion, which drastically reduce the count of expensive operations.
2) **Library Trade-offs:** A break-even analysis suggests that for MOAI's performance in Lattigo to match its performance in SEAL, the non-bootstrapping operations in SEAL would need to be only $1.9\times$ faster than Lattigo (derived from $384.8/1.8 + 189.8 \times 1.9 \approx 574.4$). However, since SEAL is actually $1.8\times \sim 3.1\times$ faster for these operations, MOAI actually benefits significantly from SEAL's efficient basic arithmetic.
3) **Resource Efficiency:** It is also noteworthy that ARION achieves these results using only 64 threads, whereas MOAI's reported results utilize 112 threads. ARION outperforms the baseline despite using approximately 43% fewer computational resources and a library that is slower for basic arithmetic operations.
4) **Depth Optimization:** Beyond operator-level improvements, we have optimized the global computational graph of the BERT model. Specifically, while MOAI requires a maximum multiplicative depth of 15 levels to support its inference pipeline, ARION streamlines the data flow to reduce this requirement to only 14 levels. This reduction in circuit depth enables the selection of more favorable homomorphic encryption parameters (i.e., a smaller total ciphertext modulus q), which inherently accelerates all underlying arithmetic operations and reduces memory consumption.

In conclusion, the performance advantage of ARION is robust and stems from algorithmic efficiency rather than library discrepancies or hardware parallelism.

Appendix H
Detailed Accuracy Examples

To further illustrate the precision of ARION, Table XII presents a side-by-side comparison of logits generated by plaintext inference and ARION's encrypted inference for specific samples from the QNLI, RTE, and SST-2 datasets. The minimal deviation observed confirms that our approximation strategies do not compromise downstream classification accuracy.

TABLE XII: Comparison of plaintext and Arion-encrypted inference logits on BERT-Tiny across datasets

Dataset & Input Sentence	Type	Logits
QNLI: What is the capital of France? Paris is the capital and most populous city of France.	Plaintext	$[0.9606,\ -0.8399]$
	ARION	$[0.9584,\ -0.8396]$
RTE: The dog is running through the park. An animal is moving outdoors.	Plaintext	$[-0.0425,\ 0.1623]$
	ARION	$[-0.0426,\ 0.1623]$
SST-2: A visually stunning rumination on love.	Plaintext	$[-1.8836,\ 1.8262]$
	ARION	$[-1.8919,\ 1.8366]$

Logits are reported as $[\text{class}_1,\ \text{class}_2]$. The encrypted inference yields predictions identical to the plaintext baseline.