Expert Systems With Applications 334 (2027) 134756



Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa



ARION: Attention-optimized transformer inference on encrypted data

Linhan Yang [a], Jingwei Chen [b,c,*], Wangchen Dai [a], Shuai Wang [d], Wenyuan Wu [b,c], Yong Feng [b,c]

[a] School of Cyber Science and Technology, Sun Yat-Sen University, Shenzhen, 518107, China
[b] Chongqing Key Lab of Secure Computing for Biology, Chongqing Institute of Green and Intelligent Technology, Chinese Academy of Sciences, Chongqing, 400714, China
[c] Chongqing School, University of Chinese Academy of Sciences, Chongqing, 400714, China
[d] School of Information Science and Engineering, Chongqing Jiaotong University, Chongqing, 400074, China

ARTICLE INFO

Keywords:
Privacy-preserving computation
Fully homomorphic encryption
Transformer
Secure inference

ABSTRACT

Privacy-preserving Transformer inference (PPTI) is essential for deploying Transformer models in sensitive application domains, where inference must protect user data while remaining practically usable. In FHE-based PPTI, encrypted attention remains a major bottleneck due to costly ciphertext rotations and the difficulty of integrating softmax with matrix operations under compatible packing formats. In this paper, we present ARION, a non-interactive FHE-based PPTI protocol that protects client-input confidentiality against a semi-honest server and targets this bottleneck through algorithmic optimization of encrypted attention. First, for the three consecutive ciphertext matrix multiplications in multi-head attention, we introduce the double Baby-Step Giant-Step (Double-BSGS) algorithm, which substantially reduces the number of ciphertext rotations. On BERT-Base, ARION achieves an 82.5% reduction in rotations over the state-of-the-art CPU-based baseline MOAI (ICLR 2026), corresponding to a 5.7× reduction in rotation count. Second, we propose a linear–nonlinear fusion technique for softmax evaluation in attention. By decomposing softmax into shift-by-maximum, exponentiation, and reciprocal sub-steps and integrating them with the surrounding encrypted matrix operations, ARION enables efficient attention evaluation while remaining compatible with practical ciphertext packing formats. We implement ARION using Lattigo and evaluate attention kernels on representative Transformer models. Controlled experiments using the same code base and hardware show that Double-BSGS speeds up the attention matrix operations by 1.76–1.89× across BERT-Tiny, BERT-Base, and LLaMA-3-8B. In a system-level comparison on the same workstation, which also reflects differences in implementation and configuration, ARION achieves an amortized latency of 224.91 s on BERT-Base, compared with 622.35 s for our reproduction of MOAI, corresponding to a 2.77× speedup. Evaluation on SST-2, QNLI, and RTE shows that encrypted inference maintains accuracy close to plaintext inference.

1. Introduction

Large language models (LLMs) and other Transformer-based models are increasingly being deployed in sensitive domains such as healthcare, finance, and public administration. In these settings, inference often involves private or regulated data, making privacy-preserving deployment a practical requirement rather than a purely theoretical goal. This demand has driven growing interest in privacy-preserving machine learning (PPML), especially in secure inference scenarios where the model service must protect user inputs without sacrificing utility.

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 (Devlin et al., 2019), a state-of-the-art MPC-based method, BLB (Xu et al., 2025), incurs approximately 16 GB of communication per encrypted inference.

FHE (Gentry, 2009), 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 (Zhang et al., 2025) performs encrypted inference on BERT-Base with only 0.16 GB of communication. Nevertheless, applying FHE to real-world deep learning inference—especially

* Corresponding author.
E-mail address: chenjingwei@cigit.ac.cn (J. Chen).

https://doi.org/10.1016/j.eswa.2026.134756
Received 7 June 2026; Received in revised form 5 October 2026; Accepted 14 October 2026
Available online 17 October 2026
0957-4174/© 2026 Elsevier Ltd. All rights are reserved, including those for text and data mining, AI training, and similar technologies.

for models like Transformers (Vaswani et al., 2017) 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) (Moon et al., 2025) 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 (Park et al., 2025) (ACL'25) still incurs a per-query latency of about 5.74 minutes on a single GPU. A very recent protocol, MOAI (Zhang et al., 2026), 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.

1.1. Challenges of FHE in non-interactive PPTI

For non-interactive FHE-based PPTI, one of the main bottlenecks lies in encrypted attention. Transformer models such as GPT (Radford et al., 2018), BERT (Devlin et al., 2019), and LLaMA (Grattafiori et al., 2024) involve extensive matrix computations of different sizes and structures. In BERT-Base inference, some matrices have dimensions as large as 3072, and matrix multiplication dominates the cost even in plaintext execution. Under FHE, this cost becomes substantially more severe. For example, in NEXUS (Zhang et al., 2025), encrypted matrix computations account for a large fraction of the total runtime for BERT-Base. A central difficulty is that encrypted attention involves multiple types of matrix operations, including ciphertext-plaintext matrix multiplication (CPMM), ciphertext-ciphertext matrix multiplication (CCMM), and ciphertext matrix transposition (CMT), often under different ciphertext packing formats. As a result, the system must jointly optimize arithmetic cost and packing compatibility. This challenge becomes especially pronounced in the three consecutive matrix multiplications inside multi-head attention, where reducing ciphertext rotations is critical for improving efficiency.

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.

1.2. Contributions

We propose Attention-optimized tRansformer Inference On eNcrypted data (ARION), an FHE-based non-interactive PPTI protocol for efficient encrypted Transformer inference. ARION is motivated by the observation that Transformer inference involves a tightly coupled computation flow of encrypted matrix multiplications, ciphertext packing formats, and nonlinear function evaluation. Rather than treating these components as isolated operations, ARION redesigns the core encrypted computation flow so that linear operations, packing formats, and nonlinear functions can be optimized jointly. Our contributions include:

- We propose *double Baby-Step Giant-Step* (*Double-BSGS*), an optimized algorithm for rotation-efficient encrypted attention. Ciphertext rotations are one of the dominant costs in FHE-based Transformer inference. By exploiting the structured matrix-product flow in attention, Double-BSGS reduces the dominant rotation complexity from $O(md_hH)$ to $O(\sqrt{md_h}H)$, where m is the input sequence length, H is the number of attention heads, and d_h is the dimension of each head. For BERT-Base, ARION achieves an approximately 82.5% reduction in ciphertext rotations, corresponding to a 5.7× smaller rotation count, compared with the state-of-the-art CPU-based baseline MOAI (Zhang et al., 2026) (Table 3). Furthermore, we evaluate Double-BSGS on DASHformer (Harmanci et al., 2024), showing that the proposed optimization is applicable beyond standard BERT models (Table 10).
- We propose a *linear–nonlinear fusion* technique for packing-compatible encrypted softmax evaluation. Softmax is difficult to evaluate under FHE because it is a multivariate nonlinear function whose normalization term depends on an entire row of encrypted values; directly approximating the whole function with a polynomial often leads to poor accuracy. To address this issue, we decompose softmax into three tractable sub-steps—shift by max, exponentiation ($\exp$), and reciprocal ($1/x$)—and integrate them into different stages of the encrypted attention pipeline. This design preserves ciphertext packing while evaluating softmax within the optimized matrix-product flow and reduces normalization multiplications from 128 to 64 by deferring normalization until after CV.
- We implement ARION with Lattigo and evaluate it on BERT-Tiny and BERT-Base. The amortized inference latency is 9.30 s for BERT-Tiny and 224.91 s for BERT-Base. [1] On the same workstation, our reproduction of MOAI requires 622.35 s for BERT-Base, giving the complete ARION system a 2.77× end-to-end speedup. Because ARION and MOAI use different HE libraries, nonlinear approximations, and evaluation pipelines, this result should be interpreted as a system-level comparison rather than as a direct measure of Double-BSGS. Under the controlled setting in Table 6, Double-BSGS speeds up the attention matrix operations by 1.76–1.89×.

The proposed optimizations target Transformer attention modules that compute $\mathsf{softmax}(\boldsymbol{QK}^\top/\sqrt{d_h})\boldsymbol{V}$. Attention variants with different computation patterns may require additional adaptation.

1.3. Limitations

ARION is currently evaluated only on CPUs, and we therefore do not claim the fastest end-to-end wall-clock performance across all hardware platforms. In addition, our end-to-end evaluation focuses on BERT-Tiny and BERT-Base, without full-system validation on larger decoder-only models. Moreover, autoregressive decoding with a growing KV cache is not considered in this work, as our evaluation focuses on the attention computation in the prefill setting. These limitations do not affect the core contribution of this work, since ARION focuses on algorithmic optimization for encrypted attention and reduces the dominant rotation complexity from $O(md_hH)$ to $O(\sqrt{md_h}H)$. Combining ARION with GPU acceleration and extending it to larger end-to-end encrypted Transformer inference remain important directions for future work.

2. 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 Ng and

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

Table 1
Comparison of FHE-based non-interactive PPTI protocols for BERT (time in seconds).

Protocols	Batch	Retrain.	Flex.	Models	Platform	HE Lib.	Tok.	Runtime per Seq.
PolyTransformer (Zimerman et al., 2024b)	✗	✓	✗	BERT-like†	CPU (64) + GPU	HeLayers (Aharoni et al., 2023)	128	211
FHE-BERT-Tiny (Rovida & Leporati, 2024)	✗	✗	✗	BERT-Tiny	CPU (8)	OpenFHE	40	≈ 600
Tricycle (Lim et al., 2025)*	✗	✗	✗	BERT-Tiny	CPU (128)	OpenFHE (Al Badawi et al., 2022)	44	322
NEXUS (Zhang et al., 2025)	✓	✗	✗	BERT-Base	CPU (32)	SEAL (Lee et al., 2022; Microsoft, Accessed in July, 2023)	128	857
					GPU	Phantom (Yang et al., 2024)	128	37
THOR (Moon et al., 2025)	✗	✗	✗	BERT-Base	GPU	Liberate.FHE (DESILO, 2023)	128	626
PowerFormer (Park et al., 2025)	✗	✓	✗	BERT-Base	GPU	Liberate.FHE (DESILO, 2023)	128	344
MOAI (Zhang et al., 2026)	✓	✗	✓	BERT-Base	CPU (112)	SEAL (Lee et al., 2022; Microsoft, Accessed in July, 2023)	128	574
					GPU	Phantom (Yang et al., 2024)	128	141
Euston (Gao et al., 2026)	✓	✗	✓	BERT-Base	CPU	SEAL (Lee et al., 2022; Microsoft, Accessed in July, 2023)	128	648
					GPU	Phantom (Yang et al., 2024)	128	46
ARION (This work)	✓	✗	✓	BERT-Tiny	CPU (64)	Lattigo (Tune-Insight, 2025)	128	9.3
				BERT-Base			128	225

Runtime is reported per sequence. For methods supporting batch inference, we report the amortized runtime over the corresponding batch; otherwise, we report the runtime of a single sequence execution. 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 sequences of arbitrary token lengths. CPU (n) indicates that the number of threads used n. ∗ Those are preprints by the submission date of this paper (August 2026). † This is a retrained BERT-based model with 6 Transformer layers and 53.3 million parameters.

Chow (2023), Zeng et al. (2025) and references therein for further reading. Broadly speaking, existing research on privacy-preserving Transformer inference (PPTI) can be classified into MPC-based (Li et al., 2023; Luo et al., 2024; Zeng et al., 2023), hybrid MPC-HE (Cheng et al., 2025; Hao et al., 2022; Kei & Chow, 2025; Xu et al., 2025), and FHE-based approaches (Chen et al., 2024, 2022; Gao et al., 2026; Moon et al., 2025; Rovida & Leporati, 2024; Wang et al., 2026; Zhang et al., 2025, 2026; Zimerman et al., 2024a,b). 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 (Li et al., 2025). Here, we mainly focus on FHE-based PPTI protocols.

2.1. FHE-based PPTI protocols

Table 1 summarizes recent FHE-based non-interactive PPTI protocols for BERT (Devlin et al., 2019). Note that THE-X (Chen et al., 2022) also relies only on HE, but requires interaction to evaluate nonlinear functions such as softmax and GELU. For reference, Table 1 includes both CPU and GPU implementations. Euston (Gao et al., 2026) reports 46 s on GPU, while ARION achieves 225 s on CPU. We therefore compare ARION only with CPU-based baselines.

Both PolyTransformer (Zimerman et al., 2024b) and PowerSoftmax (Zimerman et al., 2024a) follow the approach of replacing activation functions with polynomial 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 (Aharoni et al., 2023). A recent result in this line is PowerFormer (Park et al., 2025), 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 (Rovida & Leporati, 2024) and Tricycle (Lim et al., 2025) 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 (Chen et al., 2025) to tricycle encoding to accelerate matrix computations. NEXUS (Zhang et al., 2025), THOR (Moon et al., 2025), and MOAI (Zhang et al., 2026) 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 (Lim et al., 2025; Moon et al., 2025; Park et al., 2025; Rovida & Leporati, 2024) 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 (Rovida & Leporati, 2024). Protocols that support batch inference can process multiple sequences in a single execution (Chen et al., 2024; Gao et al., 2026; Zhang et al., 2025, 2026). 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, representative non-batch and batch protocols such as THOR (GPU) (Moon et al., 2025) and MOAI (CPU) (Zhang et al., 2026) require approximately 10 minutes to complete inference on a single sequence.

In summary, as shown in Table 1, 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 2.77x speedups on CPU for encrypted inference on BERT-Tiny and BERT-Base, respectively.

2.2. What's new of ARION

From the perspective of attention computation, the protocol most closely related to ARION is MOAI (Zhang et al., 2026). Both protocols adopt related data packing strategies to facilitate batch processing, but ARION differs from MOAI by introducing the Double-BSGS algorithm, which optimizes the structured matrix-product flow in encrypted attention. By integrating Double-BSGS with our linear–nonlinear fusion technique, ARION improves both ciphertext rotation efficiency and packing compatibility. This design enables the softmax sub-steps, including maximum extraction, exponentiation, and reciprocal, to be executed 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 with 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

Table 2
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

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-source implementation of ARION is the first end-to-end and non-interactive PPTI protocol implemented with one of the most popular FHE libraries Lattigo (Tune-Insight, 2025).

3. 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 $\{0, 1, \ldots, z-1\}$.

3.1. Fully homomorphic encryption: The CKKS scheme

We employ the CKKS scheme (Cheon et al., 2017) 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 (Lyubashevsky et al., 2013). For a detailed description of these primitives and parameters, please refer to Appendix B.

3.2. The BERT model

In this paper, we focus on the inference of Transformer-based large language models, specifically BERT (Devlin et al., 2019). 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). For each attention head, $\boldsymbol{Q}$, $\boldsymbol{K}$, and $\boldsymbol{V}$ denote the query, key, and value matrices, respectively. 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 C. The specific hyperparameters for the BERT variants evaluated in this work (BERT-Tiny and BERT-Base) are listed in Table 2.

3.3. 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 among the most intricate components in FHE-based Transformer inference. To support efficient encrypted attention, ARION adopts a packing strategy based on column packing and diagonal packing, avoiding frequent conversions between different ciphertext packing formats. This strategy is closely related to the packing design used in MOAI (Zhang et al., 2026), which has been shown to reduce ciphertext rotations compared with BOLT (Pang et al., 2024), NEXUS (Zhang et al., 2025), PowerFormer (Park et al., 2025), and THOR (Moon et al., 2025). Building on this packing direction, ARION further optimizes the structured matrix-product flow in attention through Double-BSGS and linear–nonlinear fusion. We therefore first provide a brief overview of the underlying packing approach.

3.3.1. 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 ith 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 ith (upper) diagonal of $\boldsymbol{X}$.

3.3.2. 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 $\mathsf{ct}.(\boldsymbol{XW})_{\text{col}} = \{\mathsf{ct}.(\boldsymbol{XW})_j : 0 \leq j < d_h\}$ with

$$\mathsf{ct}.(\boldsymbol{XW})_j = \mathsf{Add}_{i \in [d]}(\mathsf{CMul}(w_{i,j}, \mathsf{ct}.\boldsymbol{x}_i^\top)). \tag{1}$$

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

3.3.3. 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 both 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 $\boldsymbol{k}_i^\top$ are the ith 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 $\boldsymbol{QK}^\top$.

3.3.4. 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. (C.1) , 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 jth column of $\boldsymbol{CV}$ can be computed through

$$\mathsf{ct}.(\boldsymbol{CV})_j = \mathsf{Add}_{i \in [m]}(\mathsf{Mul}(\mathsf{ct}.\boldsymbol{d}_i(\boldsymbol{C}), \mathsf{Rot}_i(\mathsf{ct}.\boldsymbol{v}_j^\top))), \tag{2}$$

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

4. System overview

4.1. 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:

- 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.

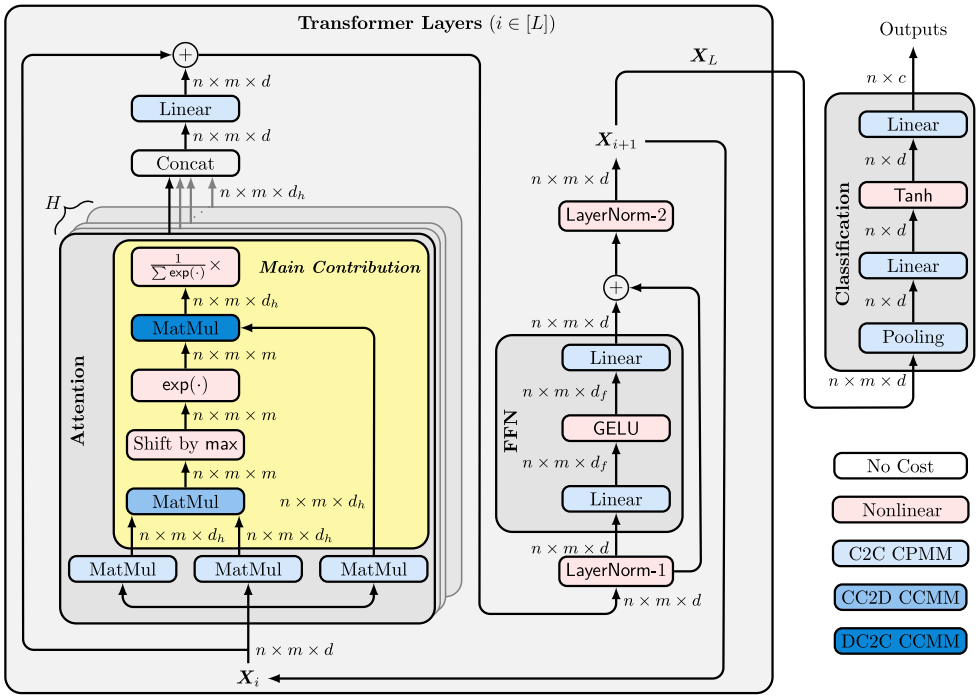


Fig. 1. The system architecture of ARION. The hyperparameters are given in Table 2. C2C CPMM is for ciphertext-plaintext matrix multiplication, where the input and output ciphertext matrices are all in column packing. CC2D CCMM is for ciphertext-ciphertext matrix multiplication, where the two input matrices 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 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 Section 4.1) 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 (Chen et al., 2024; Moon et al., 2025; Pang et al., 2024; Park et al., 2025; Zhang et al., 2025, 2026). Although these protocols do not provide security against malicious adversaries, the client's input privacy against a semi-honest server can still be preserved. Our formal analysis is limited to this guarantee; circuit privacy and model privacy against the client, which require additional analysis or mechanisms such as noise flooding, are outside the scope of this work.

4.2. 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 jth ciphertext stores the jth $n \times m$ 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 (Jiang et al., 2018, § 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 Section 5. The proposed attention optimization applies to attention modules that $\text{computesoftmax}(\boldsymbol{QK}^\top/\sqrt{d_h})\boldsymbol{V}$.

Attention variants with different computation patterns require additional adaptation. For causal masking, plaintext masks need to be applied to the corresponding rotated diagonals before exponentiation and row-wise summation. For grouped-query attention (GQA), the sharing of key/value heads requires changes to head mapping and rotation scheduling. These extensions are not implemented or evaluated end-to-end in this work.

5. Optimized attention on encrypted data

The computation in Eq. (C.1) essentially evaluates the expression $\boldsymbol{A} = \mathsf{softmax}(\boldsymbol{QK}^\top)\boldsymbol{V}$, [2] where $\boldsymbol{Q}$, $\boldsymbol{K}$, and $\boldsymbol{V} \in \mathbb{R}^{m \times d_h}$ 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 $\boldsymbol{B} = \boldsymbol{QK}^\top \in \mathbb{R}^{m \times m}$,
2. compute $\boldsymbol{C} = \mathsf{softmax}(\boldsymbol{B}) \in \mathbb{R}^{m \times m}$, and
3. compute $\boldsymbol{A} = \boldsymbol{CV} \in \mathbb{R}^{m \times d_h}$.

[2] For simplicity, we omit the constant factor $1/\sqrt{d_h}$ inside the 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.

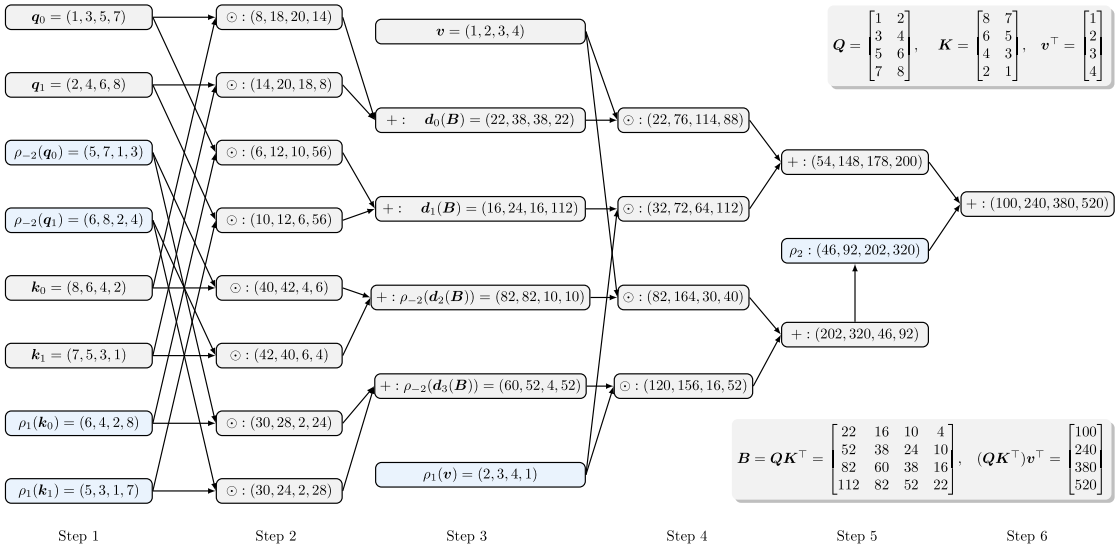


Fig. 2. An illustrative example for $(\boldsymbol{Q}\boldsymbol{K}^\top)\boldsymbol{v}$. $\rho_j(\boldsymbol{v})$ denotes a left rotation of the vector $\boldsymbol{v}$ by j positions. In this example, both $\boldsymbol{Q}$ and $\boldsymbol{K}$ are 4×2 matrices, and $\boldsymbol{v}$ is a four-dimensional row vector, i.e., $m = 4$ and $d_h = 2$. Given column-packed ciphertexts of $\boldsymbol{Q}$ and $\boldsymbol{K}$, we first rotate these ciphertexts to with appropriate step sizes (Step 1), then construct the rotated diagonal vectors of the matrix $\boldsymbol{B} = \boldsymbol{Q}\boldsymbol{K}^\top$ (Step 3) that are necessary for the baby-step giant-step (BSGS) algorithm, and finally apply BSGS to compute the result of $(\boldsymbol{Q}\boldsymbol{K}^\top)\boldsymbol{v}$ (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 Section 5.6 for detailed analysis.

To compute $\boldsymbol{A} = \boldsymbol{C}\boldsymbol{V}$, one can proceed column by column. Specifically, the jth 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 jth column of matrix $\boldsymbol{V}$. Assume that the matrix $\boldsymbol{C}$ remains in diagonal packing. Following the method proposed by Halevi and Shoup (Halevi & Shoup, 2014) (see Eq. (2)), computing $\boldsymbol{a}_i^\top$ requires m ciphertext rotations. However, Halevi and Shoup also introduced a baby-step giant-step (BSGS) algorithm (Halevi & Shoup, 2015) 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 $m = \ell \cdot k$)

$$\boldsymbol{C}\boldsymbol{v}^\top = \sum_{0 \le i < \ell} \rho_{ik}\Big(\sum_{0 \le j < k} \rho_{-ik}\big(\boldsymbol{d}_{ik+j}(\boldsymbol{C})\big) \odot \rho_j(\boldsymbol{v}) \Big), \tag{3}$$

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 § 3.3.3), while the $\mathsf{softmax}$ function is applied row by row (see § C.1 , which gives rise to the following challenges:

- Assuming that the $\mathsf{softmax}$ 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 $\mathsf{softmax}$ 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 (Halevi & Shoup, 2014) 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 ith row of $\boldsymbol{B}$.
3. Compute $\boldsymbol{C} = \exp(\boldsymbol{B})$: 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 ith component is given by $s_i = 1/\sum_j c_{i,j}$, with $\boldsymbol{C} = (c_{i,j}) = \exp(\boldsymbol{B})$.

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

$$\boldsymbol{a}_i = s_i \cdot \boldsymbol{c}_i \cdot \boldsymbol{V} = \frac{1}{\sum_j c_{i,j}} \boldsymbol{c}_i \cdot \boldsymbol{V} = \mathsf{softmax}((\boldsymbol{b})_i) \cdot \boldsymbol{V}, \tag{4}$$

where $\boldsymbol{c}_i$ and $\boldsymbol{b}_i$ are the ith row of $\boldsymbol{C} = \exp(\boldsymbol{Q}\boldsymbol{K}^\top)$ 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 $k \approx \ell \approx \sqrt{m}$.

5.1. Multiplying 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 $j \in [k]$.

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 D.

5.2. Shifting B by maximum

To prevent overflow during the computation of $\mathsf{softmax}$, 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

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 $m \le k \cdot \ell$.

for $s \in [d_h]$ **do**
  for $j \in [k]$ **do** *// Only for baby steps*
    Compute $\mathsf{ct}.\rho_j(\boldsymbol{k}_s) \leftarrow \mathsf{Rot}_j(\mathsf{ct}.\boldsymbol{k}_s)$.
  for $i \in [\ell]$ **do** *// Only for giant steps*
    Compute $\mathsf{ct}.\rho_{-ik}(\boldsymbol{q}_s) \leftarrow \mathsf{Rot}_{-ik}(\mathsf{ct}.\boldsymbol{q}_s)$.
for $i \in [\ell]$ **do**
  for $j \in [k]$ **do**
    Compute $\mathsf{ct}.\rho_{-ik}(\boldsymbol{d}_{ik+j}(\boldsymbol{B}))$ as $\mathsf{Add}_{s\in[d_h]}\left(\mathsf{Mul}\left(\mathsf{ct}.\rho_{-ik}(\boldsymbol{q}_s),\ \mathsf{ct}.\rho_{-j}(\boldsymbol{k}_s)\right)\right)$.

NEXUS (Zhang et al., 2025) and MOAI (Zhang et al., 2026) 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 (Lim et al., 2025), Lim et al. introduced a *statistical maximum estimation* technique into the PPTI setting. This 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

$$\mathbf{E}[x_{\max}] \le \mu + \sqrt{2\ln m}\cdot\sigma. \tag{5}$$

Furthermore, this upper bound is tight up to a constant; see, e.g., (van Handel, 2016, page 50). Lim et al. (2025) utilized this bound to estimate the maximum of an encrypted vector. To avoid computing square roots, they approximate the standard deviation as $\sigma \approx 1.25 + 0.1\cdot\sigma^2$ for $\sigma^2 \in (0, 100)$.

However, the input vectors to the max function may not always satisfy a σ^2-subgaussian distribution. Therefore, we adjust the coefficient in Eq. (5) and instead adopt the following formula:

$$\mathbf{E}[x_{\max}] \le \mu + k_{\max}\sqrt{\ln m}\cdot\sigma, \tag{6}$$

Here, $k_{\max}$ is an empirical constant determined from the training data. For example, on the SST-2 dataset, we obtain $k_{\max} \approx 0.944$. It is calibrated once on the plaintext training set and then fixed as a public parameter before inference. Neither the validation set nor the client's private input is used in this process, which is consistent with our threat model that protects the client's inference data. In addition, we approximate the standard deviation σ using $1.25 + 0.1\cdot\sigma^2$, which is the degree-1 minimax polynomial approximation of $\sqrt{X}$ over $(0, 100)$. The approximation guarantee applies when the row variance lies in $(0, 100)$. Outside this interval, its accuracy is not guaranteed; in our evaluation, the observed row variances remain within this range.

5.2.1. Row-wise summation from rotated diagonal encoding

Algorithm 2 Row-wise summation from rotated diagonals.

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 $m \le k \cdot \ell$.

Output: $\mathsf{ct}.\boldsymbol{s}$ with $\boldsymbol{s} = (\sum_j b_{i,j})_{i\in[m]}$.

for $i \in [\ell]$ **do**
  $\mathsf{ct}.\boldsymbol{t}_i \leftarrow \mathsf{Add}_{j\in[k]}(\mathsf{ct}.\rho_{-ik}(\boldsymbol{d}_{ik+j}(\boldsymbol{B})))$.
$\mathsf{ct}.\boldsymbol{s} \leftarrow \mathsf{Add}_{i\in[\ell]}(\mathsf{Rot}_{ik}(\mathsf{ct}.\boldsymbol{t}_i))$.

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

Recall that after the execution of Algorithm 1, we obtain rotated ciphertexts of diagonals of $\boldsymbol{B}$, i.e., $\mathsf{ct}.\rho_{-ik}(\boldsymbol{d}_{ik+j}(\boldsymbol{B}))$ for $i \in [\ell]$ and $j \in [k]$. However, computing $\max(\boldsymbol{b}_i)$ using Eq. (6) 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.

5.2.2. Shifting $\boldsymbol{B}$ by statistical maximum estimation

We now consider using Eq. (6) 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.

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

Input: $\mathsf{ct}.\rho_{-ik}(\boldsymbol{d}_{ik+j}(\boldsymbol{B}))$ for $i \in [\ell]$ and $j \in [k]$, where $\boldsymbol{B} \in \mathbb{R}^{m\times m}$, $m \le k \cdot \ell$; a parameter $k_{\max}$.

Output: $\mathsf{ct}.\rho_{-ik}(\boldsymbol{d}_{ik+j}(\boldsymbol{B}'))$ for $i \in [\ell]$ and $j \in [k]$, where $\boldsymbol{B}' = (b'_{i,j}) = (b_{i,j} - g_i)$, and g_i is the upper bound on $\mathbf{E}(\max_j(b_{i,j}))$ from Eq. (6).

Calling Algorithm 2 with $\mathsf{ct}.\rho_{-ik}(\boldsymbol{d}_{ik+j}(\boldsymbol{B}))$ for $i \in [\ell]$ and $j \in [k]$ as inputs returns $\mathsf{ct}.\boldsymbol{s}$.
$\mathsf{ct}.\boldsymbol{\mu} \leftarrow \mathsf{CMul}\left(\frac{1}{m}, \mathsf{ct}.\boldsymbol{s}\right)$. *// $\boldsymbol{\mu}$: the mean vector*
for $i \in [\ell]$ **do**
  $\mathsf{ct}.\boldsymbol{\mu}_i \leftarrow \mathsf{CMul}\left(-1, \mathsf{Rot}_{-ik}(\mathsf{ct}.\boldsymbol{\mu})\right)$.
  for $j \in [k]$ **do**
    $\mathsf{ct}.\rho_{-ik}(\boldsymbol{d}_{ik+j}(\boldsymbol{T})) \leftarrow \mathsf{Add}\left(\mathsf{ct}.\rho_{-ik}(\boldsymbol{d}_{ik+j}(\boldsymbol{B})), \mathsf{ct}.\boldsymbol{\mu}_i\right)$.
    Update $\mathsf{ct}.\rho_{-ik}(\boldsymbol{d}_{ik+j}(\boldsymbol{T}))$ to its square.
Calling Algorithm 2 with $\mathsf{ct}.\rho_{-ik}(\boldsymbol{d}_{ik+j}(\boldsymbol{T}))$ for $i \in [\ell]$ and $j \in [k]$ as inputs returns $\mathsf{ct}.(m\sigma^2)$.
$\mathsf{ct}.\sigma^2 \leftarrow \mathsf{CMul}\left(\frac{1}{m}, \mathsf{ct}.(m\sigma^2)\right)$. *// σ^2: the variance vector*
$\mathsf{ct}.\sigma \leftarrow \mathsf{Add}\left(1.25, \mathsf{CMul}\left(0.1, \mathsf{ct}.\sigma^2\right)\right)$.
$\mathsf{ct}.\boldsymbol{g} \leftarrow \mathsf{Add}\left(\mathsf{ct}.\boldsymbol{\mu}, \mathsf{CMul}\left(k_{\max}\sqrt{\ln m}, \mathsf{ct}.\sigma\right)\right)$.
for $i \in [\ell]$ **do**
  $\mathsf{ct}.\boldsymbol{g}_i \leftarrow \mathsf{CMul}\left(-1, \mathsf{Rot}_{-ik}(\mathsf{ct}.\boldsymbol{g})\right)$.
  for $j \in [k]$ **do**
    Compute $\mathsf{ct}.\rho_{-ik}(\boldsymbol{d}_{ik+j}(\boldsymbol{B}'))$ as $\mathsf{Add}\left(\mathsf{ct}.\rho_{-ik}(\boldsymbol{d}_{ik+j}(\boldsymbol{B})), \mathsf{ct}.\boldsymbol{g}_i\right)$.

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

5.3. Exponential evaluation

After shifting $\boldsymbol{B}$ by the statistical maximum, the input domain of the function $\exp(x)$ is contained within the interval $\mathbb{I}_{\exp}$, say $\mathbb{I}_{\exp} = [-60, 15]$. [3] Therefore, we focus on approximating $\exp(x)$ over the interval $\mathbb{I}_{\exp}$.

Our evaluation on SST-2 shows that approximately 5% of the shifted attention scores remain positive, with the maximum value remaining below 14.7. These values are therefore covered by the approximation interval $\mathbb{I}_{\exp} = [-60, 15]$.

Approximate $\exp(x)$ 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^rth power). This method has been used in many FHE-based PPTI protocols, e.g., NEXUS (Zhang et al., 2025), MOAI (Zhang et al., 2026) and Tricycle (Lim et al., 2025). Over the interval $\mathbb{I}_{\exp}$, the maximum absolute error (MAE) can exceed 1.12×10^6.

[3] Eq. (6) provides only a statistical estimate of the maximum and therefore does not guarantee that the entries of $\boldsymbol{B} - \mathbf{E}(\max(\boldsymbol{B}))$, i.e., the inputs to $\exp(x)$, are bounded from above by zero.

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 (Chen et al., 2019) applied this technique in the context of CKKS bootstrapping and proposed a Paterson–Stockmeyer-style (Paterson & Stockmeyer, 1973) method to efficiently evaluate Chebyshev-based polynomials over ciphertexts. In particular, we use a polynomial of degree 44 with Chebyshev basis to approximate $\exp(x)$ over $\mathbb{I}_{\exp}$.

The resulting maximum absolute error, $\max_{x\in\mathbb{I}_{\exp}}|p_{44}(x)-\exp(x)|$, is approximately 1.05×10^{-5} over $\mathbb{I}_{\exp}=[-60,15]$. Evaluating this polynomial requires 7 multiplicative depths.

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

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 $m\le k\cdot\ell$.

Output: $\mathsf{ct}.\rho_{-ik}(\boldsymbol{d}_{ik+j}(\boldsymbol{C}))$ for $i\in[\ell]$ and $j\in[k]$, where $\boldsymbol{C}=\exp(\boldsymbol{B})$.

1: **for** $i\in[\ell]$ **do**
2: **for** $j\in[k]$ **do**
3: Invoking the algorithm presented in [5, §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.*

5.4. Compute $\boldsymbol{CV}$

At this point, we have completed the construction of the ciphertexts of $\rho_{-ik}(\boldsymbol{d}_{ik+j}(\boldsymbol{C}))$, where $\boldsymbol{C}=\exp(\boldsymbol{B})\in\mathbb{R}^{m\times m}$. 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. Algorithms 1 and 5 together define the complete Double-BSGS procedure. Here, "Double-BSGS" refers to applying BSGS to the two encrypted matrix multiplications before and after Softmax, rather than the double-hoisting BSGS in ORION (Ebel et al., 2025), which targets rotation optimization within a single linear transformation.

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* $k\approx\ell\approx\sqrt{m}$.

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 $\boldsymbol{V}\in\mathbb{R}^{m\times d_h}$: $\{\mathsf{ct}.\boldsymbol{v}_0^\top,\dots,\mathsf{ct}.\boldsymbol{v}_{d_h-1}^\top\}$.

Output: $\{\mathsf{ct}.\tilde{\boldsymbol{a}}_0^\top,\dots,\mathsf{ct}.\tilde{\boldsymbol{a}}_{d_h-1}^\top\}$, 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: $\mathsf{ct}.\tilde{\boldsymbol{a}}_s^\top\leftarrow\mathsf{Add}_{i\in[\ell]}(\mathsf{Rot}_{i\cdot k}(\mathsf{ct}.\boldsymbol{t}_i))$

5.5. 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. (4), this step essentially multiplies each row of $\tilde{\boldsymbol{A}}$ by the ciphertext of the scalar $\frac{1}{\sum_j c_{i,j}}$, where $\boldsymbol{C}=(c_{i,j})=\exp(\boldsymbol{QK}^\top)$. This functionality is implemented by Algorithm 6.

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 and 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.

Algorithm 6 Normalization.

Input: Ciphertexts of $\tilde{\boldsymbol{A}}=\exp(\boldsymbol{QK}^\top)\boldsymbol{V}\in\mathbb{R}^{m\times d_h}$ in column packing: $\{\mathsf{ct}.\tilde{\boldsymbol{a}}_0^\top,\dots,\mathsf{ct}.\tilde{\boldsymbol{a}}_{d_h-1}^\top\}$; the outputs of Algorithm 4, i.e., $\mathsf{ct}.\rho_{-ik}(\boldsymbol{d}_{ik+j}(\boldsymbol{C}))$ with $\boldsymbol{C}=\exp(\boldsymbol{QK}^\top)$ 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: $\{\mathsf{ct}.\boldsymbol{a}_0^\top,\dots,\mathsf{ct}.\boldsymbol{a}_{d_h-1}^\top\}$.

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 $\mathsf{ct}.\boldsymbol{s}$. // $s=(\sum_j c_{i,j})_{i\in[m]}$
2: Computing the inverse of each component of $\boldsymbol{s}$ using a Chebyshev polynomial approximation followed by two iterations of the Goldschmidt algorithm [17] 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 at most* $\ell-1$ *ciphertext rotations. Assuming* $\ell\approx\sqrt{m}$, *this is approximately* $\sqrt{m}-1$ *rotations.*

5.6. 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 3.

As shown in Table 3, 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 (Pang et al., 2024), NEXUS (Zhang et al., 2025), PowerFormer (Park et al., 2025), and THOR (Moon et al., 2025). Table 4 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, with ciphertext rotation counts that scale quadratically and linearly with m, respectively. ARION requires

Table 3
Comparison of ciphertext rotations required by CCMM in computing $\mathrm{softmax}(\boldsymbol{QK}^\top)\boldsymbol{V}$ for all H heads in one Transformer layer.

	MOAI (Zhang et al., 2026)		ARION (Ours)	
	Eq.	No.	Eq.	No.
$\boldsymbol{B} = \boldsymbol{QK}^\top$	md_hH	98 304	$2\sqrt{m}d_hH$	18 432
$\boldsymbol{B} - \max(\boldsymbol{B})$	−*	–	$4\sqrt{m}H$	576
$\boldsymbol{C} = \exp(\boldsymbol{B})$	0	0	0	0
$\tilde{\boldsymbol{A}} = \boldsymbol{CV}$	$(m+2\sqrt{m})d_hH$	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 (Zhang et al., 2026) uses a pre-defined constant as the estimation of $\max(\boldsymbol{B})$, no ciphertext rotations are needed. For $m = 128$, we use $k = \ell = 12$ and skip the remaining 16 positions outside the valid range. The reported rotation counts are conservative upper bounds. The per-sequence counts are obtained by dividing the batch-level counts by $n = N/(2m) = 256$.

Table 4
Total required ciphertext rotations for attention computation (L layers, H heads) on BERT-Base.

Protocols	Equation	Total
BOLT† (Pang et al., 2024)	$(\frac{188md}{N} + 4mH\log m + 2mH)L$	> 556 334
NEXUS‡ (Zhang et al., 2025)	$\frac{4m}{N}(md + md_hH + 2d^2)L$	786 432
PowerFormer† (Park et al., 2025)	$(\frac{3dH(\sqrt{3m}+d_h)}{2m} + \frac{H^2\sqrt{m}}{2} + \frac{13mH}{4})L$	> 178 020
THOR (Moon et al., 2025)	$(\frac{1}{32}m^2 + \frac{H+59}{16}m + \frac{3H+5}{16}d_h)L$	14 926
MOAI‡ (Zhang et al., 2026)	$\frac{4(m+\sqrt{m})d_hH}{N/m}L$	10 080
ARION (Ours)‡	$\frac{(8d_hH+10)\sqrt{m}}{N/m}L$	**1 764**

The formulas are based on (Moon et al., 2025, Tab. 4) and Zhang et al. (2026). The parameters are identical to those in Table 3. † The equation is a high-order approximation. ‡ Costs are amortized over the batch.

substantially fewer rotations than all these methods for encrypted attention computation, reducing the asymptotic rotation count by at least a factor of $\sqrt{m}$.

5.7. 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 Section 3.3.2 can be directly applied, *without* any ciphertext rotations.

Regarding the non-linear operations involved in these layers (including $1/\sqrt{x}$ in 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 E.

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*.

6. 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 (Cheon et al., 2017). 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 is computationally indistinguishable from that obtained using an encrypted dummy input of the same dimensions. We provide a rigorous formal proof based on the simulation paradigm in Appendix G.

For approximate homomorphic encryption, standard IND-CPA security does not fully capture scenarios where an adversary can observe approximate decryption results. Li and Micciancio (Li & Micciancio, 2021) formalized this setting through IND-CPA^D security, and Cheon et al. (2020) further analyzed its relevance to practical CKKS deployments. In ARION, the server only observes CKKS ciphertexts and never obtains decrypted inference results. The client decrypts the output using its own secret key and does not reveal the plaintext result to the server. Therefore, the IND-CPA^D key-recovery setting does not provide the server with additional information about the client's input. Since IND-CPA security does not imply circuit privacy and ARION does not apply noise flooding, circuit privacy and cryptographic model privacy against the client are outside the scope of this work.

7. Implementation

We implemented ARION using the CKKS scheme in the homomorphic encryption library Lattigo (Tune-Insight, 2025), 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.

7.1. 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 (Halevi & Shoup, 2018), 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 F.

7.2. 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 the actual data distribution (e.g., mean and standard deviation). As will be shown in the next section, despite this extra cost, ARION is still faster than MOAI.

8. Evaluation

8.1. Setup

All experiments in this section were conducted on a workstation equipped with two Intel Xeon 6530 CPUs (32×2 cores at 2.1 GHz) and 1 TB of system memory. 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 modulus chain such that $\log Q = 58$ (last modulus) $+45 \cdot 14$ (for 14 levels), with $\log P = 60 \cdot 4$ for relinearization and key switching. The scaling factor Δ satisfies $\log \Delta \approx 45$. The bootstrapping parameters are derived from the standard configuration provided by Lattigo v6.1.1 (Tune-Insight, 2025). We retain its sparse ternary secret with Hamming weight 192, ephemeral sparse ternary

Table 5
TCMM with and without 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

Results are obtained using CKKS in Lattigo with $\log N = 13$, $\log q = 190$, and $\log \Delta = 40$.

secret with Hamming weight 32, and default discrete Gaussian error distribution with standard deviation 3.2 and truncation bound 19.2. The bootstrapping configuration uses $\{42 \times 3\}$-bit moduli for Slots-to-Coefficients, $\{60 \times 11\}$-bit moduli for EvalMod, and $\{58 \times 4\}$-bit moduli for Coefficients-to-Slots. Under this setting, the total modulus at bootstrapping satisfies $\log(QP) = 1749$. According to the latest Lattice Estimator (Albrecht et al., 2015), these parameters achieve a 128-bit security level.

Classification accuracy is evaluated on the full development sets of SST-2, QNLI, and RTE from the GLUE benchmark (Wang et al., 2018). The calibration parameter $k_{\max}$ is determined only from plaintext training data and fixed before inference. We assume a semi-honest server, with the secret key held exclusively by the client, and guarantee client-input confidentiality only.

8.2. Micro-benchmarks

8.2.1. 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 5, we give a comparison between the method without (used in MOAI Zhang et al., 2026) 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 5, 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.

8.2.2. 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 6, Double-BSGS achieves a consistent speedup of 1.76–1.89× for attention matrix operations across the evaluated models. For LLaMA-3-8B, the runtime of these operations is reduced from 10051.2 s to 5694.1 s. This experiment evaluates only the attention matrix operations and does not represent end-to-end LLaMA-3-8B inference. RoPE transforms $\boldsymbol{Q}$ and $\boldsymbol{K}$ before attention score computation but does not change the matrix product structure optimized by Double-BSGS. Therefore, Double-BSGS remains applicable to larger models with the same attention structure, provided that other operations do not alter this matrix-product form.

Table 6
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 $H = 32$.

Table 7
Accuracy on the GLUE development sets.

Model	Evaluation	SST-2	QNLI	RTE
BERT-Tiny	Plaintext inference	81.31%	78.69%	60.29%
	Encrypted inference	81.42%	78.66%	59.93%
	Match count	871	5,455	274
	Mismatch count	1	8	3
BERT-Base	Plaintext inference	92.66%	91.12%	70.40%
	Encrypted inference	92.09%	90.66%	69.31%
	Match count	861	5,336	274
	Mismatch count	11	127	3

8.3. 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.

8.3.1. Accuracy

For BERT-Tiny, we evaluate ARION using both the pretrained model and models fine-tuned on QNLI, RTE, and SST-2. We report classification accuracy on the full development sets of QNLI, RTE, and SST-2 for both BERT-Tiny and BERT-Base. For this evaluation, the encrypted pooler output is decrypted, and the classification head is subsequently evaluated in plaintext. For BERT-Base, we additionally evaluate layer-wise numerical accuracy by comparing ARION with our reproduction of MOAI without fine-tuning.

BERT-tiny. As shown in Fig. 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 throughout the encrypted inference process. As a result, the logits produced by ARION remain close to their plaintext counterparts, leading to nearly identical classification results in most cases. We report the agreement over the full development sets in Table 7. We provide concrete examples of input sentences and their corresponding logits in Appendix H.

BERT-base. Fig. 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. The layer-wise errors of MOAI in Fig. 4 are obtained from our own reproduction and exhibit similar trends to those reported in Zhang et al. (2026).

Table 7 reports the classification accuracy of BERT-Tiny and BERT-Base on the full development sets of SST-2, QNLI, and RTE. Across

Table 8
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 Tricycle (Lim et al., 2025) *	BERT-Tiny ARION	BERT-Base MOAI † (Zhang et al., 2026)	BERT-Base ARION
Attention	CPMM	$(\mathbb{R}^{m\times d}\times\mathbb{R}^{d\times d_h})\cdot 3HL$	14.5	0.07	39.46	13.35
	CCMM	$(\mathbb{R}^{m\times d_h}\times\mathbb{R}^{d_h\times m})\cdot HL$	10.7	1.16	51.13	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	41.12	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	106.04	32.89
Self Output	CPMM	$(\mathbb{R}^{m\times d}\times\mathbb{R}^{d\times d})\cdot L$	2.1	0.03	1.83	5.20
	LayerNorm-1	$(\mathbb{R}^{m\times d})\cdot L$	8.2	0.06	0.48	0.43
	Bts	$(\mathbb{R}^{m\times d})\cdot L$		1.67	106.81	33.29
FFN	CPMM	$(\mathbb{R}^{m\times d}\times\mathbb{R}^{d\times d_f})\cdot L$		0.08	51.61	19.80
	GELU	$(\mathbb{R}^{m\times d_f})\cdot L$	13.2	0.47	3.85	14.70
	CPMM	$(\mathbb{R}^{m\times d_f}\times\mathbb{R}^{d_f\times d})\cdot L$		0.02	7.05	4.89
	Bts	$(\mathbb{R}^{m\times d})\cdot L$		1.76	106.41	33.49
Final	LayerNorm-2	$(\mathbb{R}^{m\times d})\cdot L$	9.0	0.04	0.49	0.41
	Bts	$(\mathbb{R}^{m\times d})\cdot L$	234.6	1.74	106.06	33.13
Total			322.0	**9.30**	622.35	**224.91**

Parameters apart from n and m can be found in Table 2. The CCMM + Bts+Reciprocal entry includes the fifth bootstrapping operation in each ARION layer. Since this bootstrapping is applied to a single ciphertext, its runtime is reported together with the surrounding CCMM and reciprocal computations. *The data for Tricycle are borrowed from Lim et al. (2025), 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 MOAI results are obtained by running its publicly available implementation on the same machine used to evaluate ARION. For reference, Zhang et al. (2026) reports an amortized end-to-end latency of 574.6 s. The 41.12 s entry includes both softmax and CCMM. In addition, MOAI performs its first Bts immediately before LayerNorm-1, whereas ARION performs it at a different stage. Including the encrypted pooler adds 0.12 s per inference to the reported ARION runtime.

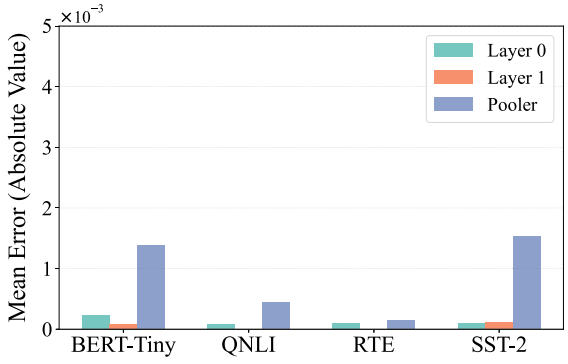


Fig. 3. Absolute mean error of Layer 0, Layer 1, and the Pooler across BERT-Tiny models.

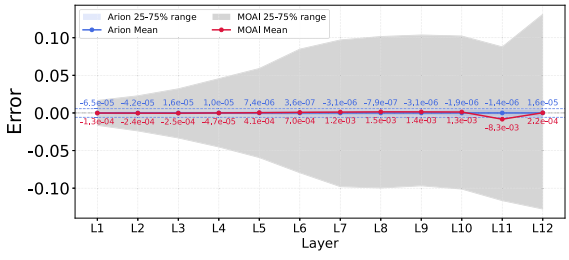


Fig. 4. Full layer-wise error propagation in BERT-Base.

all three tasks, encrypted inference remains close to the corresponding plaintext results, with only a small number of prediction mismatches.

8.3.2. Runtime

Table 8 reports the component-wise latency of encrypted inference for BERT-Tiny and BERT-Base under the ARION framework, together with comparisons against state-of-the-art CPU-based baselines. The ARION results are measured using 64 threads. For BERT-Base, we reproduce MOAI on the same workstation using its original 112 thread configuration.

For BERT-Tiny, ARION reduces the amortized per-sample inference time by 34.6× relative to the state-of-the-art CPU-based baseline reported by Lim et al. (2025) (Tricycle). For BERT-Base, our reproduction of MOAI has an amortized latency of 622.35 s, while ARION requires 224.91 s on the same workstation, corresponding to a 2.77× end-to-end speedup.

MOAI is implemented using SEAL, whereas ARION is built on Lattigo. The results in Table 8 should therefore be viewed as a comparison between two complete systems rather than as a direct ablation of Double-BSGS. The two systems also differ in several aspects of their encrypted inference pipelines. Similar to NEXUS, MOAI approximates the maximum attention score using a fixed constant, while ARION uses statistical maximum estimation, which requires a more complex attention computation. In addition, MOAI uses 112 threads, whereas ARION uses 64 threads. The two systems also require different maximum multiplicative depths: MOAI requires 15 levels, whereas ARION requires 14. This lower depth allows ARION to use a shorter modulus chain, reducing the cost of the underlying FHE operations. The reduction results from the overall computational design of ARION and the scheduling of operations throughout the encrypted inference pipeline, and should therefore be regarded as a system-level optimization. The two systems also differ in their nonlinear polynomial approximations, bootstrapping implementations, and evaluation pipelines. As a result, the 2.77× end-to-end speedup reflects the combined effects of these system-level differences and should not be attributed solely to Double-BSGS or linear-nonlinear fusion. We instead quantify the contribution of Double-BSGS using the controlled experiments in Tables 5 and 6, which show a 1.76–1.89× speedup for the attention matrix operations under the same code base and hardware configuration.

8.3.3. Memory

For the BERT-Base configuration with $m = 128$, $\log N = 16$, and bootstrapping enabled, the complete serialized key material occupies approximately 34.78 GB, including the public, secret, relinearization, ro-

Table 9
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 10
Performance of encrypted DASHformer inference.

Solution	Bts	Memory	Total Time	Amortized Time
Bossuat-I (Bossuat, 2024)	76	58	890	5.46
Bossuat-II (Bossuat, 2024)	58	56	624	3.83
Ours	**0**	**29**	**165**	**1.01**

Memory is reported in GB. Total Time and Amortized Time are reported in seconds.

tation, and bootstrapping keys. The encrypted input incurs an average client communication cost of approximately 42 MB per sequence.

During BERT-Base inference, ARION reaches a peak server memory consumption of approximately 600 GB with 64 threads, whereas our reproduction of MOAI requires approximately 800 GB under its original 112 thread configuration on the same workstation. These peak measurements include the evaluation keys, ciphertexts, and intermediate results allocated throughout the complete inference process.

8.4. Token-length flexibility

To further evaluate scalability across different input sizes, Table 9 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.

8.5. Evaluation on DASHformer

To demonstrate the applicability of ARION beyond standard NLP tasks, we further evaluate the proposed optimization on DASHformer, a compact BERT-based model for protein sequence classification in the iDASH24 benchmark (Harmanci et al., 2024). DASHformer has one Transformer layer with hidden dimension $d = 128$ and sequence length $m = 50$, providing an evaluation scenario for encrypted Transformer inference on biomedical sequence data.

As shown in Table 10, ARION completes inference for 163 encrypted protein sequences of length 50 in 165 s, resulting in an amortized runtime of approximately 1 s per sequence. The encrypted inference achieves a micro-AUC of 0.99 on the iDASH24 benchmark, with prediction performance close to the original DASHformer model. For comparison, the official DASHformer submission reports a micro-AUC of 0.97 on the iDASH24 validation set. Compared with the results of the Bossuat team reported in the official iDASH24 report (Harmanci et al., 2024), ARION achieves a 3.78× speedup. The implementation of the Bossuat team is publicly available through a GitHub repository (Bossuat, 2024) rather than a peer-reviewed publication. This improvement is mainly attributed to the proposed attention optimization and a reduction in multiplicative depth to 10, which enable leveled homomorphic evaluation without expensive bootstrapping.

The achieved latency indicates that ARION can support practical per-sample encrypted inference scenarios, where sensitive protein sequences can be analyzed while keeping the underlying biological data confidential. Although this runtime does not represent a complete clinical diagnostic pipeline, it provides a practical reference for privacy preserving protein sequence analysis and demonstrates the feasibility of applying homomorphic Transformer models to biomedical applications.

9. Conclusion

We present ARION, an efficient, non-interactive FHE-based PPTI protocol that addresses rotation overhead and packing constraints in encrypted attention. Our main contributions are Double-BSGS, which reduces rotations across consecutive encrypted matrix multiplications, and linear–nonlinear fusion, which integrates softmax evaluation into the matrix-product flow while preserving packing compatibility. On BERT-Base, ARION reduces the rotation count by 82.5% compared with MOAI. Controlled experiments in Table 6 show a 1.76–1.89× speedup for attention matrix operations. With our thread-level parallel implementation, our open-source system achieves encrypted inference speedups of 34.6× on BERT-Tiny and 2.77× on BERT-Base compared with Tricycle and our reproduction of MOAI, respectively. These end-to-end results are system-level comparisons and should not be attributed solely to our attention optimization. Evaluation on SST-2, QNLI, and RTE shows accuracy close to that of plaintext inference, and the DASHformer experiment demonstrates applicability to encrypted protein sequence classification. Since Double-BSGS targets the standard attention computation $\mathrm{softmax}(QK^T/\sqrt{d_h})V$, the same optimization can be applied to Transformer architectures that use this attention form. Future work will explore GPU acceleration and extend ARION to larger Transformer models, with support for causal masking, grouped-query attention, and autoregressive decoding.

CRediT authorship contribution statement

Linhan Yang: Conceptualization, Methodology, Software, Investigation, Formal analysis, Visualization, Writing – original draft, Writing – review & editing; **Jingwei Chen:** Conceptualization, Methodology, Supervision, Investigation, Formal analysis, Writing – review & editing; **Wangchen Dai:** Supervision, Resources; **Shuai Wang:** Formal analysis, Writing – review & editing; **Wenyuan Wu:** Resources; **Yong Feng:** Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

We thank the anonymous reviewers for their valuable comments and suggestions, which improved the quality of this paper. This work was supported in part by the National Key Research and Development Program of China (2025YFA1017201) and the National Natural Science Foundation of China (12571553).

Data availability

All data used in this study are publicly available, and the corresponding references are provided. The source code of our implementation is publicly available at https://github.com/hangenba/Arion.

Appendix A. Abbreviations

Table A.1 lists the abbreviations used in this paper and their full forms.

Table A.1
Abbreviations used in this paper.

Abbreviation	Full form or meaning	Abbreviation	Full form or meaning
ARION	Attention-optimized transformer inference on encrypted data	GQA	Grouped-Query Attention
AUC	Area Under the Curve	HE	Homomorphic Encryption
BERT	Bidirectional Encoder Representations from Transformers	IND-CPA	Indistinguishability under Chosen-Plaintext Attack
BSGS	Baby-Step Giant-Step	IND-CPA^D	IND-CPA security with exposure to approximate decryption results
Double-BSGS	Double Baby-Step Giant-Step	KV	Key and Value
C2C	Column packing to column packing	LLM	Large Language Model
CC2D	Column and column packing to diagonal packing	MPC	Multi-Party Computation
DC2C	Diagonal and column packing to column packing	NLP	Natural Language Processing
CCMM	Ciphertext-Ciphertext Matrix Multiplication	PPML	Privacy-Preserving Machine Learning
CKKS	Cheon–Kim–Kim–Song	PPTI	Privacy-Preserving Transformer Inference
CMT	Ciphertext Matrix Transposition	QNLI	Question Natural Language Inference
CPMM	Ciphertext-Plaintext Matrix Multiplication	RLWE	Ring Learning With Errors
CPU	Central Processing Unit	RNS	Residue Number System
FFN	Feed-Forward Network	RoPE	Rotary Position Embedding
FHE	Fully Homomorphic Encryption	RTE	Recognizing Textual Entailment
GELU	Gaussian Error Linear Unit	SIMD	Single Instruction, Multiple Data
GLUE	General Language Understanding Evaluation	SST-2	Stanford Sentiment Treebank, binary classification
GPT	Generative Pretrained Transformer	TCMM	Triple Ciphertext Matrix Multiplication
GPU	Graphics Processing Unit		

Appendix B. The CKKS scheme

The Cheon-Kim-Kim-Song scheme (CKKS) (Cheon et al., 2017) 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 $\mathsf{ct}.\boldsymbol{x} = \mathsf{Enc}(x_0, \dots, x_{N/2-1})$ into a new ciphertext $\mathsf{Enc}(x_k, \dots, x_{N/2-1}, x_0, \dots, x_{k-1})$. 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 (Lyubashevsky et al., 2013). 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 C. 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 (Devlin et al., 2019).

C.1. 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:

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

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 hth head can be computed in parallel:

$$\boldsymbol{A}_i^{(h)} = \mathsf{softmax}\left(\frac{\boldsymbol{Q}_i^{(h)}\boldsymbol{K}_i^{(h)\top}}{\sqrt{d_h}}\right)\boldsymbol{V}_i^{(h)} \in \mathbb{R}^{m\times d_h}, \tag{C.1}$$

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

$$\mathsf{softmax}(\boldsymbol{x}) = \left(\frac{\exp(x_i)}{\sum_i \exp(x_i)}\right)_i. \tag{C.2}$$

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

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

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 $\boldsymbol{b}_0 \in \mathbb{R}^d$.

C.2. 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 $\boldsymbol{x} = (x_i)_i$,

$$\mathsf{LayerNorm}(\boldsymbol{x}) = \left(\gamma \cdot \frac{x_i - \mu}{\sqrt{\sigma^2}} + \beta\right)_i \tag{C.3}$$

with μ and σ^2 the mean and variance of $\boldsymbol{x}$. The 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 $\boldsymbol{W}_1 \in \mathbb{R}^{d\times d_f}$, $\boldsymbol{W}_2 \in \mathbb{R}^{d_f\times d}$, $\boldsymbol{b}_1 \in \mathbb{R}^{d_f}$, $\boldsymbol{b}_2 \in \mathbb{R}^{d}$,

$$\mathsf{GELU}(x) = \frac{1}{2}x \cdot \left(1 + \mathrm{erf}\left(\frac{x}{\sqrt{2}}\right)\right) \tag{C.4}$$

where $\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}}\int_0^x \mathrm{e}^{-t^2}dt$ 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.3. 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 D. Missing proofs

In this appendix, we provide the formal proofs for the propositions stated in Section 5.

D.1. Proof of Proposition 1

Proof. Let $\boldsymbol{q}_s$ and $\boldsymbol{k}_s$ be the sth 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

$$\boldsymbol{d}_{i'}(\boldsymbol{B}) = \sum_{s\in[d_h]} \boldsymbol{q}_s \odot \rho_{i'}(\boldsymbol{k}_s). \tag{D.1}$$

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. (D.1) 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:

$$\rho_{-ik}(\boldsymbol{d}_{ik+j}(\boldsymbol{B})) = \sum_{s\in[d_h]} \rho_{-ik}(\boldsymbol{q}_s) \odot \rho_j(\boldsymbol{k}_s). \tag{D.2}$$

Algorithm 1 evaluates $\mathsf{ct}.\rho_{-ik}\big(\boldsymbol{d}_{ik+j}(\boldsymbol{B})\big)$ by strictly following Eq. (D.2), thus 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 $(k+\ell-2)d_h$. □

D.2. Proof of Proposition 2

Proof. Algorithm 2 computes:

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

This implies the ith 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. □

D.3. 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.4. Proof of Proposition 4

Proof. Let p_{exp} denote the Chebyshev polynomial used to approximate exp. Since polynomial evaluation is applied independently to each slot, it preserves the ciphertext packing structure and commutes with rotations:

$$p_{\mathsf{exp}}\big(\rho_{-ik}(\boldsymbol{d}_{ik+j}(\boldsymbol{B}))\big) = \rho_{-ik}\big(\boldsymbol{d}_{ik+j}(p_{\mathsf{exp}}(\boldsymbol{B}))\big).$$

Therefore, Algorithm 4 correctly produces the rotated diagonals of $\boldsymbol{C} = p_{\mathsf{exp}}(\boldsymbol{B}) \approx \mathsf{exp}(\boldsymbol{B})$. Since polynomial evaluation does not involve any slot permutation, no ciphertext rotation is required. □

D.5. Proof of Proposition 5

Proof. The plaintext counterpart of Algorithm 5 computes:

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

The final equality holds based on the Halevi-Shoup algorithm (Halevi & Shoup, 2014). Since the rotated diagonals of $\boldsymbol{C}$ are precomputed inputs, the algorithm requires only $k+\ell-2$ rotations for each column $s \in [d_h]$. □

D.6. 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. (4). Rotations are only needed for the summation step, requiring at most $\ell-1$ operations. □

Appendix E. 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.

E.1. Square root reciprocal

The definition of Layer Normalization is given in Eq. (C.3). 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 E.1.

E.1.1. GELU and Tanh

The definition of GELU is provided in Eq. (C.4). In practice, it can also be approximated by the following formula,

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

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 Lim et al. (2025), Rovida and Leporati (2024); see Table E.1 for parameters we use.

Table E.1
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
GELU(x)	FFN	255	–
Tanh(x)	Classification	63	–

Appendix F. 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.

F.1. Optimization on multiplication

We strive to minimize the required multiplicative depth and computational cost by leveraging mathematically equivalent formulations. For example, to realize the computation of $\boldsymbol{QK}^\top/\sqrt{d_h}$, 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 $\boldsymbol{QK}^\top$ 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.

F.2. 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 (Halevi & Shoup, 2018). By pre-computing and reusing the modulus switching and key-switching decomposition parts of the ciphertext, Hoisting significantly reduces the amortized cost of rotations.

F.3. 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 $2^{\log_2(N/2)-\ell}$ 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.

F.4. 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 G. Formal security proof

We rigorously analyze the confidentiality of the client's input against a semi-honest server. The server's view is formalized as a tuple: (Private Inputs; Received Messages; Sent Messages).

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

Here, $\boldsymbol{X}$ represents the client's private input sequences, $\mathcal{W}$ denotes the model parameters held by the server, $\mathsf{ct}.\boldsymbol{X}$ is the encrypted input received by the server, and $\mathsf{ct}.\boldsymbol{Y}$ is the encrypted inference result returned to the client.

Simulation of Server's view

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

We construct a simulator S_2 that does not have access to the real input $\boldsymbol{X}$. Instead, it generates a dummy input $\boldsymbol{X}'$, such as a zero matrix of the same dimensions as $\boldsymbol{X}$, and computes

$$\mathsf{ct}.\boldsymbol{X}' = \mathsf{Enc}_{pk}(\boldsymbol{X}'), \qquad \mathsf{ct}.\boldsymbol{Y}' = \mathsf{Eval}_{evk}(\mathcal{W}, \mathsf{ct}.\boldsymbol{X}').$$

The simulated view is

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

Based on the IND-CPA security of CKKS under the RLWE assumption, $\mathsf{ct}.\boldsymbol{X}$ is computationally indistinguishable from $\mathsf{ct}.\boldsymbol{X}'$. Since homomorphic evaluation is a polynomial-time transformation, the corresponding server views are also computationally indistinguishable:

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

Therefore, the server learns no information about the client's private input $\boldsymbol{X}$.

In summary, ARION protects the confidentiality of the client's input against a semi-honest server. This proof does not establish malicious security, circuit privacy, or cryptographic model privacy against the client.

Appendix H. Detailed accuracy examples

To further illustrate the precision of ARION, Table H.1 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 H.1
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 [class_1, class_2]. The encrypted inference yields predictions identical to the plaintext baseline.

References

Aharoni, E., Adir, A., Baruch, M., Drucker, N., Ezov, G., Farkash, A., Greenberg, L., Masalha, R., Moshkowich, G., Murik, D., Shaul, H., & Soceanu, O. (2023). HeLayers: A tile tensors framework for large neural networks on encrypted data. *PoPETs, 2023*(1), 325–342. https://doi.org/10.56553/popets-2023-0020.

Al Badawi, A. et al. (2022). OpenFHE: Open-source fully homomorphic encryption library. In M. Brenner, A. Costache, & K. Roholff (Eds.), *Proceedings of the 10th workshop on encrypted computing & applied homomorphic cryptography (los angeles, USA, 7 november 2022)* (pp. 53–63). New York: ACM. https://doi.org/10.1145/3560827.3563379.

Albrecht, M. R., Player, R., & Scott, S. (2015). On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology, 9*(3), 169–203. https://doi.org/10.1515/jmc-2015-0016. Lattice Estimator: https://github.com/malb/lattice-estimator.

Bossuat, J.-P. (2024). A solution for iDASH 2024 - HE track. https://github.com/gausslabs/idash-2024-solution.

Chen, H., Chillotti, I., & Song, Y. (2019). Improved bootstrapping for approximate homomorphic encryption. In *Eurocrypt 2019* (pp. 34–54). Cham: Springer (*vol. 11477*). LNCS. https://doi.org/10.1007/978-3-030-17656-3_2.

Chen, J., Yang, L., Wu, W., Liu, Y., & Feng, Y. (2025). Homomorphic matrix operations under bicyclic encoding. *IEEE Transactions on Information Forensics and Security*, *20*, 1390–1404. https://doi.org/10.1109/TIFS.2024.3490862,

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

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

Cheng, K., Xia, Y., Song, A., Fu, J., Qu, W., Shen, Y., & Zhang, J. (2025). 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* (pp. 4124–4138). ACM. https://doi.org/10.1145/3719027.3765028.

Cheon, J. H., Hong, S., & Kim, D. (2020). Remark on the security of CKKS scheme in practice. Cryptology ePrint Archive, Paper 2020/1581. https://eprint.iacr.org/2020/1581.

Cheon, J. H., Kim, A., Kim, M., & Song, Y. (2017). Homomorphic encryption for arithmetic of approximate numbers. In *Asiacrypt 2017* (pp. 409–437). Heidelberg: Springer (*vol. 10624*). LNCS. https://doi.org/10.1007/978-3-319-70694-8_15.

DESILO (2023). Liberate.FHE v0.9.0. https://github.com/Desilo/liberate-fhe.

Devlin, J., Chang, M., Lee, K. et al. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In J. Burstein, C. Doran, & T. Solorio (Eds.), *Naacl '19* (pp. 4171–4186). ACL. https://doi.org/10.18653/v1/N19-1423.

Ebel, A., Garimella, K., & Reagen, B. (2025). Orion: A fully homomorphic encryption framework for deep learning. In *Proceedings of the 30th ACM international conference on architectural support for programming languages and operating systems, volume 2* (pp. 734–749). ACM. https://doi.org/10.1145/3676641.3716008

Gao, X., Fu, S., Liu, L., Liu, Z., Luo, Y., & Wang, Y. (2026). Euston: Efficient and user-friendly secure transformer inference with non-interactivity. In *Proceedings of the IEEE symposium on security and privacy (SP)* (pp. 882–901). https://doi.org/10.1109/SP63933.2026.00048.

Gentry, C. (2009). Fully homomorphic encryption using ideal lattices. In M. Mitzenmacher (Ed.), *Stoc '09* (pp. 169–178). New York: ACM. https://doi.org/10.1145/1536414.1536440.

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

Grattafiori, A., Dubey, A., Jauhri, A. et al. (2024). The Llama 3 herd of models. https://arxiv.org/pdf/2407.21783.

Halevi, S., & Shoup, V. (2014). Algorithms in HElib. In *Crypto '14* (pp. 554–571). Heidelberg: Springer (*vol. 8616*). LNCS. https://doi.org/10.1007/978-3-662-44371-2_31.

Halevi, S., & Shoup, V. (2015). Bootstrapping for HElib. In *Eurocrypt '15* (pp. 641–670). Heidelberg: Springer (*vol. 9056*). LNCS. https://doi.org/10.1007/978-3-662-46800-5_25.

Halevi, S., & Shoup, V. (2018). Faster homomorphic linear transformations in HElib. In *Crypto '18* (pp. 93–120). Cham: Springer (*vol. 10991*). LNCS. https://doi.org/10.1007/978-3-319-96884-1_4.

van Handel, R. (2016). Probability in high dimension. https://web.math.princeton.edu/~rvan/APC550.pdf.

Hao, M., Li, H., Chen, H. et al. (2022). Iron: Private inference on transformers. In *NeurIPS '22* (pp. 15718–15731). Curran Associates. https://proceedings.neurips.cc/paper_files/paper/2022/file/64e2449d74f84e5b1a5c96ba7b3d308e-Paper-Conference.pdf.

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

Jiang, X., Kim, M., Lauter, K., & Song, Y. (2018). Secure outsourced matrix computation and application to neural networks. In *Ccs '18* (pp. 1209–1222). New York: ACM. https://doi.org/10.1145/3243734.3243837.

Kei, A. Y. L., & Chow, S. S. M. (2025). SHAFT: Secure, handy, accurate and fast transformer inference. In *Ndss '25*. The Internet Society. https://doi.org/10.14722/ndss.2025.242287.

Lee, E., Lee, J.-W., Lee, J., Kim, Y.-S., Kim, Y., No, J.-S., & Choi, W. (2022). Low-complexity deep convolutional neural networks on fully homomorphic encryption using multiplexed parallel convolutions. In *Icml '22* (pp. 12403–12422). PMLR. https://proceedings.mlr.press/v162/lee22e.html.

Li, B., & Micciancio, D. (2021). On the security of homomorphic encryption on approximate numbers. In A. Canteaut, & F. X. Standaert (Eds.), *Eurocrypt 2021* (p. 648–677). Cham: Springer (*vol. 12696*). LNCS. https://doi.org/10.1007/978-3-030-77870-5_23.

Li, D., Wang, H., Shao, R., Guo, H., Xing, E. P., & Zhang, H. (2023). MPCFORMER: Fast, performant and private transformer inference with MPC. In *ICLR 2023*. OpenReview.net. https://openreview.net/forum?id=CWmvjOEhgH-.

Li, Y., Zhou, X., Wang, Y., Qian, L., & Zhao, J. (2025). Private transformer inference in MLaaS: A survey. https://arxiv.org/abs/2505.10315.

Lim, L., Kalagi, V., Agrawal, D., & Abbadi, A. E. (2025). Tricycle: Private transformer inference with tricyclic encodings. https://ia.cr/2025/1200.

Luo, J., Zhang, Y., Zhang, Z., Zhang, J., Mu, X., Wang, H., Yu, Y., & Xu, Z. (2024). SecFormer: Fast and accurate privacy-preserving inference for transformer models via SMPC. In *Acl '24* (pp. 13333–13348). Bangkok, Thailand: ACL. https://doi.org/10.18653/v1/2024.findings-acl.790.

Lyubashevsky, V., Peikert, C., & Regev, O. (2013). On ideal lattices and learning with errors over rings. *Journal of ACM*, *60*(6), 43:1–35. https://doi.org/10.1145/2535925.

Microsoft (Accessed in July, 2023). cMicrosoft SEAL (release 4.1.1). https://github.com/microsoft/SEAL.

Moon, J., Yoo, D., Jiang, X., & Kim, M. (2025). THOR: Secure transformer inference with homomorphic encryption. In *Ccs '25* (pp. 3765–3779). New York: ACM. https://doi.org/10.1145/3719027.3765150.

Ng, L. K. L., & Chow, S. S. M. (2023). SoK: Cryptographic neural-network computation. In *S& p '23* (pp. 497–514). Los Alamitos: IEEE. https://doi.org/10.1109/SP46215.2023.10179483

Pang, Q., Zhu, J., Möllering, H., Zheng, W., & Schneider, T. (2024). BOLT: Privacy-preserving, accurate and efficient inference for transformers. In *Ieee s & p '24* (pp. 133–133). Los Alamitos: IEEE. https://doi.org/10.1109/SP54263.2024.00130.

Park, D., Lee, E., & Lee, J.-W. (2025). Powerformer: Efficient privacy-preserving transformer with batch rectifier-power max function and optimized homomorphic attention. In *Acl '25* (pp. 11090–11111). ACL. https://aclanthology.org/2025.acl-long.543.pdf.

Paterson, M. S., & Stockmeyer, L. J. (1973). On the number of nonscalar multiplications necessary to evaluate polynomials. *SIAM Journal on Computing*, *2*(1), 60–66. https://doi.org/10.1137/0202007.

Radford, A., Narasimhan, K., Salimans, T. et al. (2018). Improving language understanding by generative pre-training. https://openai.com/research/language-unsupervised.

Rovida, L., & Leporati, A. (2024). Transformer-based language models and homomorphic encryption: An intersection with BERT-tiny. In *Iwspa '24* (pp. 3–13). New York: ACM. https://doi.org/10.1145/3643651.3659893.

Tune-Insight (2025). Lattigo v6.1.1. https://github.com/tuneinsight/lattigo.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. In *Nips '17*. New York: Curran Associates, Inc. https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., & Bowman, S. R. (2018). GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP workshop blackboxNLP: Analyzing and interpreting neural networks for NLP* (pp. 353–355). Association for Computational Linguistics. https://doi.org/10.18653/v1/W18-5446

Wang, Z., Chen, R., Gao, X., Wang, Y., Liu, L., Lan, Z., Wang, Z., Fu, S., Wang, Q., & Huang, X. (2026). STIP: Efficient and secure non-interactive transformer inference via compact packing. Cryptology ePrint Archive, Paper 2026/174. https://eprint.iacr.org/2026/174.

Xu, T., Lu, W., Yu, J., Chen, Y., Lin, C., Wang, R., & Li, M. (2025). 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* (pp. 2653–2672). USENIX Association. https://ia.cr/2025/1532.

Yang, H., Shen, S., Dai, W., Zhou, L., Liu, Z., & Zhao, Y. (2024). Phantom: A CUDA-accelerated word-wise homomorphic encryption library. *IEEE Transactions on Dependable and Secure Computing*, *21*(5), 4895–4906. https://doi.org/10.1109/TDSC.2024.3363900.

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

Zeng, W., Xu, T., Chen, Y., Zhou, Y., Zhang, M., Tan, J., Hong, C., & Li, M. (2025). Towards efficient privacy-preserving machine learning: A systematic review from protocol, model, and system perspectives. https://arxiv.org/abs/2507.14519.

Zhang, J., Yang, X., He, L., Chen, K., Lu, W.-j., Wang, Y., Hou, X., Liu, J., Ren, K., & Yang, X. (2025). Secure transformer inference made non-interactive. In *Ndss '25*. Fredericksburg: The Internet Society. https://dx.doi.org/10.14722/ndss.2025.230868.

Zhang, L., Wang, X., Sim, J. J., Huang, Z., Zhong, J., Wang, H., Duan, P., & Lam, K.-Y. (2026). Moai: Module-optimizing architecture for non-interactive secure transformer inference. In *The fourteenth international conference on learning representations*. https://openreview.net/forum?id=qJn4HtTzhH.

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

Zimerman, I., Baruch, M., Drucker, N., Ezov, G., Soceanu, O., & Wolf, L. (2024b). Converting transformers to polynomial form for secure inference over homomorphic encryption. In *Icml '24* (pp. 62803–62814). PMLR. https://proceedings.mlr.press/v235/zimerman24a.html.