FELID: A Flexible and Efficient Design for Transformer Fine-Tuning over Encrypted Data

Linru Zhang
Nanyang Technological University

Jun Jie Sim
Ant International, Ant Group

Xiangning Wang
Nanyang Technological University

Jiahao Zhong
Nanyang Technological University

Kaiyu Zhou
Nanyang Technological University

Huanyi Ye
Nanyang Technological University

Yongsen Zheng
Nanyang Technological University

Lushan Song
Ant International, Ant Group

Xiaojian Liang
Ant International, Ant Group

Yingting Liu
Ant International, Ant Group

Yujing Sun
Nanyang Technological University

Huaxiong Wang
Nanyang Technological University

Pu Duan*
Ant International, Ant Group

Kwok-Yan Lam*
Nanyang Technological University

Abstract

Fully Homomorphic Encryption (FHE) enables non-interactive, privacy-preserving transformer fine-tuning, protecting highly sensitive data as inputs to large language models. Prior work has predominantly pursued efficiency gains through architectural refinements, yet overall performance remains bounded by the substantial cost inherent in encrypted computation. To address these issues, we target the bottleneck at the ciphertext operation level. Concretely, we propose a novel packing strategy that supports batching over a flexible number of inputs, making it well-suited for fine-tuning workloads. In addition, we design operation-specific matrix evaluation algorithms that significantly reduce (i) the number of ciphertext rotations and relinearizations, and (ii) the number of bootstrapping invocations. Additionally, we exploit functional bootstrapping to fold the evaluation of element-wise activation functions directly into the bootstrapping step, thereby reducing overall computational complexity. As a result, our framework achieves a $30\times$ reduction in ciphertext rotations and relinearizations across all matrix operations, as well as a $2.5\times$ reduction in FHE bootstrapping invocations over the entire computation flow, compared to the latest state-of-the-art work. It is worth noting that our system completes fine-tuning of a 2-layer BERT-style model with a batch size of 16 in 650 seconds, achieving a $8\times$ speedup over the state-of-the-art work.

*Corresponding author

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

1 Introduction and Related Works

Given the growing privacy concerns over AI-as-a-Service, particularly for institutions with strict privacy requirements, such as financial institutions and law enforcement, Privacy-Preserving Machine Learning (PPML) [13, 19, 17] has attracted significant attention. However, despite continuous efforts to address this privacy problem via cryptography-based techniques, they mostly suffer from a critical limitation in efficiency, with Secure Multi-Party Computation (MPC)-based methods [34, 5, 23, 26, 20, 35, 10] incurring substantial interactive communication overhead, while Fully Homomorphic Encryption (FHE)-based solutions face extremely high computational costs [36, 25, 27, 38, 32].

Given the practical limitations of interactive MPC-based methods, particularly the need for clients to remain continuously online and sustain stable, low-latency communication with the AI Cloud Service Provider (AI CSP) during a highly time-intensive process, non-interactive FHE-based approaches are draw growing interest. Specifically, with the widespread adoption of Transformer architectures, particularly in Large Language Models (LLMs) that are commonly deployed as cloud services, make efforts are made in enabling privacy-preserving machine learning (PPML) for Transformers using non-interactive FHE. However, most work [36, 25, 27, 38, 40, 37] focuses on applying FHE to Transformers during the *inference phase*, **leaving privacy-preserving fine-tuning during the *training phase*** inadequately explored, which is crucial for task-specific adaptation and personalized responses. Although [18, 19] have attempted to address this task, they are largely limited to fine-tuning the classification head while keeping the core Transformer layers untouched, failing to adequately address computational overhead.

Indeed, efficient privacy-preserving fine-tuning of Transformers with FHE is highly challenging, as FHE is mathematically secure but computationally expensive. To address this, two nontrivial aspects need to be considered: Q1). MODEL DESIGN. The efficiency of FHE varies across different operations. Replacing computationally intensive operations with more FHE-friendly alternatives can significantly improve overall performance. Q2). FHE DESIGN. More crucially, for each operation, the efficiency of FHE evaluation should be carefully optimized to reduce computational overhead. Most recently, Rho et al. [28] pioneered addressing FHE-friendly Model Design (Q1) via transforming standard Transformer models into a FHE computational friendly variant by adopting a LoRA-based fine-tuning strategy and replacing the softmax operation in attention with a Gaussian kernel (GK), but leave the more challenging problem of efficient FHE evaluation design (Q2) fully underexplored.

1.1 Our Contributions

To bridge this gap, based on the model architecture of [28], we propose **FELID**, a **F**l**e**xib**l**e and Eff**i**cient **D**esign for transformer fine-tuning over encrypted data, focusing on a novel and efficient FHE evaluation design. We modify none of the model architecture or fine-tuning procedure, and thus our framework is fully compatible with existing model designs [28]. Specifically, matrix evaluation and bootstrapping dominate the cost of FHE evaluation, accounting for approximately $70\% - 85\%$ of the runtime in [28]. We reduce the cost from all perspectives.

A Flexible Column-Wise Batch Packing Strategy is first proposed for encrypting the input data. *Batching multiple inputs* can improve the amortized latency. [2] *Flexible* batch sizes make it well-suited for Transformer fine-tuning with variable batch size (typically 4-64), and adaptable to diverse tasks and training settings.[3] And *column-wise* manner, compared to row-wise manner, further accelerates matrix multiplication.

Efficient Matrix Operations are presented to **(1).** Reduce the number of expensive FHE operations in matrix evaluations to decrease the computational complexity. [4] For ciphertext-plaintext matrix

[2]For example in CPMM without batching, which computes linear combinations over the d encrypted columns of one input in one ciphertext, aggregation of these columns with $O(\log(d))$ ciphertext rotations is required. Batching b inputs with b ciphertexts enables distributing the d columns across b ciphertexts, and the FHE rotations can be reduced from $O(\log(d))$ to $O(\log(d/b))$. When batching d inputs with d ciphertexts, the aggregation phase eliminates ciphertext rotations entirely, substantially lowering the per-input computational cost.

[3]As shown in Table 1, existing works in FHE-based transformer evaluation either do not support batching or rely on a fixed batch size strategy, both of which are unsuitable for fine-tuning scenarios with variable batch sizes.

[4]Ciphertext rotations and relinearizations are widely recognized as the most expensive operations in FHE-based matrix operations [25, 27, 38, 32].

Table 1: Comparison of packing methods in FHE-based non-interactive privacy-preserving transformer inference/fine-tuning schemes.

Scheme	NEXUS [36]	THOR [25]	Powerformer [27]	MOAI [38]	Tricycle [21]	ARION [32]	Rho et al. [28]	Our work
Task	Inference	Inference	Inference	Inference	Inference	Inference	Fine-tuning	Fine-tuning
Batch inputs	✓	✗	✗	✓	✗	✓	✗	✓
Flexible size	✗	-	-	✗	-	✗	-	✓

multiplication in attention head and feedforward layers, we achieve $213\times$ fewer ciphertext rotations and 2 fewer multiplicative levels, and for ciphertext-ciphertext matrix multiplications in LoRA, we achieve $21\times$ fewer ciphertext rotations and relinearizations and 1 fewer multiplicative level, compared to [28] in a batch size of 16. **(2)**. Reduce the multiplicative depth consumption to decrease the number of bootstrappings in the pipeline, which is required to refresh cipher-texts once the remaining levels reach 0. For instance, in kernel attention, we propose "column packing $\rightarrow$ diagonal packing" evaluation pipeline that significantly lowers the multiplicative depth required to compute $\|Q_{i,:} - K_{j,:}\|_2^2$, reducing it from 7 in [28] to 3. [5]

Our contributions are summarized as:

- We proposed a novel FHE Evaluation Design that significantly improved the privacy-preserving fine-tuning of transformers, via addressing the computational bottleneck in Matrix Evaluation and Bootstrapping.
- From computational complexity perspective, our framework achieves a $30\times$ reduction in ciphertext rotations and relinearizations across all matrix operations, as well as a $2.5\times$ reduction in FHE bootstrapping invocations over the entire computation flow, compared to SoTA [28].
- From practical perspective, we complete fine-tuning of a 2-layer BERT-style model with a batch size of 16 in 650 seconds, achieving a $8\times$ speedup over SoTA [28].

2 Preliminaries

Notations. For a positive integer m, let $[m]$ be the set of $\{0, 1, 2, ..., m-1\}$. Given a modulus p we use $a\%p$ for $a \pmod p$. We use uppercase letters to denote a matrix A, and use $A[i][j]$ to represent the element at the i-th row and j-th column of A. $A[i][:]$ is used to denote the i-th row of A. We use bold lowercase letters to represent vectors: $\boldsymbol{v}$, and use $v[i]$ or v_i to represent the i-th element of $\boldsymbol{v}$. The Hadamard product, or the element-wise multiplication is denoted with $\odot$, and is applied between vectors or matrices of identical sizes $\boldsymbol{v_1} \odot \boldsymbol{v_2}$, $A \odot B$. For vector $\boldsymbol{v}$ we define its rotation by j elements to the left as $\text{Rot}_j(\boldsymbol{v}) = (v_j, v_{j+1}, ...v_0, ..., v_{j-1})$.

2.1 Fully Homomorphic Encryption (FHE)

FHE is a special cryptographic primitive that allows one to perform computations on encrypted data without decryption. This property makes FHE one of the most powerful and widely applied privacy-enhancing technologies. In this work, we will use the Cheon-Kim-Kim-Song (CKKS) FHE scheme [7, 6] which supports arithmetic of complex numbers in $\mathbb{C}$ in single-instruction-multiple-data (SIMD) style for any given ciphertext. More concretely, given N the encryption parameter which is a power-of-2, the CKKS scheme can pack at most N complex numbers. Let $ct_1 := \text{Enc}(\boldsymbol{m}_1)$, $ct_2 := \text{Enc}(\boldsymbol{m}_2)$, the CKKS scheme supports the following encrypted computations:
- $\text{Add}(ct_1, ct_2)$. It outputs $\text{Enc}(\boldsymbol{m}_1 + \boldsymbol{m}_2)$. We use $ct_1 + ct_2$ for short.
- $\text{Mult}(ct_1, ct_2)$. It outputs $\text{Enc}(\boldsymbol{m}_1 \odot \boldsymbol{m}_2)$. We use $ct_1 \odot ct_2$ for short.
- $\text{PMult}(\boldsymbol{v}, ct_1)$. It outputs $\text{Enc}(\boldsymbol{v} \odot \boldsymbol{m}_1)$. We use $\boldsymbol{v} \odot ct_1$ for short.
- $\text{Rot}_j^{\text{HE}}(ct_1)$ outputs $\text{Enc}(\text{Rot}_j(\boldsymbol{m}_1))$; $\text{Conj}(ct_1)$ outputs $\text{Enc}(\bar{\boldsymbol{m}}_1)$, the conjugation of $\boldsymbol{m}_1$.

Bootstrapping and Functional Bootstrapping. In the CKKS FHE scheme, homomorphic multiplications inject noise into the underlying message of a ciphertext, and each ciphertext is assigned

[5] $Q_{i,:}$ and $K_{j,:}$ are the i-th and the j-th row of Q, K. Our matrix evaluation algorithms differ from those in MOAI [38] and ARION [32]. Although both use a "column packing $\rightarrow$ diagonal packing " strategy in attention block, they compute $QK^\intercal$, whereas our method targets l_2-norm of the rows from Q and K (kernel definition), so the underlying matrix evaluation algorithm is totally different.

a level to track the noise growth. A fresh ciphertext is considered to be at level L, and $\mathrm{Mult}(\cdot,\cdot)$ and $\mathrm{PMult}(\cdot,\cdot)$ consume one level each. A ciphertext at level 0 would yield a decryption error. Bootstrapping [6] is a technique that "refreshes" a ciphertext by restoring levels to enable more computations. Functional bootstrapping [2, 33] extends this idea by combining ciphertext refreshing with function evaluation in a single procedure, incurring little to no additional overhead.

2.2 Privacy-Preserving Language Model Fine-Tuning

Kernel Attention. The softmax operator in the standard attention mechanism (more details are in Appendix A.2) can be viewed as a row-wise normalized exponential kernel over query–key similarities, or more generally, a kernel-based weighting mechanism in the sense of classical kernel methods [3]. In FHE, however, the softmax operator is difficult to evaluate as it involves several hard-to-approximate operations such as exponentiation, division, and row-wise maximum used for numerical stabilization [8, 27, 28]. A natural encryption-friendly alternative is therefore to replace softmax with a FHE-friendly kernel function. [28] considered the *Gaussian kernel attention*, defined for each head $h \in [H]$ as

$$\mathrm{GK\,Attn}(Q_h, K_h, V_h) := U_h V_h, \qquad U_h[i][j] := \exp\left(-\frac{1}{2\sqrt{n_2/H}}\|Q_h[i][:] - K_h[j][:]\|_2^2\right),$$

where $Q_h, K_h, V_h \in \mathbb{R}^{n_1 \times (n_2/H)}$, $U_h \in \mathbb{R}^{n_1 \times n_1}$. n_1 denotes token length, n_2 the hidden dimension, and H the number of heads. Compared to softmax attention, this formulation eliminates the need to evaluate the division and max function over encrypted data. Moreover, the exponential function is only evaluated on non-positive inputs, where it is bounded and more amenable to polynomial approximation for FHE.

Recently, the Epanechnikov kernel has been studied in dense associative memory and compact kernel regression, where its bounded support naturally induces sparse attention scores [14, 29]. In an unnormalized FHE-friendly form, *Epanechnikov kernel attention* can be written as

$$\mathrm{EK\,Attn}(Q_h, K_h, V_h) := U_h V_h, \qquad U_h[i][j] := \mathrm{ReLU}\left(1 - \frac{1}{2\eta_h^2}\|Q_h[i][:] - K_h[j][:]\|_2^2\right),$$

where $\eta_h > 0$ is a learnable parameter for the h-th head. In our implementation, η_h is initialized as $\sqrt{n_2/H}$ and is fixed during fine-tuning. Both the Gaussian and Epanechnikov kernel are viable alternatives to softmax attention.

LoRA Fine-Tuning. Low-Rank Adaptation (LoRA) [15] is a parameter-efficient fine-tuning method that keeps the set of pre-trained weights fixed and learns only a low-rank update, which is used to reduce the computational overhead in FHE-based privacy-preserving fine-tuning [28]. For a linear layer with weight matrix $W \in \mathbb{R}^{n_2 \times n_2'}$, LoRA parameterizes the updatable matrices as

$$\Delta W = AB \in \mathbb{R}^{n_2 \times n_2'}, \qquad A \in \mathbb{R}^{n_2 \times n_{\text{LoRA}}}, \quad B \in \mathbb{R}^{n_{\text{LoRA}} \times n_2'}, \quad n_{\text{LoRA}} \ll \min(n_2, n_2').$$

Only the low-rank factors A and B are trained, while the backbone weights W are frozen.

3 Our Method

To address the computational bottleneck in FHE-based transformer fine-tuning, we focus on designing efficient FHE evaluation methods tailored to fine-tuning workloads, which involves: (1) a flexible batching strategy for encrypting input data, enabling support for variable batch sizes in transformer fine-tuning. (2) efficient matrix operations to reduce the computational complexity and decrease the number of bootstrapping operations in the pipeline, and (3) flexible interleaved column packing for vector-wise non-linear functions to avoid "internal rotation" and functional bootstrapping for element-wise non-linear functions to reduce computational cost.

Problem Statement and Threat Model. We consider a server–client computing model under a privacy-preserving threat setting. The server hosts a language model while the client holds private data and the FHE secret key. The embedding layer is assumed to be known to the client and is computed locally on the client side. During fine-tuning, the client first computes the token embeddings and

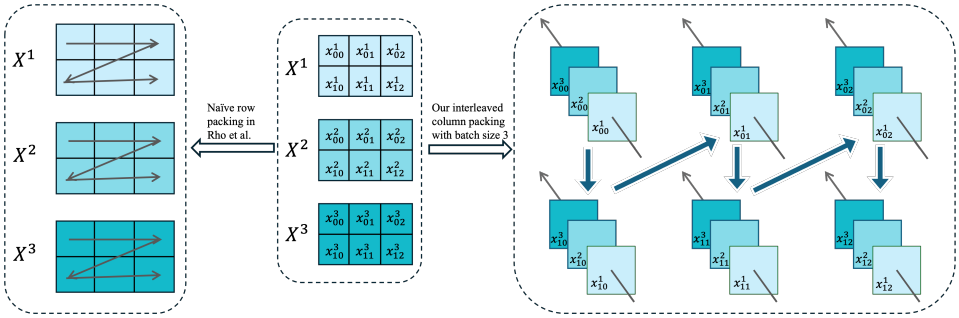

Figure 1: Toy example of comparison between row packing in Rho et al. [28] and our flexible interleaved column packing. Here $n_1 = 2, n_2 = 3, b = 3$. If $N = 6$, then the 3-flexible interleaved column packing divides $\{X^1, X^2, X^3\}$ into 3 ciphertexts encrypting: $(x^1_{00}, x^2_{00}, x^3_{00}, x^1_{10}, x^2_{10}, x^3_{10}), (x^1_{01}, x^2_{01}, x^3_{01}, x^1_{11}, x^2_{11}, x^3_{11}), (x^1_{02}, x^2_{02}, x^3_{02}, x^1_{12}, x^2_{12}, x^3_{12})$.

encrypts them before sending the encrypted inputs to the server. The server then performs the forward pass and evaluates the loss function on encrypted data. Based on this process, encrypted weight updates are obtained and used to update the transformer parameters in the encrypted form. During inference, the server executes the forward pass on encrypted inputs and returns encrypted predictions to the client, who decrypts them to obtain the final results. We assume an *honest-but-curious* server: it honestly follows the protocol but may attempt to infer information about the client's private data from all the encrypted computations.

3.1 A Flexible Column-Wise Batch Packing Strategy

In transformer fine-tuning, inputs are processed in batches, where the batch size is a tunable hyperparameter and typically ranges from 4 to 64. So that a practically and effective packing solution to input data should support flexible batch packing, to better adapt to varying batch sizes and improve slot utilization within ciphertexts, enhancing overall performance. For example, when the batch size is 16, flexible packing can significantly reduce slot underutilization compared to fixed-size schemes, where up to 50% of slots in NEXUS and 93% in MOAI and ARION may otherwise remain unused.

However, the only state-of-the-art work for privacy-preserving transformer finetuing by Rho et al. [28] does not support true batching at all. Instead, it adopts a row-packing strategy in which each input sample is encoded independently into ciphertexts, with matrix elements arranged row-wise, and can only "simulate" batching by assigning one sample per GPU across eight GPUs. Thought works on privacy-preserving transformer inference, NEXUS [36], MOAI [38] and ARION [32] explored reducing amortized computational overhead via batching, their batch sizes are fixed (e.g., 32 in NEXUS and 256 in MOAI and ARION for BERT-base).

Therefore, we propose a *Flexible Column-Wise Batch Packing Strategy*, which supports flexible batch sizes within a unified FHE framework, maximizes batching efficiency to reduce amortized cost, and avoids slot under-utilization in ciphertexts. Our packing strategy packs columns across multiple inputs into ciphertexts, with elements from different inputs interleaved across slots. More precisely, a batch of inputs are organized via *Flexible Interleaved Column Packing* and encrypted.

In Figure 1, we compare the row-wise packing in [28] with our flexible interleaved column-wise packing. Compared to row packing, our method avoids "internal rotations" within each row when computing layers in form of $\mathrm{Enc}(X)W$. Furthermore, during the evaluation of transformer operations, to avoid expensive format conversion and minimize encrypted matrix transpositions, our framework further incorporates *flexible interleaved diagonal packing*, enabling efficient evaluation across different types of computations.

The description of our *Flexible Interleaved Packing* is given in Definition 3.1, and a more detailed definition is in Definition B.1 and Definition B.2 from Appendix B.

Definition 3.1 ((informal) Flexible interleaved column/diagonal packing)**.** *Let $\{X^i\}_{i\in[b]}$ be a batch of matrices with $X^i \in \mathbb{R}^{n_1\times n_2}$ for all $i \in [b]$. Given row index r and column index c, we first pack all the (r, c)-th element from each X^i, denoted as $\{X^i[r][c]\}_{i\in[b]}$. We say we pack $\{X^i\}_{i\in[b]}$ in*

*b-**interleaved column packing** if we concatenate sets $\{X^i[r][c]\}_{i\in[b]}$ in column-major:*

$$\{X^i[0][0]\}_{i\in[b]}\Big\|\{X^i[1][0]\}_{i\in[b]}\Big\|...\Big\|\{X^i[n_1-1][0]\}_{i\in[b]}\Big\|\{X^i[0][1]\}_{i\in[b]}\Big\|...$$

*We say we pack $\{X^i\}_{i\in[b]}$ in b-**interleaved (upper) diagonal packing** if X^i is square and we concatenate sets $\{X^i[r][c]\}_{i\in[b]}$ in (upper) diagonal-major:*

$$\{X^i[0][0]\}_{i\in[b]}\Big\|\{X^i[1][1]\}_{i\in[b]}\Big\|...\Big\|\{X^i[n_1-1][n_1-1]\}_{i\in[b]}\Big\|\{X^i[0][1]\}_{i\in[b]}\Big\|...$$

3.2 Matrix Operations in Flexible Interleaved Packing

In this section, we describe our efficient matrix operations, mainly including 4 types: (1) Ciphertext-plaintext matrix multiplication, which is used to multiply the input with the plaintext weights like W_Q, W_K, W_V and weight matrices in feedforward block; (2) Ciphertext-ciphertext matrix multiplication for LoRA; (3) Ciphertext-ciphertext matrix operation for kernel attention, i.e., the $l2$-norm of Q and K, and $\times V$; (4) Ciphertext-ciphertext matrix multiplication for classification head. These matrix operations are also employed in the backward pass. Further details are provided in Appendix C.2.

Ciphertext-Plaintext Matrix Multiplication (CPMM). The r-th column in X^iW is a linear combination of all the columns from X^i, with weights being the r-th column of W. Therefore, the CPMM, i.e., the linear combination of columns contains two phases:
(1) Multiplication: With properly chosen elements from plaintext matrix W, we can construct plaintext vectors of size N for the weight multiplications. This step costs 1 multiplicative level.
(2) Accumulation: The results from phase (1) are accumulated to form partial weighted sums of columns, and finally are combined using FHE rotations to produce ciphertexts encrypting columns of X^iW. This step does not costs multiplicative level. The detailed algorithm is elaborated in Algorithm 1 and Algorithm 2 from Appendix B. Compare with the CPMM in [28], ours with b-interleaved column packing requires about $213\times$ fewer FHE rotations for the CPMM in attention block, when evaluating $b = 16$ inputs. In addition, our CPMM only costs 1 multiplicative level, while [28] costs 3.

Ciphertext-Ciphertext Matrix Multiplication for LoRA. Compute $(XA)B$ instead of $X(AB)$ can reduce the FHE operation complexity from $O(n_2^2)$ to $O(n_2 n_{\text{LoRA}})$ [28]. We use LoRA-friendly packing for $A \in \mathbb{R}^{n_2\times n_{\text{LoRA}}}$ and $B \in \mathbb{R}^{n_{\text{LoRA}}\times n_2}$, in Algorithm 3 and 7 respectively. Informally, let $\text{Enc}(X)$ contains $d := \lceil \frac{bn_1n_2}{N} \rceil$ ciphertexts, we use $2d$ ciphertexts to pack A or B, so that either $\text{Enc}(X)\,\text{Enc}(A)$ or $(\text{Enc}(X)\,\text{Enc}(A))\,\text{Enc}(B)$ are simplified to one round of ciphertext-ciphertext (ctct) multiplication and one round of accumulation. When evaluating $b = 16$ inputs, ours requires $21\times$ fewer key-switching operations in the attention block compared to [28]. The detailed algorithm is elaborated in Algorithm 5 and Algorithm 8 from Appendix B. Ours only costs 2 multiplicative level, while the one in [28] costs 3.

Ciphertext-Ciphertext Matrix Operations for Kernel Attention. We compute matrix $S_h \in \mathbb{R}^{n_1\times n_1}$ s.t. $S_h[i][j] := \|Q_h[i][:] - K_h[j][:]\|_2^2$. [28] computes $S_h[i][j]$ by $\|Q_h[i][:]\|_2^2 + \|K_h[j][:]\|_2^2 - Q_h[i][:](K_h[j][:])^\intercal$. However, this step requires several matrix transpositions and consumes too many multiplicative levels. We propose a novel method to avoid matrix transpositions. In addition, our novel method generates S_h in diagonal-major packing, making it possible to apply the optimization from [32] to significantly reduce the number of FHE rotations. Our method is based on:
(1) Compute $\{Q_h[i][:] - K_h[i][:]\}_{i\in[n_1]}$ by $Q_h - K_h$. Then the l_2-norm can be obtained by multiplication, FHE rotation and summations. Observe that the result is the main diagonal of S_h.
(2) Rotating *each column* of K_h by d steps, we can compute $\{Q_h[i][:] - K_h[(i+d)\%n_1][:]\}_{i\in[n_1]}$. Then we can obtain the d-th (upper) diagonal of S_h.
We integrate the optimizations from [32] and elaborate the detailed algorithms for S_h in Algorithm 11 and for the multiplication with V_h in Algorithm 12. With all these optimizations, we use total 6 multiplicative levels to compute S_h and $\sigma(S_h)V_h$, while in [28] it costs 12. Our algorithm has $1.40\times$ reduction in the number of key-switching compared to [28].

Ciphertext-Ciphertext Vector-Matrix Multiplication for Classification Head. To optimize the classification head within our framework, we have designed a pipeline that efficiently handles the

Table 2: Comparisons with [28]. Numbers in the table are based on evaluating all kinds of matrix operations (CPMM, CCMM for LoRA, CCMM for kernel, CCMM for $\times V$, and CCMM in classification head) in the forward pass of a single BERT layer with kernel attention on a batch of inputs (batch size b). The concrete parameters are as follows: $X^i \in \mathbb{R}^{128\times 768}, i \in [16]$, $W_Q, W_K, W_V \in R^{768\times 768}$, $A \in \mathbb{R}^{768\times n_{\mathrm{LoRA}}}$, $B \in \mathbb{R}^{n_{\mathrm{LoRA}}\times 768}$, $W_{\mathrm{c2}} \in \mathbb{R}^{768\times 2}$. We have: number of heads $H = 12$, batch size $b = 16$, and $n_{\mathrm{LoRA}} = 2$, which are consistently used in both our experiments and those of [28].

Transformer operations	Matrix operations	Schemes	#(Rot^{HE})	#(Relinearization^{HE})	Total #(KeySwitch^{HE})	Multiplicative level used
X^iW_*, $i \in [b]$, $* = Q, K$, and V	CPMM	Ours	480	0	480 (213× fewer)	1 (2 levels fewer)
		[28]	102480	0	102480	3
$X^iA_*B_*$, $i \in [b]$, $* = Q, K$, and V	CCMM-LoRA	Ours	24	78	102 (21× fewer)	2 (1 level fewer)
		[28]	2016	192	2208	3
S_h, $i \in [b], h \in [H]$	CCMM-kernel	Ours	21912	1536	23448	3 (4 levels fewer)
		[28]	19776	384	20160	7
$\sigma(S_h^i)V_h^i$, $i \in [b], h \in [H]$	CCMM-V	Ours	7320	192	7512 (3.1× fewer)	3 (2 levels fewer)
		[28]	23232	192	23424	5
$X^i_{[CLS]}W_{c1}, i \in [b]$	CCMM-classification-1[6]	Ours	361	6	367 (1.4× fewer)	2 (1 level fewer)
		[28]	512	32	544	3
$\tanh(\cdot)W_{c2}, i \in [b]$	CCMM-classification-2	Ours	10	1	11 (17.4× fewer)	1
		[28]	176	16	192	1

transition from transformer layers to final predictions. We utilize special packing strategies for weight matrices in Algorithms 13 and 14 respectively, and a "repeated packing" format for the token in Algorithm 15. This approach ensures that the two multiplication steps cost minimal key-switching operations, resulting in a $1.4\times$ to $17.4\times$ reduction in the number of key-switching compared to [28]. Detailed formal descriptions of these packing algorithms and the corresponding homomorphic evaluation procedures are provided in Appendix B.4.

3.2.1 Analysis of Computation Overheads

Key-switching, which is involved in HE rotations and relinearizations, is recognized as the most expensive operation in HE-based matrix operations such as matrix multiplications and kernel evaluations. Therefore, existing works [25, 27, 38, 32] usually use the number of key-switching as the benchmark in comparing different matrix operations.

Table 2 presents a comprehensive comparison between our algorithms and [28] in terms of number of key-switching operations across all major matrix operations. Aggregated over all matrix operations in fine-tuning a 2-layer BERT-style model, our approach reduces key-switching operations by up to $30\times$ while reducing the total multiplicative depth by approximately half.

In kernel computation $\{S_h[u][v] := ||Q_h[u][:] - K_h[v][:]||_2^2\}_{u,v\in[n_1]}$, our method incurs approximately $1.16\times$ more key switching operations. This trade-off is intentional for two reasons: (1) it significantly reduces the multiplicative depth from 7 to 3, which is critical for decreasing the frequency of bootstrapping; and (2) we jointly optimize the kernel computation and the subsequent $\times V_h$ operation using a double BSGS technique. While this slightly increases the number of rotations in the kernel computation, it leads to a larger reduction in rotations in the $\times V_h$ step.

Overall, the results demonstrate that our algorithms provide substantial efficiency gains across the entire transformer pipeline by reducing both the number of expensive homomorphic operations and required multiplicative levels.

3.3 Non-Linear Function Evaluation

We split non-linear functions into two broad categories: element-wise functions, like $\exp$, ReLU, GELU, and Tanh; and vector-wise functions, like $\mathrm{LayerNorm}$ in transformer layers and $\mathrm{SoftMax}$ in the classification head.

For vector-wise functions, we organize inputs using a flexible interleaved column packing. Compared to the row-packing strategy in [28], (1) column-packing avoids "internal rotations" within rows when computing reductions such as $\sum_{i\in[n_1]} X[\cdot][i]$, thereby reducing the number of ciphertext rotations by

[6] In [28], W_{c1} is replaced with two dense block of LoRA-style shapes with rank=32, thus they can use CCMM for LoRA in this step. While in our scheme, we do not modify the model architecture and use $W_{c1} \in R^{768\times 768}$ as the original model.

half (a toy example is provided in Appendix E.1), and (2) interleaved batching further reduce the amortized number of ciphertext rotations to $\log(N/bn_1)/b$, where N is the slot count and b is the batch size. Following [25, 38], we approximate the inverse and inverse square root in $\mathrm{LayerNorm}$ and $\mathrm{SoftMax}$ using Goldschmidt division algorithm and Newton iteration. More details of the algorithms used are in Appendix E.1.

Element-wise functions are evaluated during Functional Bootstrapping [33], combining ciphertext refresh and activation function evaluation into a single operation with negligible additional cost. Compared to the conventional "non-linear evaluation + bootstrapping" pipeline used in [28, 38, 25, 27, 36], functional bootstrapping eliminates the need for a separate activation evaluation step, thereby improving the overall workflow. More details can be found in Appendix E.2. In Table 5 from Section 4.1, we show that our algorithm achieves $4.3\times$ speedup over [28] in evaluating the activation functions in the feedforward block.

4 Experimental Results

Model and Datasets. Experiments are conducted on 3 GLUE tasks [31], MRPC, RTE and STSB. Following [28], we use a compact BERT backbone with 2 transformer layers, hidden size is 768, feedforward size is 3072, and has 12 attention heads [7]. We refer the reader to Appendix G.1 for a more detailed justification behind our choices. We use a batch size of 16 for all tasks.

We compare softmax attention (SM), Gaussian kernel attention (GK), and Epanechnikov kernel attention (EK). The SM baseline is evaluated from the original pretrained checkpoint, and the GK baseline is taken directly from the published results of [28]. More details are in Appendix G.1.

FHE Parameters and Implementations. We set the degree of the polynomial ring as 2^{16}, which supports $N = 2^{15}$ plaintext slots in a ciphertext. The ciphertext modulus Q is chosen to ensure a 128-bit security level [1], and support 12 multiplicative levels after bootstrapping.[8] Our scheme is initially implemented using the Microsoft SEAL library [30] and subsequently migrated to the PhantomFHE library. [9] All of our experiments used a single NVIDIA H200 GPU. Our code is available. [10]

Baselines. The implementation in [28] is based on the proprietary (non-open-sourced) HEaaN library and evaluated on an NVIDIA GeForce RTX 4090 GPU, whereas our implementation is built on PhantomFHE and runs on an NVIDIA H200 GPU. In Table 17 in Appendix G.2, we benchmark the primitive operations of the CKKS scheme in both libraries under same parameters. To ensure fair comparison, we reproduce their evaluation algorithms using the PhantomFHE library and measure runtime on the same hardware platform.

4.1 Microbenchmarks

We first compare against Rho et al. [28] to demonstrate the speedups achieved by our method on the following aspects. All experiments are conducted under the same parameter settings and hardware, using a single-threaded configuration.

Matrix Operations. Figure 2 demonstrates the runtime performance of CPMM, CCMM for LoRA, CCMM for kernel, and CCMM for value matrix over a batch of inputs. Our scheme consistently outperforms Rho et al. [28] across different output ciphertext levels in all four types of matrix operations. In particular, our scheme achieves $32\times$ to $63\times$ speedup in CPMM, $6.6\times$ to $14.5\times$ speedup in CCMM for LoRA, $1.5\times$ to $2.2\times$ speedup in CCMM for kernel, and $2\times$ to $3.8\times$ speedup in CCMM for value matrix.

Functional Bootstrapping. In Table 3, we compare the time cost of the functional bootstrapping [33] and the traditional "evaluation + bootstrapping" flow used in [28] in evaluating the activation function in the feedforward block from [28], with input/output matrix in $\mathbb{R}^{128\times 1536}$. To demonstrate the

[7] `google/bert_uncased_L-2_H-768_A-12` [9]

[8] Our choice of ciphertext modulus and multiplicative depth is consistent with the baseline in [28]. Specifically, [28] reports 9 available levels before bootstrapping and a minimum requirement of 3 levels to perform bootstrapping, which matches the requirement for functional bootstrapping in our scheme. Their open-source implementation also confirms that 12 levels are restored after bootstrapping.

[9] https://github.com/encryptorion-lab/phantom-fhe

[10] https://github.com/LWEpsilon-FHE/FELID

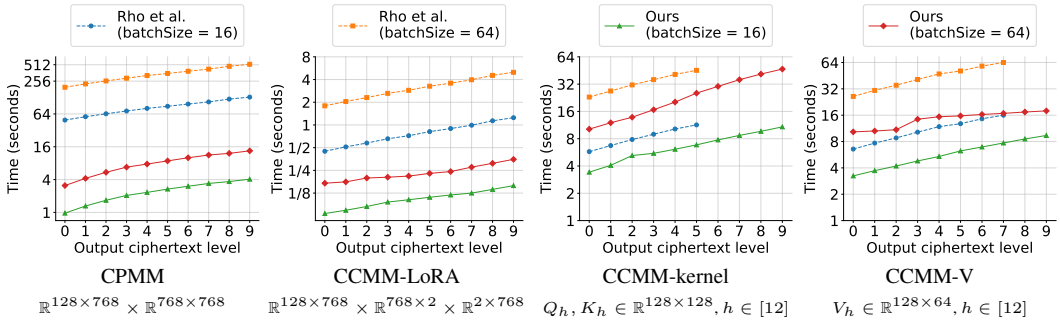

Figure 2: Runtime results of homomorphic matrix operations across different output ciphertext levels.

Table 3: The comparison between CKKS functional bootstrapping and the traditional "evaluation + bootstrapping" flow in [28]. The comparison is set to the activation in feedforward block. The input level and the output level are set to the same in the two settings.

	#(input ct)	Input lv.	#(BTS)	#(FuncBTS)	#(Output ct)	Output lv.	(Worst prec., Avg prec.)	Time(s)
FELID (ours)	6	4	0	6	6	12	$(2^{-13.54}, 2^{-13.97})$[33]	4.02 (4.33 × faster)
Rho et al. [28]*	6	4	18	0	6	12	$(2^{-10}, 2^{-16.4})$[28]	17.4

*For fairness, we reproduce their result under the same settings as ours using PhantomFHE.

improvement from functional bootstrapping, both experiments use the same row packing method as in [28]. As a result, our algorithm achieves $4.3\times$ speedup over [28].

Runtime and Bootstrapping Counts. We count the number of bootstrapping of our scheme and [28] in Table 4. Our method reduces the number of (functional) bootstrapping operations by $2.5\times$, leading to a $2.69\times$ reduction in total bootstrapping runtime compared to Rho et al. [28].

Table 4: Runtime (seconds) and total number of bootstrapping and functional bootstrapping operations in our scheme and in [28] in LoRA fine-tuning, over a batch of input with batch size = 16.

Scheme	Forward pass		Backward pass + Optimizer		Total	
	Number	Time (s)	Number	Time (s)	Number	Time (s)
FELID (ours)	683 (2.18 × fewer)	295.05	236 (3.44 × fewer)	111.95	919 (2.5 × fewer)	407 (2.69 × faster)
Rho et al. [28] *	1488	708	812	386	2300	1094

*For fairness, we reproduce their result under the same settings as ours using PhantomFHE.

4.2 Fine-Tuning for 2-Layer BERT-Style Model

Fine-Tuning Time. In Table 5, we report the execution time of each component, with a detailed breakdown of transformer operations provided in Appendix G.3. Benefiting from (1) optimized packing and more efficient matrix operations (Figure 2); (2) eliminating separate point-wise activation evaluations via functional bootstrapping (Table 3); and (3) reducing the number of bootstrapping invocations (Table 4), our method achieves consistent improvements across all components, reducing the fine-tuning time per batch from 5238 seconds to 650.6 seconds ($8\times$ speedup).

Downstream Task Performance. We evaluate both full and LoRA fine-tuning in plaintext and LoRA fine-tuning in ciphertext. Under consistent plaintext settings, kernel-based attention preserves competitive accuracy compared to softmax, demonstrating that both Gaussian and Epanechnikov kernels are viable alternatives to softmax attention. The ciphertext results in Table 6 indicate that the additional approximation error introduced by the FHE evaluation has a minimal impact on fine-tuning.

5 Conclusions

In this work, we propose FELID, a flexible and efficient fully homomorphic encryption framework for non-interactive transformer fine-tuning. FELID introduces (i) a novel packing strategy that enables flexible batching, (ii) operation-specific matrix evaluation algorithms tailored for kernel-based transformers, and (iii) the use of functional bootstrapping to eliminate separate evaluation of element-wise activations. As a result, FELID reduces ciphertext rotations and relinearizations by $30\times$ and FHE bootstrapping invocations by $2.5\times$ across the entire computation flow compared to the

Table 5: Fine-tuning time (seconds) over a batch of input with batch size = 16.

Scheme	Forward pass			Backward pass			Optimizer	Total	Amortized
	Matrix	Non-linear	(Func)Boot	Matrix	Non-linear	(Func)Boot			
FELID (ours)	67.79	9.56	295.05	164.04	1.91	92.45	19.9	650.6	40.66 ($8\times$ faster)
Rho et al. [28]*	1429	291	708	2377	47	355	31	5238	327.38

*For fairness, we reproduce their result under the same settings as ours using PhantomFHE.

Table 6: Plaintext and ciphertext GLUE results. Results marked with [28] are taken from the published Gaussian-kernel transformer; "ours" denotes models trained using our continued-pretraining and distillation pipeline. To ensure a fair comparison, we fine-tune the models for 5 epochs as per [28].

	Plaintext Full fine-tuning				Plaintext LoRA fine-tuning			Ciphertext LoRA fine-tuning		
Task (Metric)	SM [28]	GK [28]	GK (ours)	EK (ours)	GK [28]	GK (ours)	EK (ours)	GK [28]	GK (ours)	EK (ours)
MRPC (F1 ↑)	0.8328	0.8431	0.8349	0.8381	0.8258	0.8298	0.8315	0.8147	0.8255	0.8323
RTE (Accuracy ↑)	0.5884	0.6101	0.5957	0.6173	0.5776	0.6029	0.6065	0.5993	0.5957	0.5921
STS-B (Pearson ↑)	0.8415	0.8215	0.8311	0.8230	0.8107	0.8184	0.8274	0.7997	0.8155	0.7824
Average	0.7542	0.7582	0.7539	0.7595	0.7380	0.7504	0.7551	0.7379	0.7456	0.7356

state-of-the-art [28]. Fine-tuning of a 2-layer BERT-style model with a batch size of 16 is completed in 650 seconds, achieving an $8\times$ speedup over prior work.

Acknowledgments

This work was conducted through the joint research laboratory of Ant International and Nanyang Technological University, with funding support from Ant International. The laboratory was established under a five-year partnership to advance digital trust and privacy-enhancing technologies (PETs). This research is supported by the National Research Foundation, Singapore and Infocomm Media Development Authority under its Trust Tech Funding Initiative. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore and Infocomm Media Development Authority.

References

[1] Martin Albrecht, Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Shai Halevi, Jeffrey Hoffstein, Kim Laine, Kristin Lauter, et al. Homomorphic encryption standard. In *Protecting privacy through homomorphic encryption*, pages 31–62. Springer, 2022.

[2] Andreea Alexandru, Andrey Kim, and Yuriy Polyakov. General functional bootstrapping using ckks. In *Annual International Cryptology Conference*, pages 304–337. Springer, 2025.

[3] Alain Berlinet and Christine Thomas-Agnan. *Reproducing kernel Hilbert spaces in probability and statistics*. Springer Science & Business Media, 2011.

[4] Jingwei Chen, Linhan Yang, Chen Yang, Shuai Wang, Rui Li, Weijie Miao, Wenyuan Wu, Li Yang, Kang Wu, and Lizhong Dai. Secure transformer-based neural network inference for protein sequence classification. Cryptology ePrint Archive, Paper 2024/1851, 2024.

[5] Tianyu Chen, Hangbo Bao, Shaohan Huang, Li Dong, Binxing Jiao, Daxin Jiang, Haoyi Zhou, Jianxin Li, and Furu Wei. THE-X: Privacy-preserving transformer inference with homomorphic encryption. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio, editors, *Findings of the Association for Computational Linguistics: ACL 2022*, pages 3510–3520, Dublin, Ireland, May 2022. Association for Computational Linguistics.

[6] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. Bootstrapping for approximate homomorphic encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 360–384. Springer, 2018.

[7] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 409–437. Springer, 2017.

[8] Wonhee Cho, Guillaume Hanrot, Taeseong Kim, Minje Park, and Damien Stehlé. Fast and accurate homomorphic softmax evaluation. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, pages 4391–4404, 2024.

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pages 4171–4186, 2019.

[10] Ye Dong, Wen-jie Lu, Yancheng Zheng, Haoqi Wu, Derun Zhao, Jin Tan, Zhicong Huang, Cheng Hong, Tao Wei, Wen-Guang Chen, et al. Puma: Secure inference of llama-7b in five minutes. *Security and Safety*, 4:2025014, 2025.

[11] Seyda Nur Guzelhan, Lohit Daksha, Carlos Agulló Domingo, Gilbert Jonatan, John Kim, Jose L. Abellan, David Kaeli, and Ajay Joshi. ELLMo: Packing- and depth-aware encrypted transformer inference. Cryptology ePrint Archive, Paper 2026/198, 2026.

[12] Shai Halevi and Victor Shoup. Algorithms in helib. In *Annual Cryptology Conference*, pages 554–571. Springer, 2014.

[13] Kyoohyung Han, Seungwan Hong, Jung Hee Cheon, and Daejun Park. Logistic regression on homomorphic encrypted data at scale. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 9466–9471, 2019.

[14] Benjamin Hoover, Zhaoyang Shi, Krishna Balasubramanian, Dmitry Krotov, and Parikshit Ram. Dense associative memory with epanechnikov energy. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2026.

[15] Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

[16] Xiaoqian Jiang, Miran Kim, Kristin Lauter, and Yongsoo Song. Secure outsourced matrix computation and application to neural networks. In *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, pages 1209–1222, 2018.

[17] Eunsang Lee, Joon-Woo Lee, Junghyun Lee, Young-Sik Kim, Yongjune Kim, Jong-Seon No, and Woosuk Choi. Low-complexity deep convolutional neural networks on fully homomorphic encryption using multiplexed parallel convolutions. In *International Conference on Machine Learning*, pages 12403–12422, 2022.

[18] Garam Lee, Minsoo Kim, Jai Hyun Park, Seung-won Hwang, and Jung Hee Cheon. Privacy-preserving text classification on bert embeddings with homomorphic encryption. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3169–3175, 2022.

[19] Seewoo Lee, Garam Lee, Jung Woo Kim, Junbum Shin, and Mun-Kyu Lee. Hetal: Efficient privacy-preserving transfer learning with homomorphic encryption. In *International conference on machine learning*, pages 19010–19035, 2023.

[20] Dacheng Li, Hongyi Wang, Rulin Shao, Han Guo, Eric Xing, and Hao Zhang. Mpcformer: Fast, per- formant and private transformer inference with mpc. In *The Eleventh International Conference on Learning Representations (ICLR)*, 2023.

[21] Lawrence Lim, Vikas Kalagi, Divyakant Agrawal, and Amr El Abbadi. Tricycle: Private transformer inference with tricyclic encodings. Cryptology ePrint Archive, Paper 2025/1200, 2025.

[22] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[23] Wen-jie Lu, Zhicong Huang, Zhen Gu, Jingyu Li, Jian Liu, Cheng Hong, Kui Ren, Tao Wei, and WenGuang Chen. Bumblebee: Secure two-party inference framework for large transformers. *Cryptology ePrint Archive*, 2023.

[24] Wen-jie Lu, Zhicong Huang, Cheng Hong, Yiping Ma, and Hunter Qu. Pegasus: Bridging polynomial and non-polynomial evaluations in homomorphic encryption. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1057–1073. IEEE, 2021.

[25] Jungho Moon, Dongwoo Yoo, Xiaoqian Jiang, and Miran Kim. Thor: Secure transformer inference with homomorphic encryption. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security*, pages 3765–3779, 2025.

[26] Qi Pang, Jinhao Zhu, Helen Möllering, Wenting Zheng, and Thomas Schneider. Bolt: Privacy-preserving, accurate and efficient inference for transformers. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 4753–4771. IEEE, 2024.

[27] Dongjin Park, Eunsang Lee, and Joon-Woo Lee. Powerformer: Efficient and high-accuracy privacy-preserving language model with homomorphic encryption. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11090–11111, 2025.

[28] Donghwan Rho, Taeseong Kim, Minje Park, Jung Woo Kim, Hyunsik Chae, Ernest K Ryu, and Jung Hee Cheon. Encryption-friendly llm architecture. In *13th International Conference on Learning Representations, ICLR 2025*, pages 67315–67341. International Conference on Learning Representations, ICLR, 2025.

[29] Saul Santos, Nuno Gonçalves, Daniel C McNamee, Marcos Treviso, and André FT Martins. Sparse attention as compact kernel regression. *arXiv preprint arXiv:2601.22766*, 2026.

[30] Microsoft SEAL (release 4.1). `https://github.com/Microsoft/SEAL`, January 2023. Microsoft Research, Redmond, WA.

[31] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 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*, pages 353–355, 2018.

[32] Linhan Yang, Jingwei Chen, Wangchen Dai, Shuai Wang, Wenyuan Wu, and Yong Feng. ARION: Attention-optimized transformer inference on encrypted data. Cryptology ePrint Archive, Paper 2025/2271, 2025.

[33] Zhaomin Yang, Chao Niu, Benqiang Wei, Zhicong Huang, Cheng Hong, and Tao Wei. RBOOT: Accelerating homomorphic neural network inference by fusing ReLU within bootstrapping. Cryptology ePrint Archive, Paper 2025/1534, 2025.

[34] Andrew C Yao. Protocols for secure computations. In *23rd annual symposium on foundations of computer science (sfcs 1982)*, pages 160–164. IEEE, 1982.

[35] Wenxuan Zeng, Ye Dong, Jinjin Zhou, Junming Ma, Jin Tan, Runsheng Wang, and Meng Li. Mpcache: Mpc-friendly kv cache eviction for efficient private large language model inference. *arXiv preprint arXiv:2501.06807*, 2025.

[36] Jiawen Zhang, Xinpeng Yang, Lipeng He, Kejia Chen, Wen-jie Lu, Yinghao Wang, Xiaoyang Hou, Jian Liu, Kui Ren, and Xiaohu Yang. Secure transformer inference made non-interactive. In *NDSS*, 2025.

[37] Linru Zhang, Xiangning Wang, Xianhui Lu, Huaxiong Wang, and Kwok Yan Lam. LEAF: A low-latency evaluation architecture for feedforward block in privacy-preserving transformer inference. Cryptology ePrint Archive, Paper 2025/956, 2025.

[38] Linru Zhang, Xiangning Wang, Jun Jie Sim, Zhicong Huang, Jiahao Zhong, Huaxiong Wang, Pu Duan, and Kwok-Yan Lam. MOAI: Module-optimizing architecture for non-interactive secure transformer inference. In *The Fourteenth International Conference on Learning Representations*, 2026.

[39] Yushun Zhang, Congliang Chen, Tian Ding, Ziniu Li, Ruoyu Sun, and Zhi-Quan Luo. Why transformers need adam: A hessian perspective. *Advances in neural information processing systems*, 37:131786–131823, 2024.

[40] Itamar Zimerman, Moran Baruch, Nir Drucker, Gilad Ezov, Omri Soceanu, and Lior Wolf. Converting transformers to polynomial form for secure inference over homomorphic encryption. In *Forty-first International Conference on Machine Learning*, 2024.

A Extended preliminaries

A.1 More detailed CKKS scheme

The CKKS scheme [7] also includes the following building blocks:

- Keygen: it generates the secret key sk.
- $\mathrm{Enc}_{\mathrm{sk}}(\mathbf{z})$: for $\mathbf{z} \in \mathbb{C}^N$, it generates the ciphertext of complex vector $\mathbf{z}$ using sk and it can only be decrypted with sk. When sk is clear from the context, $\mathrm{Enc}(\mathbf{z})$ is used.
- $\mathrm{Dec}_{\mathrm{sk}}(\mathrm{Enc}_{\mathrm{sk}}(\mathbf{z}))$: it decrypts the ciphertext encrypting $\mathbf{z}$ with the secret key sk.

Encoding and decoding. The CKKS scheme does not directly encrypt vector in $\mathbb{C}^N$. It first encodes a complex vector into a polynomial with integer coefficients through a sequence of *embedding* and *scaling* operations. Let $\mathbf{z} \in \mathbb{C}^N$ denote the input vector. CKKS first applies an *inverse canonical embedding* that maps $\mathbf{z}$ into a polynomial $z(X) \in \mathbb{C}[X]/(X^{2N}+1)$, where N is a power of two. The complex coefficients of $m(X)$ are scaled by a factor $\Delta > 0$ and rounded to obtain an integer-coefficient polynomial:

$$\tilde{z}(X) = \lfloor \Delta \cdot m(X) \rceil \in \mathbb{Z}[X]/(X^{2N}+1).$$

This process is referred to as $\tilde{z}(X) := \mathrm{Encode}(\boldsymbol{z}, \Delta)$, and it allows representation of complex numbers (of $\mathbf{z}$) with finite precision.

Encryption is then performed over the ring $\mathbb{Z}_Q[X]/(X^{2N}+1)$ for a large modulus Q. Given the secret key $\mathrm{sk} = s(X)$, a ciphertext is generated as a pair of polynomials:

$$\mathrm{Enc}_{\mathrm{sk}}(\tilde{z}(X)) := (c_0(X), c_1(X)) := (\tilde{z}(X) + e(X) - a(X)s(X),\ a(X)) \in \left(\mathbb{Z}_Q[X]/(X^{2N}+1)\right)^2,$$

where $a(X)$ is sampled uniformly at random from $\mathbb{Z}_Q[X]/(X^{2N}+1)$, and $e(X)$ are small noise polynomials sampled from a pre-defined discrete Gaussian distribution. Therefore we can write $\mathrm{Enc}_{\mathrm{sk}}(\mathrm{Encode}(\mathbf{z}, \Delta))$ in to $\mathrm{Enc}_{\mathrm{sk}}(\mathbf{z})$ for short as we do not modify and do not focus on encoding/decoding.

Decryption recovers an approximation of $\tilde{m}(X)$ via:

$$\tilde{m}(X) \approx c_0(X) + c_1(X)s(X) \mod Q,$$

followed by *decoding* (i.e., the inverse of $\mathrm{Encode}(\cdot, \Delta)$, scaling down by Δ and applying the canonical embedding) to obtain an approximate reconstruction of the original vector $\mathbf{z}$.

A.2 Softmax attention transformer architecture

Currently many large language models are based on transformer architecture. Such a model contains multiple layers and each layer has H heads. Let n_1 be the token length of input, n_2 be the hidden dimension, $n_3 = n_2/H$ be the dimension of each head. For simplicity bias vectors are omitted. In each layer, given input matrix X, the model first compute $Q_h = XW_{Q_h}, K_h = XW_{K_h}, V_h = XW_{V_h}$ for each head $h \in [H]$. For models with transformer architecture, like BERT[9], the layer will output

$$X_h := \mathrm{Attention}(Q_h, K_h, V_h) := \mathrm{Softmax}(\frac{Q_h K_h^\intercal}{\sqrt{n_3}})V_h,\ \forall h \in [H]\ .$$

B Matrix multiplication algorithms

Definition B.1 (Flexible interleaved column packing)**.** *Suppose we have b matrices $\{X^i \in \mathbb{R}^{n_1 \times n_2}\}_{i\in[b]}$ be:*

$$X^i = \begin{pmatrix} x^i[0][0] & \cdots & X^i[0][n_2-1] \\ \vdots & \ddots & \vdots \\ x^i[n_1-1][0] & \cdots & X^i[n_1-1][n_2-1] \end{pmatrix}.$$

Let $\{\mathbf{m}^i \in \mathbb{R}^N\}_{i\in[r]}$, $r = \lceil n_2/\lfloor N/(bn_1)\rfloor\rceil$ denote the plaintext vectors obtained by interleaved column packing the matrices $X^0, ..., X^{b-1}$. For simplicity, we assume $(bn_1)|N$ and $N|(bn_1n_2)$. We define:

$$m^i[j] = X^{j\%b}[(\lfloor\frac{j}{b}\rfloor)\%n_1][i\frac{N}{bn_1} + \lfloor\frac{j}{bn_1}\rfloor]$$

*where $i \in [\frac{bn_1n_2}{N}]$, and $j \in [N]$. We say that $\{\mathrm{Enc}(\mathbf{m}^i)\}_{i\in[\frac{bn_1n_2}{N}]}$ is in b-**interleaved column packing**.*

B.1 Ciphertext-plaintext matrix multiplication (CPMM)

Our CPMM takes an encrypted matrix in b-interleaved column packing with and a plaintext matrix, and output an encrypted matrix in b-interleaved column packing. For simplicity, we assume $(bn_1)|N$, $N|(bn_1n_2)$, and $N|(bn_1n_2')$. The details of algorithm is in Algorithm 1.

Optimization with imaginary parts. Making use of imaginary parts of each CKKS slot will further save HE operations, especially in the ct-pt matrix multiplications. We modify Algorithm 1 so that each X^iW in the encrypted output is compactly packed in the complex number field $\mathbb{C}$. $\forall W \in \mathbb{R}^{n_2\times n_2'}$. Let $W_{even} \in \mathbb{R}^{n_2\times(n_2'/2)}$ include all the even-index columns of W (starting from index 0), and let $W_{odd} \in \mathbb{R}^{n_2\times(n_2'/2)}$ include all the odd-index columns of W. Define $W_{\mathbb{C}} := W_{even} + \sqrt{-1}W_{odd} \in \mathbb{C}^{n_2\times(n_2'/2)}$. The details of algorithm is in Algorithm 2.

Rotations in a batch. Number of HE rotations in Algorithm 1 is

$$\#(\text{Rot}^{\text{HE}}) = r_1t_1 + r_2t_2 \approx \frac{bn_1n_2}{N}\cdot\frac{\sqrt{N}}{\sqrt{bn_1}} + \frac{bn_1n_2'}{N}\cdot\frac{\sqrt{N}}{\sqrt{bn_1}} = (n_2+n_2')\sqrt{\frac{bn_1}{N}}.$$

By using imaginary parts, we reduce the number of output ciphertexts by half, so the number of HE rotations in Algorithm 2 is

$$\#(\text{Rot}^{\text{HE}}) = r_1t_1 + \frac{r_2}{2}t_2 \approx (n_2+\frac{n_2'}{2})\sqrt{\frac{bn_1}{N}}.$$

If we set $N = 32768$, $b = 16$, $X^i \in \mathbb{R}^{128\times768}$, and $W \in \mathbb{R}^{768\times768}$, then numbers of HE rotations in Algorithm 1 and Algorithm 2 are 384 and 288 respectively.

Multiplicative level and ct-pt multiplications in a batch. CPMM consumes 1 multiplicative level with $r_2t_2r_1t_1$ ct-pt multiplications in Algorithm 1 and $\frac{r_2t_2r_1t_1}{2}$ ct-pt multiplications in Algorithm 2.

If we set $N = 32768$, $b = 16$, $X^i \in \mathbb{R}^{128\times768}$, and $W \in \mathbb{R}^{768\times768}$, then numbers of HE ct-pt multiplications in Algorithm 1 and Algorithm 2 are 36864 and 18432 respectively.

In summary, the required number of each HE operation for $\{\mathrm{Enc}(X^i)W\}_{i\in[b]}$, where $X^i \in \mathbb{R}^{n_1\times n_2}$, and $W \in \mathbb{R}^{n_2\times n_2'}$, is listed in Table 7.

Remark. In the attention block, using LoRA fine-tuning we can set $W_* = W_{*,frozen} + AB$, where $* = Q, K, V$. $W_{*,frozen}$ are plaintext, so we can set $W_{attn,frozen} = (W_{Q,frozen}\|W_{K,frozen}\|W_{V,frozen})$. By Table 7, applying Algorithm 2 with $W_{attn,frozen}$ and $n_2' = 3n_2$ is much better than applying Algorithm 2 3 times, with $W_{Q,frozen}, W_{K,frozen}, W_{V,frozen}$ respectively.

Algorithm 1 Ciphertext-plaintext matrix multiplication.

Input: Encrypted matrices $\{X^i \in \mathbb{R}^{n_1 \times n_2}\}_{i \in [b]}$ in b-interleaved column packing $\{\mathrm{Enc}(\mathbf{m}^i)\}_{i \in [r_1]}$, $r_1 = \frac{bn_1n_2}{N}$, matrix $W \in \mathbb{R}^{n_2 \times n_2'}$

Output: Encrypted matrices $\{X^iW \in \mathbb{R}^{n_1 \times n_2'}\}_{i \in [b]}$ in b-interleaved column packing $\{\mathrm{Enc}(\hat{\mathbf{m}}^i)\}_{i \in [r_2]}$, $r_2 = \frac{bn_1n_2'}{N}$.

Let $t_1 = \lceil \sqrt{\frac{N}{bn_1}} \rceil$, and $t_2 = \lceil \frac{N}{bn_1t_1} \rceil$.
for i from 0 to $r_1 - 1$ **do**
  for k from 0 to $t_1 - 1$ **do**
    Evaluate $ct_{it_1+k} = \mathrm{Rot}^{\mathrm{HE}}_{kbn_1}(\mathrm{Enc}(\mathbf{m}^i))$.
  end for
end for
for i from 0 to $r_2 - 1$ **do**
  for k_1 from 0 to $t_2 - 1$ **do**
    Let $\hat{ct} = \mathrm{Enc}(\mathbf{0})$. ▷ Initialization of ciphertext
    for j from 0 to $r_1 - 1$ **do**
      for k_2 from 0 to $t_1 - 1$ **do**
        Let $\mathbf{d} = \mathbf{0}$.
        for l_1 from 0 to $\frac{N}{bn_1} - 1$ **do**
          for l_2 from 0 to $n_1 - 1$ **do**
            for l_3 from 0 to $b - 1$ **do**
              Let $\mathrm{Ind}_{row} = \frac{jN}{bn_1} + (k_1t_1 + k_2 + l_1)\%\frac{N}{bn_1}$, and $\mathrm{Ind}_{col} = \frac{iN}{bn_1} + l_1\%\frac{N}{bn_1}$.
              Let $d[l_1bn_1 + l_2b + l_3] = W[\mathrm{Ind}_{row}][\mathrm{Ind}_{col}]$. ▷ Pack weights into vector $\mathbf{d}$
            end for
          end for
        end for
        Evaluate $\hat{ct} = \hat{ct} + \mathrm{PMult}(\mathrm{Rot}_{-k_1t_1bn_1}(\mathbf{d}), ct_{jt_1+k_2k})$. ▷ ct-pt multiplication
      end for
    end for
    if $k_1 = 0$ **then**
      Evaluate $\mathrm{Enc}(\hat{\mathbf{m}}^i) = \mathrm{Rot}^{\mathrm{HE}}_{k_1t_1bn_1}(\hat{ct})$.
    else
      Evaluate $\mathrm{Enc}(\hat{\mathbf{m}}^i) = \mathrm{Enc}(\hat{\mathbf{m}}^i) + \mathrm{Rot}^{\mathrm{HE}}_{k_1t_1bn_1}(\hat{ct})$.
    end if
  end for
end for

Table 7: The number of each HE operation for $\{\mathrm{Enc}(X^i)W\}_{i \in [b]}$ in our scheme

Operations	Rot	ct-ct Mult	(Relinearization)	ct-pt Mult	Level
CPMM-basic	$(n_2 + n_2')\sqrt{\frac{bn_1}{N}}$	0	0	$\frac{bn_1n_2n_2'}{N}$	1
CPMM-optimized	$(n_2 + \frac{n_2'}{2})\sqrt{\frac{bn_1}{N}}$	0	0	$\frac{bn_1n_2n_2'}{2N}$	1

Algorithm 2 Optimized Ciphertext-plaintext matrix multiplication.

Input: Encrypted matrices $\{X^i \in \mathbb{R}^{n_1 \times n_2}\}_{i\in[b]}$ in b-interleaved column packing $\{\mathrm{Enc}(\mathbf{m}^i \in \mathbb{R}^N)\}_{i\in[r_1]}$, $r_1 = \frac{bn_1n_2}{N}$, matrix $W \in \mathbb{R}^{n_2 \times n_2'}$

Output: Encrypted matrices $\{X^iW_{\mathbb{C}} \in \mathbb{C}^{n_1 \times (n_2'/2)}\}_{i\in[b]}$ in b-interleaved column packing $\{\mathrm{Enc}(\hat{\mathbf{m}}^i \in \mathbb{C}^N)\}_{i\in[r_2]}$, $r_2 = \frac{bn_1n_2'}{2N}$.

Let $t_1 = \lceil\sqrt{\frac{N}{bn_1}}\rceil$, and $t_2 = \lceil\frac{N}{bn_1t_1}\rceil$.
for i from 0 to $r_1 - 1$ **do**
  for k from 0 to $t_1 - 1$ **do**
    Evaluate $ct_{it_1+k} = \mathrm{Rot}^{\mathrm{HE}}_{kbn_1}(\mathrm{Enc}(\mathbf{m}^i))$.
  end for
end for
Let $W_{even} \in \mathbb{R}^{n_2 \times (n_2'/2)}$ include all the even-index columns of W. (starting from index 0).
Let $W_{odd} \in \mathbb{R}^{n_2 \times (n_2'/2)}$ include all the odd-index columns of W.
Define $W_{\mathbb{C}} := W_{even} + \sqrt{-1}W_{odd} \in \mathbb{C}^{n_2 \times (n_2'/2)}$. ▷ Packing W in $\mathbb{C}$.
for i from 0 to $r_2 - 1$ **do**
  for k_1 from 0 to $t_2 - 1$ **do**
    Let $\hat{ct} = \mathrm{Enc}(\mathbf{0})$.
    for j from 0 to $r_1 - 1$ **do**
      for k_2 from 0 to $t_1 - 1$ **do**
        Let $\mathbf{d} = \mathbf{0}$.
        for l_1 from 0 to $\frac{N}{bn_1} - 1$ **do**
          for l_2 from 0 to $n_1 - 1$ **do**
            for l_3 from 0 to $b - 1$ **do**
              Let $\mathrm{Ind}_{row} = \frac{jN}{bn_1} + (k_1t_1 + k_2 + l_1)\%\frac{N}{bn_1}$, and $\mathrm{Ind}_{col} = \frac{iN}{bn_1} + l_1\%\frac{N}{bn_1}$.
              Let $d[l_1bn_1 + l_2b + l_3] = W_{\mathbb{C}}[\mathrm{Ind}_{row}][\mathrm{Ind}_{col}]$.
            end for
          end for
        end for
        Evaluate $\hat{ct} = \hat{ct} + \mathrm{PMult}(\mathrm{Rot}_{-k_1t_1bn_1}(\mathbf{d}), ct_{jt_1+k_2k})$. ▷ ct-pt multiplication
      end for
    end for
    if $k_1 = 0$ **then**
      Evaluate $\mathrm{Enc}(\hat{\mathbf{m}}^i) = \mathrm{Rot}^{\mathrm{HE}}_{k_1t_1bn_1}(\hat{ct})$.
    else
      Evaluate $\mathrm{Enc}(\hat{\mathbf{m}}^i) = \mathrm{Enc}(\hat{\mathbf{m}}^i) + \mathrm{Rot}^{\mathrm{HE}}_{k_1t_1bn_1}(\hat{ct})$.
    end if
  end for
end for

B.2 Ciphertext-ciphertext matrix multiplication (CCMM) in LoRA

LoRA fine-tuning updates low rank matrices A, B in $W+AB$, where $W \in \mathbb{R}^{n_2 \times n_2'}$, $A \in \mathbb{R}^{n_2 \times n_{\mathrm{LoRA}}}$, $B \in \mathbb{R}^{n_{\mathrm{LoRA}} \times n_2'}$, and $n_{\mathrm{LoRA}} \ll n_2'$. Notice that A and B are updating during fine-tuning, so they are encrypted matrices.

Therefore, evaluating the linear layer $X(W + AB) = XW + (XA)B$ requires one CPMM (Algorithm 2) to evaluate XW and 2 CCMMs (Algorithm 5 and 8) to evaluate $(XA)B$. For simplicity, we assume $(bn_1)|N$, $N|(bn_1n_2)$, and $N|(bn_1n_2')$. As shown in [28], special packing and matrix multiplications for low rank matrices A, B enables much better latency. The packing of A and B are elaborated in Algorithm 3 and 6, respectively.

Optimizations with imaginary part. According to Algorithm 2, the encrypted XW is packed in complex number field $\mathbb{C}$ instead of $\mathbb{R}$. Thus we hope encrypted XAB has the same packing format as XW so that we can add them directly. In addition, packing XAB in $\mathbb{C}$ also saves many FHE

Algorithm 3 Repeated column packing for matrix A in LoRA.

Input: Matrix $A \in \mathbb{R}^{n_2 \times n_{\text{LoRA}}}$, $r_1 = \frac{bn_1n_2}{N}$.
Output: $\{\text{Enc}(\mathbf{a}^i)\}_{i\in[r_A]}$ where $r_A = r_1 n_{\text{LoRA}}$.

for i from 0 to $n_{\text{LoRA}} - 1$ **do**
  for j from 0 to $r_1 - 1$ **do**
    for k_1 from 0 to $\frac{n_2}{r_1} - 1$ **do**
      for k_2 from 0 to $bn_1 - 1$ **do**
        Let $\mathbf{a}^{ir_1+j}[k_1bn_1 + k_2] = A[j\frac{n_2}{r_1} + k_1][i]$.
      end for
    end for
    Evaluate $\text{Enc}(\mathbf{a}^{ir_1+j})$.
  end for
end for

Algorithm 4 Repeated row packing for matrix B in LoRA.

Input: Matrix $B \in \mathbb{R}^{n_{\text{LoRA}} \times n_2'}$, $r_1 = \frac{bn_1n_2}{N}$.
Output: $\{\text{Enc}(\mathbf{b}^i)\}_{i\in[r_B]}$ where $r_B = r_1 n_{\text{LoRA}}$.

for i from 0 to $n_{\text{LoRA}} - 1$ **do**
  for j from 0 to $r_1 - 1$ **do**
    for k_1 from 0 to $\frac{n_2'}{r_1} - 1$ **do**
      for k_2 from 0 to $bn_1 - 1$ **do**
        Let $\mathbf{b}^{ir_1+j}[k_1bn_1 + k_2] = B[i][j\frac{n_2'}{r_1} + k_1]$.
      end for
    end for
    Evaluate $\text{Enc}(\mathbf{b}^{ir_1+j})$.
  end for
end for

Algorithm 5 Ciphertext-Ciphertext matrix multiplication for matrix A in LoRA.

Input: Encrypted matrices $\{X^i \in \mathbb{R}^{n_1 \times n_2}\}_{i\in[b]}$ in b-interleaved column packing $\{\text{Enc}(\mathbf{m}^i)\}_{i\in[r_1]}$, $r_1 = \frac{bn_1n_2}{N}$. Encrypted matrix $A \in \mathbb{R}^{n_2 \times n_{\text{LoRA}}}$ in repeated column packing $\{\text{Enc}(\mathbf{a}^i)\}_{i\in[r_A]}$, $r_A = r_1 n_{\text{LoRA}}$.
Output: Encrypted matrices $\{X^iA \in \mathbb{R}^{n_1 \times n_{\text{LoRA}}}\}_{i\in[b]}$: $\{\text{Enc}(\hat{\mathbf{m}}^i)\}_{i\in[r_{\text{XA}}]}$, $r_{\text{XA}} = n_{\text{LoRA}}$.

for i from 0 to $n_{\text{LoRA}} - 1$ **do**
  for j from 0 to $r_1 - 1$ **do**
    Evaluate $ct = \text{Mult}(\text{Enc}(\mathbf{m}^j), \text{Enc}(\mathbf{a}^{ir_1+j}))$.
    Let $\text{Ind}_{rot} = N/2$.
    if j = 0 **then**
      Let $\text{Enc}(\hat{\mathbf{m}}^i) = ct$.
    else
      Evaluate $\text{Enc}(\hat{\mathbf{m}}^i) = \text{Enc}(\hat{\mathbf{m}}^i) + ct$.
    end if
  end for
  while $\text{Ind}_{rot} \geq (bn_1)$ **do**
    Evaluate $\text{Enc}(\hat{\mathbf{m}}^i) = \text{Enc}(\hat{\mathbf{m}}^i) + \text{Rot}^{\text{HE}}_{\text{Ind}_{rot}}(\text{Enc}(\hat{\mathbf{m}}^i))$.
    Evaluate $\text{Ind}_{rot} = \text{Ind}_{rot}/2$.
  end while
end for

Algorithm 6 Ciphertext-Ciphertext matrix multiplication for matrix B in LoRA.

Input: Encrypted matrices $\{(XA)^i \in \mathbb{R}^{n_1 \times n_{\text{LoRA}}}\}_{i\in[b]}$ output by Algorithm 5: $\{\text{Enc}(\mathbf{m}^i)\}_{i\in[r_{\text{XA}}]}$, $r_{\text{XA}} = n_{\text{LoRA}}$. Encrypted matrix $B \in \mathbb{R}^{n_{\text{LoRA}} \times n_2'}$ in repeated row packing: $\{\text{Enc}(\mathbf{b}^i)\}_{i\in[r_B]}$, $r_B = r_A$.

Output: Encrypted matrices $\{X^iAB \in \mathbb{R}^{n_1 \times n_2'}\}_{i\in[b]}$ in b-interleaved column packing $\{\text{Enc}(\hat{\mathbf{m}}^i)\}_{i\in[r_2]}$, $r_2 = \frac{bn_1n_2'}{N}$.

for i from 0 to $r_2 - 1$ **do**
  for j from 0 to $r_{\text{XA}} - 1$ **do**
    Evaluate $ct = \text{Mult}(\text{Enc}(\mathbf{m}^j), \text{Enc}(\mathbf{b}^{jr_2+i}))$.
    if j = 0 **then**
      Let $\text{Enc}(\hat{\mathbf{m}}^i) = ct$.
    else
      Evaluate $\text{Enc}(\hat{\mathbf{m}}^i) = \text{Enc}(\hat{\mathbf{m}}^i) + ct$.
    end if
  end for
end for

operations. We apply this optimization by modifing the repeated row packing of B, as shown in Algorithm 7. The optimized matrix multiplication algorithm with B is in Algorithm 8.

Similarly we can make use of the imaginary part of the CKKS slots in the packing and multiplications of the encrypted matrix $B \in \mathbb{R}^{n_{LoRA} \times n_2'}$. By using imaginary part, we can reduce the number of output ciphertexts in both packing and multiplication by half. We present the packing and multiplication algorithms for B in Algorithm 7 and Algorithm 8, respectively.

Algorithm 7 Repeated row packing in $\mathbb{C}$ for matrix B in LoRA.

Input: Matrix $B \in \mathbb{R}^{n_{\text{LoRA}} \times n_2'}$

Output: $\{\text{Enc}(\mathbf{b}^i \in \mathbb{C}^N)\}_{i\in[r_B]}$, $r_B = \frac{r_1 n_{\text{LoRA}}}{2}$, $r_1 = \frac{bn_1n_2}{N}$.

for i from 0 to $n_{\text{LoRA}} - 1$ **do**
  for j from 0 to $\frac{r_1}{2} - 1$ **do**
    for k_1 from 0 to $\frac{n_2'}{r_1} - 1$ **do**
      for k_2 from 0 to $bn_1 - 1$ **do**
        Let $\mathbf{b}^{\frac{ir_1}{2}+j}[k_1bn_1 + k_2] = B[i][2j\frac{n_2'}{r_1} + k_1] + \sqrt{-1}B[i][(2j+1)\frac{n_2'}{r_1} + k_1]$.
      end for
    end for
    Evaluate $\text{Enc}(\mathbf{b}^{\frac{ir_1}{2}+j})$.
  end for
end for

Rotations in a batch. Number of HE rotations in $(XA)B$ is

$$\#(\text{Rot}^{\text{HE}}) = n_{\text{LoRA}} \cdot \log(N/(bn_1)).$$

If we set $N = 32768$, $b = 16$, $X^i \in \mathbb{R}^{128\times768}$, $A \in \mathbb{R}^{768\times2}$, and $B \in \mathbb{R}^{2\times768}$ then the number of HE rotations is 8.

Relinearizations. Number of relinearizations in $(XA)B$ by using Algorithm 5 and Algorithm 6 (with lazy relinearization in Algorithm 6) is

$$\#(\text{Relinearization}^{\text{HE}}) = n_{\text{LoRA}} + r_2 = n_{\text{LoRA}} + \frac{bn_1n_2'}{N}.$$

By using imaginary parts in B, we reduce the number of output ciphertexts by half, so the number of relinearizations in $(XA)B$ by using Algorithm 5 and Algorithm 8 (with lazy relinearization in Algorithm 8) is

$$\#(\text{Relinearization}^{\text{HE}}) = n_{\text{LoRA}} + \frac{bn_1n_2'}{2N}.$$

Algorithm 8 Optimized Ciphertext-Ciphertext matrix multiplication for matrix B in LoRA.

Input: Encrypted matrices $\{(XA)^i \in \mathbb{R}^{n_1 \times n_{\mathrm{LoRA}}}\}_{i\in[b]}$ output by Algorithm 5: $\{\mathrm{Enc}(\mathbf{m}^i \in \mathbb{R}^N)\}_{i\in[r_{\mathrm{XA}}]}$, $r_{\mathrm{XA}} = n_{\mathrm{LoRA}}$. Encrypted matrix $B \in \mathbb{R}^{n_{\mathrm{LoRA}} \times n_2'}$ in optimized repeated row packing $\{\mathrm{Enc}(\mathbf{b}^i \in \mathbb{C}^N)\}_{i\in[r_B]}$, $r_B = \frac{r_A}{2}$.

Output: Encrypted matrices $\{X^iAB \in \mathbb{C}^{n_1 \times (n_2'/2)}\}_{i\in[b]}$ in b-interleaved column packing: $\{\mathrm{Enc}(\hat{\mathbf{m}}^i \in \mathbb{C}^N)\}_{i\in[r_2]}$, $r_2 = \frac{bn_1n_2'}{2N}$.

1: **for** i from 0 to $r_2 - 1$ **do**
2: **for** j from 0 to $r_{\mathrm{XA}} - 1$ **do**
3: Evaluate $ct = \mathrm{Mult}(\mathrm{Enc}(\mathbf{m}^j), \mathrm{Enc}(\mathbf{b}^{jr_2+i}))$.
4: **if** j = 0 **then**
5: Let $\mathrm{Enc}(\hat{\mathbf{m}}^i) = ct$.
6: **else**
7: Evaluate $\mathrm{Enc}(\hat{\mathbf{m}}^i) = \mathrm{Enc}(\hat{\mathbf{m}}^i) + ct$.
8: **end if**
9: **end for**
10: **end for**

If we set $N = 32768$, $b = 16$, $X^i \in \mathbb{R}^{128\times768}$, $A \in \mathbb{R}^{768\times2}$, and $B \in \mathbb{R}^{2\times768}$ then numbers of HE relinearzations are 50 and 26 respectively.

Multiplicative levels and ct-ct multiplications in a batch. Each CCMM consumes 1 multiplicative level. The two CCMMs in LoRA consume 2 multiplicative levels with $n_{\mathrm{LoRA}}(r_1 + r_2)$ ct-ct multiplications. r_2 is decided by the packing and multiplication algorithms of B. If we set $N = 32768$, $b = 16$, $X^i \in \mathbb{R}^{128\times768}$, $A \in \mathbb{R}^{768\times2}$, and $B \in \mathbb{R}^{2\times768}$ then numbers of HE ct-ct multiplications is 192 and 144 respectively.

In summary, the required number of each HE operation for obtaining $\{\mathrm{Enc}(X^iAB)\}_{i\in[b]}$, where $X^i \in \mathbb{R}^{n_1\times n_2}$, $A \in \mathbb{R}^{n_2\times n_{\mathrm{LoRA}}}$, and $B \in \mathbb{R}^{n_{\mathrm{LoRA}}\times n_2'}$, is listed in Table 8.

Table 8: The number of each HE operation for $\{\mathrm{Enc}(X^i)\,\mathrm{Enc}(A)\,\mathrm{Enc}(B)\}_{i\in[b]}$ in our scheme

Operations	Rot	ct-ct Mult	Relinearization	ct-pt Mult	Level
LoRA-CCMM-basic	$n_{\mathrm{LoRA}} \cdot \log(N/(bn_1))$	$\frac{bn_1n_{\mathrm{LoRA}}(n_2+n_2')}{N}$	$n_{\mathrm{LoRA}} + \frac{bn_1n_2'}{N}$	0	2
LoRA-CCMM-optimized	$n_{\mathrm{LoRA}} \cdot \log(N/(bn_1))$	$\frac{bn_1n_{\mathrm{LoRA}}(2n_2+n_2')}{2N}$	$n_{\mathrm{LoRA}} + \frac{bn_1n_2'}{2N}$	0	2

B.3 Ciphertext matrix computations in kernel

For the sake of simplicity we omit the index $i \in [b]$ in X in the matrix computation flow, so thus Q_h, K_h, V_h and S_h, unless such an index is necessary.

The matrix computations in kernels, like Gaussian kernel and Epanechnikov kernel, consists of (1) computing l_2 norms of $Q_h[i][:] - K_h[j][:]$, where $i, j \in [n_1]$ and $h \in [H]$; (2) computing ciphertext-ciphertext matrix multiplication between kernel results and V_h.

For head $h \in [H]$, the matrix that stores all l_2 norms for one head is defined by:

$$S_h = \begin{pmatrix} \sum_{i=0}^{n_3-1}(Q_h[0][i] - K_h[0][i])^2 & \cdots & \sum_{i=0}^{n_3-1}(Q_h[0][i] - K_h[n_1-1][i])^2 \\ \vdots & \ddots & \vdots \\ \sum_{i=0}^{n_3-1}(Q_h[n_1-1][i] - K_h[0][i])^2 & \cdots & \sum_{i=0}^{n_3-1}(Q_h[n_1-1][i] - K_h[n_1-1][i])^2 \end{pmatrix},$$

where $S_h \in \mathbb{R}^{n_1\times n_1}$, and $Q_h, K_h \in \mathbb{R}^{n_1\times n_3}$, $n_3 = n_2/H$.

We propose an efficient matrix computation pipeline for this kind of kernels.

Definition B.2 (Flexible interleaved diagonal packing)**.** *Suppose we have b matrices $\{X^i \in \mathbb{R}^{n_1 \times n_1}\}_{i\in[b]}$ be:*

$$X^i = \begin{pmatrix} X^i[0][0] & \cdots & X^i[0][n_1-1] \\ \vdots & \ddots & \vdots \\ X^i[n_1-1][0] & \cdots & X^i[n_1-1][n_1-1] \end{pmatrix}.$$

Let $\{\mathbf{m}^i \in \mathbb{R}^N\}_{i\in[r]}$, $r = \lceil n_1/\lfloor N/(bn_1)\rfloor\rceil$ denote the plaintext vectors obtained by interleaved column packing the matrices $X^0, ..., X^{b-1}$. For simplicity, we assume $(bn_1)|N$ and $N|(bn_1^2)$. We define:

$$m^i[j] = X^{j\%b}\left[\lfloor\frac{j}{b}\rfloor\%n_1\right]\left[\left(\frac{iN}{bn_1} + \lfloor\frac{j}{bn_1}\rfloor + \lfloor\frac{j}{b}\rfloor\right)\%n_1\right]$$

*where $i \in [\frac{bn_1^2}{N}]$, and $j \in [N]$. We say that $\{\mathrm{Enc}(\mathbf{m}^i)\}_{i\in[\frac{bn_1^2}{N}]}$ is in b-**interleaved diagonal packing**.*

Algorithm 9 takes encrypted Q_h, K_h in b-interleaved column packing as input and output an encrypted matrix S_h in b-interleaved diagonal packing. [11] Then Algorithm 10 takes encrypted $\sigma(S_h)$ in b-interleaved diagonal packing and encrypted V_h in b-interleaved column packing as input, and outputs an encrypted matrix $\sigma(S_h)V_h$ in b-interleaved column packing.

Optimizations. Applying the baby-step-giant-step (BSGS) from [12, 24] can reduce the required FHE rotations from $O(n_1)$ to $O(\sqrt{n_1})$ in the multiplication between plaintext matrix and ciphertext vector with dimension n_1. When the matrix is encrypted, [4, 32] proposed the double BSGS: the matrix is packed and encrypted in diagonal form, and all the diagonals are already rotated. By modifying Algorithm 9, the output diagonals are rotated as desired by the double BSGS. This modification can also reduce the number of required rotations in Algorithm 9 from $O(n_1)$ to $O(\sqrt{n_1})$. The modified algorithm is shown in Algorithm 11.

Further, if V_h is packed in $\mathbb{C}$ instead of $\mathbb{R}$, then about half of the FHE operations can be saved because the number of ciphertexts to encrypt V_h is halved. Algorithm 11 and Algorithm 12 show the optimized matrix computations for S_h and $\sigma(S_h) \times V_h$, respectively.

Rotations in a batch. Number of HE rotations in computing S_h by using Algorithm 9 is

$$\#(\text{Rot}^{\text{HE}}) = n_1 \cdot (2 \cdot r_2' + \log(N/(bn_1))) = \frac{2bn_1^2 n_3}{N} + n_1 \log(N/(bn_1)).$$

Number of HE rotations in $\times V_h$ by using Algorithm 10 is also

$$\#(\text{Rot}^{\text{HE}}) = n_1 \cdot (2 \cdot r_2' + \log(N/(bn_1))) = \frac{2bn_1^2 n_3}{N} + n_1 \log(N/(bn_1)).$$

Numbers of HE rotations in S_h and $\times V_h$ are reduced in Algorithm 11 and Algorithm 12 due to applying the double BSGS technique [4, 32].

Number of HE rotations in S_h by using Algorithm 11 is

$$\#(\text{Rot}^{\text{HE}}) = 2 \cdot r_{\text{QK}} \cdot r_2' + 2 \cdot \frac{n_1}{r_{\text{QK}}} \cdot r_2' + n_1 \cdot \log(\frac{N}{bn_1})) = \frac{bn_1 n_3}{N}(\frac{2bn_1^2}{N} + \frac{2N}{bn_1}) + n_1 \log(\frac{N}{bn_1}).$$

Number of HE rotations in $\times V_h$ by using Algorithm 12 is

$$\#(\text{Rot}^{\text{HE}}) = 2\cdot\frac{n_1}{r_{\text{QK}}}\cdot r_2^* + n_1\cdot\log(\frac{N}{bn_1})) + 2\cdot r_{\text{QK}}\cdot r_2^* + r_2^* = \frac{bn_1 n_3}{2N}(\frac{2N}{bn_1} + \frac{2bn_1^2}{N} + 1) + n_1 \log(\frac{N}{bn_1}).$$

If we set $N = 32768$, $b = 16$, $H = 12$, $Q_h, K_h, V_h \in \mathbb{R}^{128\times 64}$, $S_h \in \mathbb{R}^{128\times 128}$ then numbers of HE rotations in two versions of matrix operations of $\sigma(S_h)V_h$ are 3072 and 1314, respectively.

[11] Using this notation, the b-interleaved diagonal packing in [38, 32] is $\frac{N}{n_1}$-interleaved diagonal packing with a predetermined parameter $\frac{N}{n_1}$. Our packing parameter b is flexible.

Algorithm 9 Ciphertext-Ciphertext matrix computation for S_h.

Input: Encrypted matrices Q_h, and K_h in b-interleaved column packing $\{\mathrm{Enc}(\mathbf{q}^i)\}_{i\in[r'_2]}$, and $\{\mathrm{Enc}(\mathbf{k}^i)\}_{i\in[r'_2]}$, $r'_2 = \frac{bn_1n_3}{N}$ where $n_3 = n_2/H$.

Output: Encrypted matrix S_h in b-interleaved diagonal packing $\{\mathrm{Enc}(\hat{\mathbf{m}}^i)\}_{i\in[r_{\mathrm{QK}}]}$, $r_{\mathrm{QK}} = \frac{bn_1^2}{N}$.

for ind from 0 to $r_{\mathrm{QK}} - 1$ **do**
  for i from 0 to $\frac{n_1}{r_{\mathrm{QK}}} - 1$ **do**
    $\hat{ct} = \mathrm{Enc}(\mathbf{0})$
    for j from 0 to $r'_2 - 1$ **do**
      Evaluate $ct' = \mathrm{Rot}^{\text{HE,Internal}}_{\mathrm{ind}n_1/r_{\mathrm{QK}}+i}(\mathrm{Enc}(\mathbf{k}^j))$.
      Evaluate $ct' = \mathrm{Mult}(\mathrm{Enc}(\mathbf{q}^j) - ct', \mathrm{Enc}(\mathbf{q}^j) - ct')$.
      Let $\mathrm{Ind}_{rot} = N/2$.
      if j = 0 **then**
        $\hat{ct} = ct'$.
      else
        Evaluate $\hat{ct} = \hat{ct} + ct'$
      end if
    end for
    while $\mathrm{Ind}_{rot} \geq (bn_1)$ **do**
      Evaluate $\hat{ct} = \hat{ct} + \mathrm{Rot}^{\text{HE}}_{\mathrm{Ind}_{rot}}(\hat{ct})$.
      Evaluate $\mathrm{Ind}_{rot} = \mathrm{Ind}_{rot}/2$.
    end while
    Let $\mathbf{mask} = \mathbf{0}$.
    for j from $i \cdot bn_1$ to $(i+1) \cdot bn_1 - 1$ **do**
      Let $\mathbf{mask}[j] = 1$.
    end for
    Evaluate $\hat{ct} = \mathrm{PMult}(\hat{ct}, \mathbf{mask})$.
    if i = 0 **then**
      Let $\mathrm{Enc}(\hat{\mathbf{m}}^{\mathrm{ind}}) = \hat{ct}$.
    else
      Evaluate $\mathrm{Enc}(\hat{\mathbf{m}}^{\mathrm{ind}}) = \mathrm{Enc}(\hat{\mathbf{m}}^{\mathrm{ind}}) + \hat{ct}$.
    end if
  end for
end for

Relinearizations. Number of relinearizations in S_h in either algorithm is

$$\#(\text{Relinearization}^{\text{HE}}) = n_1.$$

Numbers of relinearizations in $\times V_h$ by using Algorithm 10 (with lazy relinearization) are

$$\#(\text{Relinearization}^{\text{HE}}) = r'_2 = \frac{bn_1n_3}{N}.$$

Numbers of relinearizations in $\times V$ by using Algorithm 12 (with lazy relinearization) are

$$\#(\text{Relinearization}^{\text{HE}}) = r^*_2 r_{\mathrm{QK}} = \frac{b^2n_1^3n_3}{2N^2},$$

If we set $N = 32768$, $b = 16$, $H = 12$, $Q_h, K_h, V_h \in \mathbb{R}^{128\times 64}$, $S_h \in \mathbb{R}^{128\times 128}$ then numbers of relinearizations in two versions of matrix operations of $\sigma(S_h)V_h$ are 132 and 144, respectively.

Multiplicative levels, ct-ct multiplications and ct-pt multiplications in a batch. In the basic version (Algorithm 9 and Algorithm 10), computing S_h consumes 2 multiplicative levels for $\mathrm{Enc}(Q_h)$ and 3 multiplicative levels for $\mathrm{Enc}(K_h)$ because $\mathrm{Enc}(K_h)$ requires internal rotations which consumes 1 level. The number of HE ct-ct multiplications is $n_1r'_2$, and the number of HE ct-pt multiplications is $n_1(r'_2 + 1)$. Computing $\times V_h$ consumes 2 multiplicative levels for both inputs with also $n_1r'_2$ HE ct-ct multiplications and $n_1r'_2 + n_1$ HE ct-pt multiplications.

Algorithm 10 Ciphertext-Ciphertext matrix computation for $\times V_h$.

Input: Encrypted matrix S_h in b-interleaved diagonal packing $\{\mathrm{Enc}(\mathbf{m}^i)\}_{i\in[r_{\mathrm{QK}}]}$, $r_{\mathrm{QK}} = \frac{bn_1^2}{N}$. Encrypted matrix V_h in b-interleaved column packing $\{\mathrm{Enc}(\mathbf{v}^i)\}_{i\in[r_2']}$, $r_2' = \frac{bn_1n_3}{N}$.

Output: Encrypted matrix $\sigma(S_h)V_h$ in b-interleaved column packing $\{\mathrm{Enc}(\hat{\mathbf{m}}^i)\}_{i\in[r_3']}$, $r_3' = \frac{bn_1n_3}{N}$.

for i from 0 to $r_{\mathrm{QK}} - 1$ **do**
  for j from 0 to $\frac{n_1}{r_{\mathrm{QK}}} - 1$ **do**
    Let $\mathbf{mask} = \mathbf{0}$.
    for k from $i \cdot bn_1$ to $(i+1) \cdot bn_1 - 1$ **do**
      Let $\mathbf{mask}[k] = 1$.
    end for
    Evaluate $\mathrm{Ext}_j(\mathrm{Enc}(\mathbf{m}^i)) = \mathrm{PMult}(\mathrm{Enc}(\mathbf{m}^i), \mathbf{mask})$. ▷ Extract j-th diagonal from $\mathrm{Enc}(\mathbf{m}^i)$.
    Let $\mathrm{Ind}_{rot} = N/2$.
    while $\mathrm{Ind}_{rot} \geq (bn_1)$ **do**
      $\mathrm{Ext}_j(\mathrm{Enc}(\mathbf{m}^i)) += \mathrm{Rot}^{\mathrm{HE}}_{\mathrm{Ind}_{rot}}(\mathrm{Ext}_j(\mathrm{Enc}(\mathbf{m}^i)))$. ▷ Copy j-th diagonal to all slots.
      Evaluate $\mathrm{Ind}_{rot} = \mathrm{Ind}_{rot}/2$.
    end while
  end for
end for
for ind from 0 to $r_2' - 1$ **do**
  for i from 0 to $r_{\mathrm{QK}} - 1$ **do**
    for j from 0 to $\frac{n_1}{r_{\mathrm{QK}}} - 1$ **do**
      Evaluate $ct^* = \mathrm{Rot}^{\mathrm{HE,Internal}}_{in_1/r_{\mathrm{QK}}+j}(\mathrm{Enc}(\mathbf{v}^{\mathrm{ind}}))$.
      Evaluate $ct' = \mathrm{Ext}_j(\mathrm{Enc}(\mathbf{m}^i)) \cdot ct^*$.
      if i = 0 and j = 0 **then**
        Let $\mathrm{Enc}(\hat{\mathbf{m}}^{\mathrm{ind}}) = ct'$.
      else
        Evaluate $\mathrm{Enc}(\hat{\mathbf{m}}^{\mathrm{ind}}) = \mathrm{Enc}(\hat{\mathbf{m}}^{\mathrm{ind}}) + ct'$.
      end if
    end for
  end for
end for

In the optimized version(Algorithm 11 and Algorithm 12), computing S_h consumes 3 multiplicative levels for $\mathrm{Enc}(Q_h)$ and $\mathrm{Enc}(K_h)$ because both of them require internal rotations. The number of HE ct-ct multiplications is $n_1 r_2'$, and the number of HE ct-pt multiplications is $r_2'(r_{\mathrm{QK}} + \frac{n_1}{r_{\mathrm{QK}}}) + n_1$. Computing $\times V_h$ consumes 3 multiplicative levels for both inputs with also $n_1 r_2^*$ HE ct-ct multiplications and $r_2^*(\frac{n_1}{r_{\mathrm{QK}}} + r_{\mathrm{QK}}) + n_1$ HE ct-pt multiplications.

If we set $N = 32768$, $b = 16$, $H = 12$, $Q_h, K_h, V_h \in \mathbb{R}^{128\times 64}$, $S_h \in \mathbb{R}^{128\times 128}$, then numbers of HE ct-ct multiplications in two versions of matrix operations of $\sigma(S_h)V_h$ are 1024 and 768, respectively. Numbers of HE ct-pt multiplications in two versions of matrix operations of $\sigma(S_h)V_h$ is 1280 and 400, respectively.

In summary, the required number of each HE operation for $\{S_h^i\}_{i\in[b]}$ is listed in Table 9. The required number of each HE operation for $\times V^i, i \in [b]$, where $V^i \in \mathbb{R}^{n_1\times n_3}$, is listed in Table 10.

Table 9: The number of each HE operation for $\{S_h^i\}_{i\in[b]}$ in one head in our scheme

Operations	Rot	ct-ct Mult	Relinearization	ct-pt Mult	Level
S_h-basic	$\frac{2bn_1^2n_3}{N} + n_1\log(N/(bn_1))$	$\frac{bn_1^2n_3}{N}$	n_1	$\frac{bn_1^2n_3}{N} + n_1$	3
S_h-optimized	$\frac{bn_1n_3}{N}(\frac{2bn_1^2}{N} + \frac{2N}{bn_1}) + n_1\log(\frac{N}{bn_1})$	$\frac{bn_1^2n_3}{N}$	n_1	$\frac{b^2n_1^3n_3}{N^2}$	3

Algorithm 11 Ciphertext-Ciphertext matrix computation for S_h with double BSGS.

Input: Encrypted matrices Q_h, and K_h in b-interleaved column packing $\{\mathrm{Enc}(\mathbf{q}^i)\}_{i\in[r'_2]}$, and $\{\mathrm{Enc}(\mathbf{k}^i)\}_{i\in[r'_2]}$, $r'_2 = \frac{bn_1n_3}{N}$.

Output: Encrypted matrix S_h in rotated b-interleaved diagonal packing $\{\mathrm{Enc}(\hat{\mathbf{m}}^i)\}_{i\in[r_{\mathrm{QK}}]}$, $r_{\mathrm{QK}} = \frac{bn_1^2}{N}$.

for ind from 0 to $r_{\mathrm{QK}} - 1$ **do**
  for j from 0 to $r'_2 - 1$ **do**
    Let $ct^q_{\mathrm{ind}r'_2+j} = \mathrm{Rot}^{\mathrm{HE,Internal}}_{-\mathrm{ind}n_1/r_{\mathrm{QK}}}(\mathrm{Enc}(\mathbf{q}^j))$.
  end for
end for
for i from 0 to $\frac{n_1}{r_{\mathrm{QK}}} - 1$ **do**
  for j from 0 to $r'_2 - 1$ **do**
    Let $ct^k_{ir'_2+j} = \mathrm{Rot}^{\mathrm{HE,Internal}}_{i}(\mathrm{Enc}(\mathbf{k}^j))$.
  end for
end for
for ind from 0 to $r_{\mathrm{QK}} - 1$ **do**
  for i from 0 to $\frac{n_1}{r_{\mathrm{QK}}} - 1$ **do**
    Initialize $\hat{ct} = \mathrm{Enc}(\mathbf{0})$.
    for j from 0 to $r'_2 - 1$ **do**
      Evaluate $ct' = \mathrm{Mult}(ct^q_{\mathrm{ind}r'_2+j} - ct^k_{ir'_2+j}, ct^q_{\mathrm{ind}r'_2+j} - ct^k_{ir'_2+j})$.
      Let $\mathrm{Ind}_{rot} = N/2$.
      if j = 0 **then**
        $\hat{ct} = ct'$.
      else
        Evaluate $\hat{ct} = \hat{ct} + ct'$
      end if
    end for
    while $\mathrm{Ind}_{rot} \geq (bn_1)$ **do**
      Evaluate $\hat{ct} = \hat{ct} + \mathrm{Rot}^{\mathrm{HE}}_{\mathrm{Ind}_{rot}}(\hat{ct})$.
      Evaluate $\mathrm{Ind}_{rot} = \mathrm{Ind}_{rot}/2$.
    end while
    Let $\mathbf{mask} = \mathbf{0}$.
    for j from $i \cdot bn_1$ to $(i+1) \cdot bn_1 - 1$ **do**
      Let $\mathbf{mask}[j] = 1$.
    end for
    Evaluate $\hat{ct} = \mathrm{PMult}(\hat{ct}, pt_{\mathbf{mask}})$.
    if i = 0 **then**
      Let $\mathrm{Enc}(\hat{\mathbf{m}}^{\mathrm{ind}}) = \hat{ct}$.
    else
      Evaluate $\mathrm{Enc}(\hat{\mathbf{m}}^{\mathrm{ind}}) = \mathrm{Enc}(\hat{\mathbf{m}}^{\mathrm{ind}}) + \hat{ct}$.
    end if
  end for
end for

Table 10: The number of each HE operation for $\{\sigma(S_h^i)V_h^i\}, i \in [b]$ in our scheme

Operations	Rot	ct-ct Mult	Relinearization	ct-pt Mult	Level
$\times V_h$-CCMM-basic	$\frac{2bn_1^2n_3}{N} + n_1\log(N/(bn_1))$	$\frac{bn_1^2n_3}{N}$	$\frac{bn_1n_3}{N}$	$\frac{bn_1^2n_3}{N} + n_1$	2
$\times V_h$-CCMM-optimized	$\frac{bn_1n_3}{2N}(\frac{2N}{bn_1} + \frac{2bn_1^2}{N} + 1) + n_1\log(\frac{N}{bn_1})$	$\frac{bn_1^2n_3}{2N}$	$\frac{b^2n_1^3n_3}{2N^2}$	$\frac{bn_1n_3}{2N}(\frac{bn_1^2}{N} + \frac{N}{bn_1}) + n_1$	3

Algorithm 12 Ciphertext-Ciphertext matrix computation for $\times V$ with double BSGS.

Input: Encrypted matrix $\sigma(S_h)$ in rotated b-interleaved diagonal packing $\{\mathrm{Enc}(\mathrm{Rot}_l(\mathbf{m}^i \in \mathbb{R}^N))\}_{i\in[r_{\mathrm{QK}}]}$, $r_{\mathrm{QK}} = \frac{bn_1^2}{N}$. Encrypted matrix V in b-interleaved column packing in complex number field: $\{\mathrm{Enc}(\mathbf{v}^i \in \mathbb{C}^N)\}_{i\in[r_2^*]}$, $r_2^* = r_2'/2 = \frac{bn_1n_3}{2N}$.

Output: Encrypted matrix $\sigma(S_h)V_h$ in b-interleaved column packing $\{\mathrm{Enc}(\hat{\mathbf{m}}^i \in \mathbb{C}^N)\}_{i\in[r_3']}$, $r_3' = \frac{bn_1n_3}{N}$.

for i from 0 to $r_{\mathrm{QK}} - 1$ **do**
  for j from 0 to $\frac{n_1}{r_{\mathrm{QK}}} - 1$ **do**
    Let $\mathbf{mask} = \mathbf{0}$.
    for k from $i \cdot bn_1$ to $(i+1) \cdot bn_1 - 1$ **do**
      Let $\mathbf{mask}[k] = 1$.
    end for
    $\mathrm{Ext}_j(\mathrm{Enc}(\mathbf{m}^i)) = \mathrm{PMult}(\mathrm{Enc}(\mathbf{m}^i), \mathbf{mask})$. ▷ Extract j-th diagonal from $\mathrm{Enc}(\mathbf{m}^i)$.
    Let $\mathrm{Ind}_{rot} = N/2$.
    while $\mathrm{Ind}_{rot} \geq (bn_1)$ **do**
      $\mathrm{Ext}_j(\mathrm{Enc}(\mathbf{m}^i)) += \mathrm{Rot}^{\mathrm{HE}}_{\mathrm{Ind}_{rot}}(\mathrm{Ext}_j(\mathrm{Enc}(\mathbf{m}^i)))$. ▷ Copy j-th diagonal to all slots.
      Evaluate $\mathrm{Ind}_{rot} = \mathrm{Ind}_{rot}/2$.
    end while
  end for
end for
for ind from 0 to $r_2^* - 1$ **do**
  for j from 0 to $\frac{n_1}{r_{\mathrm{QK}}} - 1$ **do**
    Let $ct_j^* = \mathrm{Rot}_j^{\mathrm{HE,Internal}}(\mathrm{Enc}(\mathbf{v}^{\mathrm{ind}}))$. ▷ Baby step.
  end for
  for i from 0 to $r_{\mathrm{QK}} - 1$ **do**
    Initialize $\hat{ct} = \mathrm{Enc}(\mathbf{0})$.
    for j from 0 to $\frac{n_1}{r_{\mathrm{QK}}} - 1$ **do**
      Evaluate $ct' = \mathrm{Mult}(\mathrm{Ext}_j(\mathrm{Enc}(\mathbf{m}^i)), ct_j^*)$.
      if $j = 0$ **then**
        Let $\hat{ct} = ct'$.
      else
        Evaluate $\hat{ct} = \hat{ct} + ct'$.
      end if
    end for
    Evaluate $\hat{ct} = \mathrm{Rot}^{\mathrm{HE,Internal}}_{in_1/r_{\mathrm{QK}}}(\hat{ct})$.
    if $i = 0$ **then**
      Let $\mathrm{Enc}(\hat{\mathbf{m}}^{\mathrm{ind}}_{\mathbb{C}}) = \hat{ct}$.
    else
      Evaluate $\mathrm{Enc}(\hat{\mathbf{m}}^{\mathrm{ind}}_{\mathbb{C}}) = \mathrm{Enc}(\hat{\mathbf{m}}^{\mathrm{ind}}_{\mathbb{C}}) + \hat{ct}$.
    end if
  end for
  Extract: ▷ Note that $r_3' = 2r_2^*$
  $\mathrm{Enc}(\hat{\mathbf{m}}^{2\mathrm{ind}}) := \mathrm{Enc}(\hat{\mathbf{m}}^{\mathrm{ind}}_{\mathbb{C}}) + \mathrm{Conj}(\mathrm{Enc}(\hat{\mathbf{m}}^{\mathrm{ind}}_{\mathbb{C}}))$.
  $\mathrm{Enc}(\hat{\mathbf{m}}^{2\mathrm{ind}+1}) := \mathrm{Enc}(\hat{\mathbf{m}}^{\mathrm{ind}}_{\mathbb{C}}) - \mathrm{Conj}(\mathrm{Enc}(\hat{\mathbf{m}}^{\mathrm{ind}}_{\mathbb{C}}))$.
end for

B.4 Ciphertext-ciphertext matrix multiplication in classification head

The matrix computations in classification head consists of (1) Extracting the $[CLS]$ (usually the first row): $\mathbf{h} \in \mathbb{R}^{1\times n_2}$ of the output of transformer layers; (2) Computing ciphertext-ciphertext matrix multiplication between $\mathbf{h}$ and $W_{c1} \in \mathbb{R}^{n_2\times n_2}$; (3) Computing ciphertext-ciphertext matrix multiplication between $\mathbf{h}' = \mathrm{Tanh}(\mathbf{h}W_{c1} + \mathbf{b}_{c1})$ and $W_{c2} \in \mathbb{R}^{n_2\times c}$.

The packing of W_{c1} and W_{c2} are elaborated in Algorithm 13 and Algorithm 14. The 3 computations are shown in Algorithm 15, Algorithm 16, and Algorithm 17.

Algorithm 13 Packing for matrix W_{c1} in classification head.

Input: Matrix $W_{c1} \in \mathbb{R}^{n_2\times n_2}$

Output: $\{\mathrm{Enc}(\mathbf{w}^i_{c1} \in \mathbb{R}^N)\}_{i\in[r_{c1}]}$, $r_{c1} = \frac{bn_2^2}{N}$.

for i from 0 to $\frac{n_2}{n_1} - 1$ **do**
  for j from 0 to $\frac{bn_1n_2}{N} - 1$ **do**
    Let $\mathbf{d} = \mathbf{0}$.
    for k_1 from 0 to $\frac{N}{bn_1} - 1$ **do**
      for k_2 from 0 to $n_1 - 1$ **do**
        for k_3 from 0 to $b - 1$ **do**
          Let $d[k_1bn_1 + k_2b + k_3] = W_{c1}[j\frac{N}{bn_1} + k_1][in_1 + k_2]$.
        end for
      end for
    end for
    Evaluate $\mathrm{Enc}(\mathbf{w}_{c1}^{i\frac{bn_1n_2}{N}+j}) = \mathrm{Enc}(\mathbf{d})$.
  end for
end for

Algorithm 14 Packing for matrix W_{c2} in classification head

Input: Matrix $W_{c2} \in \mathbb{R}^{n_2\times c}$

Output: $\{\mathrm{Enc}(\mathbf{w}^i_{c2} \in \mathbb{R}^N)\}_{i\in[c/2]}$.

for i from 0 to $\frac{c}{2} - 1$ **do**
  for j from 0 to $n_2 - 1$ **do**
    for k from 0 to $b - 1$ **do**
      Let $\mathbf{w}^i_{c2}[jb + k] = W_{c2}[j][2i]$.
      Let $\mathbf{w}^i_{c2}[N/2 + jb + k] = W_{c2}[j][2i + 1]$.
    end for
  end for
  Evaluate $\mathrm{Enc}(\mathbf{w}^i_{c2})$.
end for

Rotations in a batch. Number of HE rotations in extract the first row is

$$\#(\mathrm{Rot}^{\mathrm{HE}}) = r_1 \cdot \log(n_1) = \frac{bn_1n_2\log(n_1)}{N}.$$

Number of HE rotations in $(\mathbf{h}W_{c1})$ is

$$\#(\mathrm{Rot}^{\mathrm{HE}}) = \frac{n_2}{n_1} \cdot \log(N/(bn_1)) + 1.$$

Number of HE rotations in $(\mathbf{h}W_{c2})$ is

$$\#(\mathrm{Rot}^{\mathrm{HE}}) = \frac{c}{2} \cdot \log(N/(2b)).$$

If we set $N = 32768$, $b = 16$, $X^i \in \mathbb{R}^{128\times 768}$, $W_{c1} \in \mathbb{R}^{768\times 768}$, and $W_{c2} \in \mathbb{R}^{768\times 2}$ then the number of HE rotations in each step is 336, 25, and 10, respectively.

Algorithm 15 Extract the first row from output of transformer layers

Input: Encrypted matrix $\{X^i \in \mathbb{R}^{n_1 \times n_2}\}_{i\in[b]}$ in b-interleaved column packing $\{\mathrm{Enc}(\mathbf{m}^i \in \mathbb{R}^N)\}_{i\in[r_1]}$, $r_1 = \frac{bn_1n_2}{N}$.

Output: Encrypted vectors $\{\mathbf{h}^i \in \mathbb{R}^{n_2}\}_{i\in[b]}$ in repeated packing $\{\mathrm{Enc}(\hat{\mathbf{h}}^i)\}_{i\in[r_1]}$.

Let $\mathbf{mask} = \mathbf{0}$.
for i from 0 to $\frac{N}{bn_1} - 1$ **do**
  for j from 0 to $b - 1$ **do**
    Let $\mathbf{mask}[ibn_1 + j] = 1$.
  end for
end for
for i from 0 to $r_1 - 1$ **do**
  Evaluate $ct' = \mathrm{PMult}(\mathrm{Enc}(\mathbf{m}^i), \mathrm{mask})$.
  Let $\mathrm{Ind}_{rot} = b$.
  while $\mathrm{Ind}_{rot} < bn_1$ **do**
    Evaluate $ct' = ct' + \mathrm{Rot}^{\mathrm{HE}}_{N-\mathrm{Ind}_{rot}}(ct')$. ▷ Fill all the slots by repeated packing.
    $\mathrm{Ind}_{rot} = \mathrm{Ind}_{rot} \cdot 2$.
  end while
  Let $\mathrm{Enc}(\hat{\mathbf{h}}^i) = ct'$.
end for

Algorithm 16 Ciphertext-Ciphertext matrix multiplication for matrix W_{c1} in classification head

Input: Encrypted vectors $\{\mathbf{h}^i \in \mathbb{R}^{n_2}\}_{i\in[b]}$ in repeated packing $\{\mathrm{Enc}(\hat{\mathbf{h}}^i)\}_{i\in[r_1]}$, $r_1 = \frac{bn_1n_2}{N}$, encrypted matrix $W_{c1} \in R^{n_2\times n_2}$ output by Algorithm 13: $\{\mathrm{Enc}(\mathbf{w}^i_{c1} \in \mathbb{R}^N)\}_{i\in[r_{c1}]}$, $r_{c1} = \frac{bn_2^2}{N}$

Output: Encrypted vectors $\{\mathbf{h}^i W_{c1}\}_{i\in[b]}$: $\mathrm{Enc}(\mathbf{h}')$

for i from 0 to $\frac{r_{c1}}{r_1} - 1$ **do**
  Let $ct'_i = \mathrm{Enc}(\mathbf{0})$.
  for j from 0 to $r_1 - 1$ **do**
    Evaluate $ct'_i = ct'_i + \mathrm{Enc}(\hat{\mathbf{h}}^j) \cdot \mathrm{Enc}(\mathbf{w}^{ir_1+j}_{c1})$.
  end for
  Let $\mathrm{Ind}_{rot} = N/2$
  while $\mathrm{Ind}_{rot} \geq bn_1$ **do**
    Evaluate $ct'_i = ct'_i + \mathrm{Rot}^{\mathrm{HE}}_{\mathrm{Ind}_{rot}}(ct'_i)$.
    $\mathrm{Ind}_{rot} = \mathrm{Ind}_{rot}/2$.
  end while
  Let $\mathrm{mask} = \mathbf{0}$.
  for j from ibn_1 to $(i+1)bn_1 - 1$ **do**
    Let $\mathrm{mask}[j] = 1$.
  end for
  Evaluate $ct'_i = ct'_i \cdot \mathrm{mask}$.
  if i = 0 **then**
    Let $\mathrm{Enc}(\mathbf{h}') = ct'_i$.
  else
    Evaluate $\mathrm{Enc}(\mathbf{h}') = \mathrm{Enc}(\mathbf{h}') + ct'_i$.
  end if
end for
Evaluate $\mathrm{Enc}(\mathbf{h}') = \mathrm{Enc}(\mathbf{h}') + \mathrm{Rot}^{\mathrm{HE}}_{N/2}(\mathrm{Enc}(\mathbf{h}'))$.

Algorithm 17 Ciphertext-Ciphertext matrix multiplication for matrix W_{c2} in classification head

Input: Encrypted vectors $\{\mathbf{h}^i\}_{i\in[b]}$ output by Algorithm 16: $\mathrm{Enc}(\mathbf{h}')$, and encrypted matrix $W_{c2} \in \mathbb{R}^{n_2\times c}$ output by Algorithm 14: $\{\mathrm{Enc}(\mathbf{w}^i_{c2} \in \mathbb{R}^N)\}_{i\in[c/2]}$.

Output: Encrypted vectors $\{\mathbf{h}^i W_{c2} \in R^c\}_{i\in[b]}$: $\{\mathrm{Enc}(\hat{\mathbf{h}}^i)\}_{i\in[c/2]}$.

1: **for** i from 0 to $c/2$ **do**
2: Evaluate $\mathrm{Enc}(\mathbf{w}^i_{c2}) = \mathrm{Enc}(\mathbf{w}^i_{c2}) \cdot \mathrm{Enc}(\mathbf{h}')$.
3: Let $\mathrm{Ind}_{rot} = N/4$
4: **while** $\mathrm{Ind}_{rot} \geq b$ **do**
5: Evaluate $\mathrm{Enc}(\mathbf{w}^i_{c2}) = \mathrm{Enc}(\mathbf{w}^i_{c2}) + \mathrm{Rot}^{\mathrm{HE}}_{\mathrm{Ind}_{rot}}(\mathrm{Enc}(\mathbf{w}^i_{c2}))$.
6: $\mathrm{Ind}_{rot} = \mathrm{Ind}_{rot}/2$.
7: **end while**
8: **end for**

Relinearizations. Number of relinearizations in $(\mathbf{h}W_{c1})$ is

$$\#(\text{Relinearization}^{\text{HE}}) = \frac{r_{c1}}{r_1} = \frac{n_2}{n_1}.$$

Number of relinearizations in $(\mathbf{h}W_{c2})$ is

$$\#(\text{Relinearization}^{\text{HE}}) = \frac{c}{2}.$$

If we set $N = 32768$, $b = 16$, $X^i \in \mathbb{R}^{128\times 768}$, $W_{c1} \in \mathbb{R}^{768\times 768}$, and $W_{c2} \in \mathbb{R}^{768\times 2}$ then numbers of HE relinearzations are 6 and 1 respectively.

Multiplicative levels and ct-ct multiplications in a batch. Each CCMM consumes 1 multiplicative level, extraction also consumes 1 multiplicaitve level. The two CCMMs in classiciation heads consume 2 multiplicative levels with $\frac{bn_2^2}{N}$, and $\frac{c}{2}$ ct-ct multiplications. If we set $N = 32768$, $b = 16$, $X^i \in \mathbb{R}^{128\times 768}$, $W_{c1} \in \mathbb{R}^{768\times 768}$, and $W_{c2} \in \mathbb{R}^{768\times 2}$ then numbers of HE ct-ct multiplications is 288 and 1 respectively.

In summary, the required number of each HE operation in classification head, is listed in Table 11.

Table 11: The number of each HE operation for matrix computations in classification head in our scheme

Operations	Rot	ct-ct Mult	Relinearization	ct-pt Mult	Level
Extraction	$\frac{bn_1n_2\log(n_1)}{N}$	0	0	$\frac{bn_1n_2}{N}$	1
CCMM for W_{c1}	$\frac{n_2}{n_1}\cdot\log(N/(bn_1))+1$	$\frac{bn_2^2}{N}$	$\frac{n_2}{n_1}$	$\frac{n_2}{n_1}$	1
CCMM for W_{c2}	$\frac{c}{2}\cdot\log(N/(2b))$	$\frac{c}{2}$	$\frac{c}{2}$	0	1

C Backward Propagation for 2-Layer BERT with Kernel Attention and LoRA

Notations. We consider a 2-layer Transformer encoder with $H = 12$ heads per layer. The standard softmax attention is replaced by a kernel function $\phi(Q, K) \in \mathbb{R}^{n_1\times n_1}$ whose (i, j)-th entry is $\sigma\left(\|Q[i][:] - K[j][:]\|_2^2\right)$. For example, in the gaussian kernel:

$$\sigma(x) = \exp\left(-\frac{x}{2\sqrt{n_1}}\right), \quad \phi(Q,K)[i][j] = \exp\left(-\frac{1}{2\sqrt{n_1}}\|Q[i][:] - K[j][:]\|_2^2\right),$$

where n_1 is the sequence length.

Apply LoRA fine-tuning with rank n_{LoRA} in attention blocks. We define:

$$W_Q^{(\ell)} := W_{Q,frozen}^{(\ell)} + A_Q^{(\ell)} B_Q^{(\ell)},\ W_K^{(\ell)} := W_{K,frozen}^{(\ell)} + A_K^{(\ell)} B_K^{(\ell)},\ W_V^{(\ell)} := W_{V,frozen}^{(\ell)} + A_V^{(\ell)} B_V^{(\ell)}\ .$$

In our scheme, $\{A_*^{(\ell)}, B_*^{(\ell)}\}_{*=Q,K,V;\ell=0,1}, W_{c1}, W_{c2}$ are trainable parameters. Given vector $\mathbf{v} \in \mathbb{R}^l$, we have $\frac{\partial L}{\partial \mathbf{v}} \in \mathbb{R}^l$, i.e., the same shape of $\mathbf{v}$. Similarly, for matrix $M \in \mathbb{R}^{r\times l}$, we have $\frac{\partial L}{\partial M} \in \mathbb{R}^{r\times l}$ and thus $\frac{\partial L}{\partial M[i][j]} = \frac{\partial L}{\partial M}[i][j]$.

In this section, we list how to compute the gradients using encrypted data of all the trainable parameters. All data are packed using the *b-interleaved batching*, i.e., each batch contains b inputs. Since the evaluations for different inputs are independent and are executed in a SIMD manner, they do not interfere with one another. Therefore, it suffices to focus on a single input within a batch to explain the FHE algorithm flow in this section.

In Section C.1, we list the forward pass step-by-step. In Section C.2, we list the backward pass, using plaintext formulas. The implementations under FHE is elaborated in the boxed paragraphs in Section C.2. Finally, in Section C.3 we include how to compactly compress all the gradients in the fewest ciphertexts so that the number of the expensive bootstrapping operations is minimized. After this compression and optimizer, we decompress all the gradients and perform the model weight update.

C.1 Forward Pass

Let the initial input be $X^{(0)} \in \mathbb{R}^{n_1\times n_2}$, the first layer output be $X^{(1)}$, and the second layer output be $X^{(2)}$. Let $\mathbf{y}_{\text{true}} \in \{0,1\}^c$ be the sample's label in one-hot encoding.

Attention block: For the sake of simplicity, we omit all the bias vectors in the attention blocks. Define $W_Q^{(\ell)} = (W_{Q,0}^{(\ell)} \| W_{Q,1}^{(\ell)} \| ... \| W_{Q,11}^{(\ell)}) \in \mathbb{R}^{n_2\times n_2}$. Similarly define $W_K^{(\ell)}$ and $W_V^{(\ell)}$. Let $U_h^{(\ell)} = \phi(Q_h^{(\ell)}, K_h^{(\ell)})$. For layer $\ell \in \{0,1\}$ and attention head $h \in \{0,\ldots,11\}$:

$$Q_h^{(\ell)} = X^{(\ell)} W_{Q,h}^{(\ell)},\ K_h^{(\ell)} = X^{(\ell)} W_{K,h}^{(\ell)},\ V_h^{(\ell)} = X^{(\ell)} W_{V,h}^{(\ell)}$$
$$U_h^{(\ell)}[i][j] = \sigma\left(\|Q_{i,h}^{(\ell)}[i][:] - K_{j,h}^{(\ell)}[j][:]\|_2^2\right), \quad X_h^{(\ell)} = U_h^{(\ell)} V_h^{(\ell)}$$

Then the output of the attention block can be obtained by:

$$X_O^{(\ell)} = (X_0^{(\ell)} \| \ldots \| X_{11}^{(\ell)}) W_O^{(\ell)}, \quad X_{attn}^{(\ell)} = \text{LayerNorm}(X^{(\ell)} + X_O^{(\ell)})$$

Feedforward block:

$$X_{ffn}^{(\ell)} = \text{GELU}(X_{attn}^{(\ell)} W_{fc1}^{(\ell)} + \mathbf{b}_{fc1}^{(\ell)}) W_{fc2}^{(\ell)} + \mathbf{b}_{fc2}^{(\ell)}$$
$$X^{(\ell+1)} = \text{LayerNorm}(X_{ffn}^{(\ell)} + X_{attn}^{(\ell)})$$

Pooling and classification head:

$$\mathbf{h} = X_{[CLS]}^{(2)} \in \mathbb{R}^{n_2},\ \mathbf{y} = \tanh(\mathbf{h} W_{c1} + \mathbf{b}_{c1}) W_{c2} + \mathbf{b}_{c2}, \quad \text{Loss function } L = \mathcal{L}(\text{Softmax}(\mathbf{y}), \mathbf{y}_{\text{true}}) \in \mathbb{R}$$

C.2 Backward Propagation

To begin with, with the loss function being cross-entropy, we have $\frac{\partial L}{\partial \mathbf{y}} = \text{Softmax}(\mathbf{y}) - \mathbf{y}_{\text{true}}$, which is an encrypted row vector.

C.2.1 Pooling and classification head

$$\frac{\partial L}{\partial W_{c2}} = \tanh(\mathbf{h} W_{c1} + \mathbf{b}_{c1})^\intercal \frac{\partial L}{\partial \mathbf{y}}, \quad \frac{\partial L}{\partial W_{c1}} = \mathbf{h}^\intercal \left(\left(\frac{\partial L}{\partial \mathbf{y}} W_{c2}^\intercal\right) \odot \left(1 - \tanh^2(\mathbf{h} W_{c1} + \mathbf{b}_{c1})\right)\right)$$
$$\frac{\partial L}{\partial \mathbf{h}} = \left(\left(\frac{\partial L}{\partial \mathbf{y}} W_{c2}^\intercal\right) \odot \left(1 - \tanh^2(\mathbf{h} W_{c1} + \mathbf{b}_{c1})\right)\right) W_{c1}^\intercal.$$

From $\frac{\partial L}{\partial \mathbf{h}}$ to $\frac{\partial L}{\partial X^{(2)}}$ Recall that $\mathbf{h} = X^{(2)}_{[\mathrm{CLS}]} \in \mathbb{R}^{n_2}$ is the first row of $X^{(2)} \in \mathbb{R}^{n_1 \times n_2}$ corresponding to the [CLS] token. Then the gradient w.r.t. $X^{(2)}$ is zero everywhere except at the first row:

$$\frac{\partial L}{\partial X^{(2)}[i][:]} = \begin{cases} \frac{\partial L}{\partial \mathbf{h}}, & \text{if } i = 0, \\ \mathbf{0}, & \text{otherwise.} \end{cases}$$

1. $\frac{\partial L}{\partial \mathbf{y}} \in \mathbb{R}^{1\times c}$, $\tanh(\mathbf{h}W_{c1} + \mathbf{b}_{c1}) \in \mathbb{R}^{1\times n_2}$ are pre-stored from forward pass. Using $\mathbf{1} \in \mathbb{R}^{1\times n_2}$ we first obtain the ciphertext of $\tanh(\mathbf{h}W_{c1} + \mathbf{b}_{c1})^\intercal(1,1)$ and $\mathbf{1}^\intercal \frac{\partial L}{\partial \mathbf{y}}$ by $O(\log n_2)$ FHE rotations and additions. Then $\tanh(\mathbf{h}W_{c1} + \mathbf{b}_{c1})^\intercal \frac{\partial L}{\partial \mathbf{y}}$ is obtained by $\left(\tanh(\mathbf{h}W_{c1} + \mathbf{b}_{c1})^\intercal(1,1)\right) \odot \left(\mathbf{1}^\intercal \frac{\partial L}{\partial \mathbf{y}}\right)$.
2. Compute $\frac{\partial L}{\partial \mathbf{y}} W_{c2}^\intercal$ can be done by linear combination of all columns of W_{c2}, using $O(\log c)$ FHE rotations and 1 ctct multiplication. Then multiply $(1 - \tanh^2(\mathbf{h}W_{c1} + \mathbf{b}_{c1}))$ using Hadamard multiplication. The left multiplication with $\mathbf{h}^\intercal$ can be done similarly as the previous step.
3. Note that each sub-matrix of W_{c1} is packed in *row-major* packing, therefore each sub-matrix of $W_{c1}^\intercal$ is packed in *column-packing*. $\frac{\partial L}{\partial \mathbf{h}}$ is obtained by FHE multiplication between a row vector and a matrix in column-packing. This can be achieved by several inner-products - Hadamard multiplications followed by aggregations across slots.
4. $\frac{\partial L}{\partial X^{(2)}} \in \mathbb{R}^{n_1 \times n_2}$ is a matrix in which the first row is $\frac{\partial L}{\partial \mathbf{h}}$ and all other entries are 0.

C.2.2 Feedforward block

From $\frac{\partial L}{\partial X^{(\ell+1)}}$ to $\frac{\partial L}{\partial X^{(\ell)}_{attn}}$ Let $X^{(\ell+1)} = \mathrm{LayerNorm}(Z), Z = X^{(\ell)}_{ffn} + X^{(\ell)}_{attn}$ and $G = \frac{\partial L}{\partial X^{(\ell+1)}}$. Given $i \in [n_1]$, LayerNorm parameters $\boldsymbol{\gamma}, \boldsymbol{\beta} \in \mathbb{R}^{n_2}$, we have $\mu_i = \sum_{j\in[n_2]} Z[i][j]/n_2$ and:

$$\sigma_i^2 = \sum_{j\in[n_2]} (Z[i][j] - \mu_i)^2/n_2,\ \hat{Z}[i][j] = \frac{Z[i][j] - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}},\ X^{(\ell+1)}[i][j] = \gamma_j \hat{Z}[i][j] + \beta_j$$

We first find the backward formula of layer normalization for each $\frac{\partial L}{\partial Z[i][j]}$:

$$H[i][j] = G[i][j]\,\gamma_j, \quad m_i = \frac{1}{n_2}\sum_{j\in[n_2]} H[i][j], \quad v_i = \frac{1}{n_2}\sum_{j\in[n_2]} H[i][j]\hat{Z}[i][j]$$
$$\frac{\partial L}{\partial Z[i][j]} = \frac{1}{\sqrt{\sigma_i^2 + \epsilon}}\left(H[i][j] - m_i - \hat{Z}[i][j]v_i\right)\ .$$

And then we can obtain its compact matrix form ($\mathbf{m}, \mathbf{v}, \mathbf{1}$ are row vectors):

$$H = G \odot (\mathbf{1}^\intercal \boldsymbol{\gamma}), \quad \mathbf{m}^\intercal = \frac{1}{n_2} H\mathbf{1}^\intercal, \quad \mathbf{v}^\intercal = \frac{1}{n_2}(H \odot \hat{Z})\mathbf{1}^\intercal \tag{1}$$
$$\frac{\partial L}{\partial Z} = \left(\left(\frac{1}{\sqrt{\sigma_0^2 + \epsilon}}, \frac{1}{\sqrt{\sigma_1^2 + \epsilon}}, ..., \frac{1}{\sqrt{\sigma_{n_1}^2 + \epsilon}}\right)\mathbf{1}\right) \odot \left(H - \mathbf{m}^\intercal\mathbf{1} - \hat{Z} \odot (\mathbf{v}^\intercal\mathbf{1})\right) \tag{2}$$

We include Eq.(2) in the backward process of the feedforward block, defined as $G_Z := \frac{\partial L}{\partial Z}$. Then we have for residual split:

$$\frac{\partial L}{\partial X^{(\ell)}_{attn}}\Big|_{\text{res}} = G_Z, \quad \frac{\partial L}{\partial X^{(\ell)}_{ffn}} = G_Z$$

1. H is obtained from $G = \frac{\partial L}{\partial X^{(\ell+1)}}$ and plaintext matrix $\mathbf{1}^\intercal\boldsymbol{\gamma}$. $\mathbf{m}$ is obtained by FHE rotations and summations, computing the mean of each row in H.
2. $\hat{Z}$ is pre-stored from the forward pass, so $\mathbf{v}$ is computing the mean of each row in $H \odot \hat{Z}$.
3. $\left(\frac{1}{\sqrt{\sigma_0^2+\epsilon}}, \frac{1}{\sqrt{\sigma_1^2+\epsilon}}, ..., \frac{1}{\sqrt{\sigma_{n_1}^2+\epsilon}}\right)$ is pre-stored from the forward pass.

The feedforward block also contains $X_{ffn}^{(\ell)} = \mathrm{GELU}(U)W_{fc2}+\mathbf{b}_{fc2}$ where $U = X_{attn}^{(\ell)}W_{fc1}+\mathbf{b}_{fc1}$. Combining all parts together:

$$\frac{\partial L}{\partial X_{attn}^{(\ell)}} = \frac{\partial L}{\partial X_{attn}^{(\ell)}}\Big|_{\text{res}} + \frac{\partial L}{\partial X_{attn}^{(\ell)}}\Big|_{\text{ffn}} = G_Z + \left((G_Z W_{fc2}^T) \odot \mathrm{GELU}'(U)\right) W_{fc1}^T$$

1. $W_{fc1}^\intercal, W_{fc2}^\intercal$ are plaintext matrices. The multiplications with the two matrices are ciphertext-plaintext matrix multiplications.
2. $\mathrm{GELU}(x)$ is approximated by: $\mathrm{GELU}(x) \approx \frac{1}{2}x\left(1+\tanh(p(x))\right)$ where $p(x) := \sqrt{\frac{2}{\pi}}\left(x+0.044715\,x^3\right)$. Then

$$\mathrm{GELU}'(x) = \frac{1}{2}\left(1+\tanh(p(x))\right) + \frac{1}{2}x\left(1-\tanh^2(p(x))\right)\sqrt{\frac{2}{\pi}}\left(1+0.134145\,x^2\right) .$$

   Therefore, given x and $\tanh(p(x))$ pre-stored from the forward pass, we can obtain $\mathrm{GELU}'(U)$ by FHE polynomial evaluations.

C.2.3 Attention block

The first step is LayerNorm. The residual split: $\frac{\partial L}{\partial X^{(\ell)}}\Big|_{\text{res}} = \frac{\partial L}{\partial X_O^{(\ell)}} = \frac{\partial L}{\partial (X^{(\ell)}+X_O^{(\ell)})}$ can be obtained by Eq.(2). Let $X_{cat}^{(\ell)} := (X_0^{(\ell)}\| \ldots \|X_{11}^{(\ell)})$ and let $G_h := \frac{\partial L}{\partial X_h^{(\ell)}}$:

$$\frac{\partial L}{\partial X_{cat}^{(\ell)}} = \frac{\partial L}{\partial X_O^{(\ell)}}(W_O^{(\ell)})^\intercal = \left(\frac{\partial L}{\partial X_0^{(\ell)}}\|\frac{\partial L}{\partial X_1^{(\ell)}}\|...\|\frac{\partial L}{\partial X_{11}^{(\ell)}}\right) = (G_0\|G_1\|...\|G_{11}) .$$

Then we can obtain the gradients G_h for each $h \in [12]$.

1. From$X_h^{(\ell)} = U_h^{(\ell)}V_h^{(\ell)}$: $\quad \frac{\partial L}{\partial U_h^{(\ell)}} = G_h(V_h^{(\ell)})^\intercal, \qquad \frac{\partial L}{\partial V_h^{(\ell)}} = (U_h^{(\ell)})^\intercal G_h .$

1. For $\frac{\partial L}{\partial U_h^{(\ell)}}$, We apply again the double BSGS algorithm from [4, 32] to compute two matrices in (interleaved) column packing, and the result is in (interleaved) diagonal packing and is pre-rotated (similar to Algorithm 12).
2. For $\frac{\partial L}{\partial V_h^{(\ell)}}$, we first obtain $(U_h^{(\ell)})^\intercal$ by n_1 rotations. It is also in the (interleaved) diagonal packing and is pre-rotated for the next step. Next, we apply the BSGS matrix multiplication between $(U_h^{(\ell)})^\intercal$ and G_h. Note that matrices are already rotated.

2. Define: $S_h^{(\ell)}[i][j] := \|Q_h^{(\ell)}[i][:] - K_h^{(\ell)}[j][:]\|_2^2$, and $D_h^{(\ell)}[i][j] := \dfrac{\partial L}{\partial S_h^{(\ell)}[i][j]} = \dfrac{\partial L}{\partial U_h^{(\ell)}[i][j]} \cdot \sigma'(S_h^{(\ell)}[i][j])$.

3. Gradients w.r.t. $Q_h^{(\ell)}[i][:], K_h^{(\ell)}[j][:]$:

$$\frac{\partial L}{\partial Q_h^{(\ell)}[i][:]} = \sum_j 2D_h^{(\ell)}[i][j]\big(Q_h^{(\ell)}[i][:] - K_h^{(\ell)}[j][:]\big), \frac{\partial L}{\partial K_h^{(\ell)}[j][:]} = -\sum_i 2D_h^{(\ell)}[i][j]\big(Q_h^{(\ell)}[i][:] - K_h^{(\ell)}[j][:]\big) .$$

4. Matrix form:

$$\frac{\partial L}{\partial Q_h^{(\ell)}} = 2\Big(\mathrm{diag}(D_h^{(\ell)}\mathbf{1})\, Q_h^{(\ell)} - D_h^{(\ell)} K_h^{(\ell)}\Big), \frac{\partial L}{\partial K_h^{(\ell)}} = 2\Big(\mathrm{diag}((D_h^{(\ell)})^\intercal \mathbf{1})\, K_h^{(\ell)} - (D_h^{(\ell)})^\intercal Q_h^{(\ell)}\Big) .$$

1. All $\sigma'(S_h^{(\ell)}[i][j])$ are pre-stored from the forward pass. Together with $\frac{\partial L}{\partial U_h^{(\ell)}}$ from the previous step, we can obtain $D_h^{(\ell)}$, $(D_h^{(\ell)})^\intercal$ and $\mathrm{diag}(D_h^{(\ell)}\mathbf{1}), \mathrm{diag}((D_h^{(\ell)})^\intercal\mathbf{1})$.
2. $Q_h^{(\ell)}$ and $K_h^{(\ell)}$ are pre-stored from the forward pass. Combining together we can obtain $\frac{\partial L}{\partial Q_h^{(\ell)}}$ and $\frac{\partial L}{\partial K_h^{(\ell)}}$ for each $h \in [12]$.

5. Gradients w.r.t. head h**:** $\dfrac{\partial L}{\partial W_{Q,h}^{(\ell)}} = (X^{(\ell)})^\intercal \dfrac{\partial L}{\partial Q_h^{(\ell)}}, \dfrac{\partial L}{\partial W_{K,h}^{(\ell)}} = (X^{(\ell)})^\intercal \dfrac{\partial L}{\partial K_h^{(\ell)}}, \dfrac{\partial L}{\partial W_{V,h}^{(\ell)}} = (X^{(\ell)})^\intercal \dfrac{\partial L}{\partial V_h^{(\ell)}}$.

6. Gradients concatenated all heads $h \in [12]$**:**

$$\frac{\partial L}{\partial W_Q^{(\ell)}} = \left(\frac{\partial L}{\partial W_{Q,0}^{(\ell)}} \| \frac{\partial L}{\partial W_{Q,1}^{(\ell)}} \| ... \| \frac{\partial L}{\partial W_{Q,11}^{(\ell)}}\right) = (X^{(\ell)})^\intercal \left(\frac{\partial L}{\partial Q_0^{(\ell)}} \| \frac{\partial L}{\partial Q_1^{(\ell)}} \| ... \| \frac{\partial L}{\partial Q_{11}^{(\ell)}}\right) = (X^{(\ell)})^\intercal \frac{\partial L}{\partial Q^{(\ell)}} ,$$

$$\frac{\partial L}{\partial W_K^{(\ell)}} = \left(\frac{\partial L}{\partial W_{K,0}^{(\ell)}} \| \frac{\partial L}{\partial W_{K,1}^{(\ell)}} \| ... \| \frac{\partial L}{\partial W_{K,11}^{(\ell)}}\right) = (X^{(\ell)})^\intercal \left(\frac{\partial L}{\partial K_0^{(\ell)}} \| \frac{\partial L}{\partial K_1^{(\ell)}} \| ... \| \frac{\partial L}{\partial K_{11}^{(\ell)}}\right) = (X^{(\ell)})^\intercal \frac{\partial L}{\partial K^{(\ell)}} ,$$

$$\frac{\partial L}{\partial W_V^{(\ell)}} = \left(\frac{\partial L}{\partial W_{V,0}^{(\ell)}} \| \frac{\partial L}{\partial W_{V,1}^{(\ell)}} \| ... \| \frac{\partial L}{\partial W_{V,11}^{(\ell)}}\right) = (X^{(\ell)})^\intercal \left(\frac{\partial L}{\partial V_0^{(\ell)}} \| \frac{\partial L}{\partial V_1^{(\ell)}} \| ... \| \frac{\partial L}{\partial V_{11}^{(\ell)}}\right) = (X^{(\ell)})^\intercal \frac{\partial L}{\partial V^{(\ell)}} .$$

7. Gradient w.r.t. input $X^{(\ell)}$:

$$\frac{\partial L}{\partial X^{(\ell)}} = \frac{\partial L}{\partial X^{(\ell)}}\Big|_{\text{res}} + \frac{\partial L}{\partial Q^{(\ell)}}(W_{Q,frozen}^{(\ell)} + A_Q^{(\ell)} B_Q^{(\ell)})^\intercal + \frac{\partial L}{\partial K^{(\ell)}}(W_{K,frozen}^{(\ell)} + A_K^{(\ell)} B_K^{(\ell)})^\intercal + \frac{\partial L}{\partial V^{(\ell)}}(W_{V,frozen}^{(\ell)} + A_V^{(\ell)} B_V^{(\ell)})^\intercal .$$

- Note that (1) each of $\frac{\partial L}{\partial Q^{(\ell)}}, \frac{\partial L}{\partial K^{(\ell)}}, \frac{\partial L}{\partial V^{(\ell)}}$ has the same size and the same packing method as $\frac{\partial L}{\partial X^{(\ell)}}$; and (2) $(W_{*,frozen}^{(\ell)} + A_* B_*)^\intercal = (W_{*,frozen}^{(\ell)})^\intercal + (B_*)^\intercal (A_*)^\intercal$, where $* = Q, K, V$. We can use the same matrix multiplication code block from the forward pass for the computation of $\frac{\partial L}{\partial *^{(\ell)}}(W_{*,frozen}^{(\ell)} + A_*^{(\ell)} B_*^{(\ell)})^\intercal$, where $* = Q, K, V$.

8. Gradient w.r.t. LoRA matrices : $\dfrac{\partial L}{\partial A_*^{(\ell)}} = \dfrac{\partial L}{\partial W_*^{(\ell)}}(B_*^{(\ell)})^\intercal, \dfrac{\partial L}{\partial B_*^{(\ell)}} = (A_Q^{(\ell)})^\intercal \dfrac{\partial L}{\partial W_*^{(\ell)}}, * = Q, K, V$.

1. Note that $\frac{\partial L}{\partial W_*^{(\ell)}} = (X^{(\ell)})^\intercal \frac{\partial L}{\partial *^{(\ell)}}$, we have $\frac{\partial L}{\partial B_*^{(\ell)}} = (X^{(\ell)} A_*^{(\ell)})^\intercal \frac{\partial L}{\partial W_*^{(\ell)}}$. $X^{(\ell)} A_*^{(\ell)}$ is pre-stored from the forward pass.
2. In $\frac{\partial L}{\partial A_*^{(\ell)}} = (X^{(\ell)})^\intercal \frac{\partial L}{\partial *^{(\ell)}} (B_*^{(\ell)})^\intercal$, we first compute the latter multiplication $\frac{\partial L}{\partial *^{(\ell)}}(B_*^{(\ell)})^\intercal$ to reduce the number of ctct multiplications.

C.3 Update model weights

In this section, we describe how to compactly pack the gradients of all updatable parameters from Section C.2 into as few ciphertexts as possible. This is motivated by the fact that the number of bootstrapping operations required in the optimizer and during model weight updates is $\Theta(L)$, where L denotes the number of ciphertexts used to store these gradients.

Average. For each updatable matrix, the gradients have to be averaged across a batch of size b. These can be obtained by $O(\log b)$ FHE rotations and additions.

Packing compactly. To use the minimal number of ciphertexts, we can pack all the updatable weights compactly. In summary, all the updatable weights are matrices:

- $A_Q^{(\ell)}, A_K^{(\ell)}, A_V^{(\ell)} \in \mathbb{R}^{n_2 \times n_{\text{LoRA}}}$, $\ell = 0, 1$.
- $B_Q^{(\ell)}, B_K^{(\ell)}, B_V^{(\ell)} \in \mathbb{R}^{n_2 \times n_{\text{LoRA}}}$, $\ell = 0, 1$.
- $W_{c1} \in \mathbb{R}^{n_2 \times n_2}, W_{c2} \in \mathbb{R}^{n_2 \times c}$.

For example, given $n_2 = 768, n_{\text{LoRA}} = 2$ and $c = 2$, all the LoRA matrices and W_{c2} have totally $6 \times 768 \times 2 + 768 \times 2 = 10752 < N$. So we can pack all these matrices compactly in one CKKS ciphertext, by masking, rotation and addition. 9 ciphertexts are required to pack W_{c1} compactly.

In **Average** step and **Packing** step, it will cost totally 1 multiplicative level to pack all these matrices compactly. Therefore, we keep 1 multiplicative level in Figure 4.

Optimizer. We apply the AdamW [22] optimizer for better performance as suggested in [39], and use [28]'s FHE-friendly modification ($\sqrt{\hat{v}_t} + \epsilon \to \sqrt{\hat{v}_t + \epsilon}$). We include it in Algorithm 18 for completeness.

Algorithm 18 AdamW-HE [28]

Parameters: $\gamma, \beta_1, \beta_2, \epsilon > 0, \lambda$.
Initialize: $m_0 = 0$, $v_0 = 0$, $t = 1$.
In round t:
  Compute gradients of all updatable matrices and pack them compactly g_t in α ciphertexts.
  Let θ_t be all the updatable matrices (in compact form).
  Apply bootstrapping on g_t. ▷ to the highest multiplicative level
  $\theta_t := (1 - \gamma\lambda)\theta_{t-1}$.
  $m_t := \beta_1 m_{t-1} + (1 - \beta_1) g_t$, $v_t := \beta v_{t-1} + (1 - \beta_2)(g_t \odot g_t)$.
  Apply bootstrapping on m_t, v_t if the remaining multiplicative level is not enough.
  Store m_t, v_t for the next iteration.
  $\hat{m}_t := m_t/(1 - \beta_1^t)$, $\hat{v}_t := v_t/(1 - \beta_2^t)$.
  Update matrices: $\theta_t := \theta_t - \gamma \hat{m}_t/(\sqrt{\hat{v}_t + \epsilon})$.
  Apply bootstrapping on θ_t.

Repacking and updating. After homomorphically evaluate AdamW-HE and obtain θ_t, we repack all the updatable matrices from θ_t and derive all the updated matrices in our required form (e.g., the repeated row packing for LoRA matrices).

[28] uses 1 ciphertext for each updatable matrix except W_{c1}. By compactly packing the updatable matrices, the number of required ciphertexts in our scheme is reduced from 22 (if we follow [28]) to 10, which dramatically reduce the cost of bootstrapping.

D FHE Matrix Multiplications from [28]

D.1 CPMM and CCMM in [28]

In this section, we summarize the computational complexity of CPMM and CCMM in [28]. In [28], CPMM and CCMM involve (1) split the matrices as several block matrices which can be packed in a

single ciphertext; (2) call JKLS algorithm [16]. The number of each HE operation in JKLS algorithm is listed in Table 12. [12]

Table 12: The number of each HE operation for one $d \times d$ homomorphic matrix multiplication required by JKLS algorithm [16]

Operations	Rot	ct-ct Mult	ct-pt Mult	Level
CPMM in [16]	$2d + 3\sqrt{d}$	0	$4d$	3
CCMM in [16]	$3d + 5\sqrt{d}$	d	$4d$	3

For example, as mentioned in Section D.1 of [28], when performing homomorphic matrix multiplication of size 128×768 by 768×768, it will first split the matrices as three and eighteen block matrices, respectively, then go through each block matrix multiplication to get a corresponding result. As the same left matrices are used several times to multiply by the right block matrices, it can compute and save the left matrices, so the total number of JKLS algorithm calls in this example is 21, instead of 36.

In Table 13, we summarize the required number of each HE operation for $\mathrm{Enc}(X)W$ in [28]. X is split into three block matrices and W is split into eighteen block matrices, with block shapes 128×256 and 256×128, respectively. With pre-computation and reuse for the matrix X, the number of rectangular JKLS calls is reduced from 36 to 21 (see Section D.1 in [28]). Each call has input matrices with shape $l \times d$ by $d \times l$, where $l = 128$ and $d = 256$.

Table 13: The number of each HE operation for CPMM ($\mathrm{Enc}(X)W$) for a single input in [28].

Operations	Rot	ct-ct Mult	Relinearization	ct-pt Mult	Level
CPMM in [28]	$21\left(2l + 3\sqrt{d} + \log_2(d/l)\right)$	0	0	$21(3d + 2l)$	3

In Table 14, we summarize the required number of each HE operation for $\{\mathrm{Enc}(\sigma(S_h))\,\mathrm{Enc}(V_h)\}_{h\in[H]}$ in [28], where $S_h \in \mathbb{R}^{n_1 \times n_1}$, and $V_h \in \mathbb{R}^{n_1 \times (n_3)}$. Using [28]'s notation, let $m := n_1$ and $g := H/2$. In the BERT-style setting, $m = 128$ and $g = 6$. The value multiplication uses g packed groups. Each packed group consists of complex packing, one packed matrix multiplication, and repacking; the whole value path additionally uses $g/2$ masking and $g/2$ left/right collection additions.

Table 14: The number of each HE operation for value CCMM ($\{\mathrm{Enc}(\sigma(S_h))\,\mathrm{Enc}(V_h)\}_{h\in[H]}$) for a single input in [28], following the GitHub implementation.

Operations	Rot	ct-ct Mult	Relinearization	ct-pt Mult	Level
Value CCMM in [28]	$g\left(\frac{3m}{2} + 4\lceil\sqrt{m}\rceil + 2\right)$	$\frac{mg}{2}$	$2g$	$g(3m + 1)$	3

Furthermore, [28] proposed an optimized LoRA CCMMs (Algorithm 5 in [28]) to reduce the required number of HE operations and multiplicative levels. The required number of each HE operation for $(\mathrm{Enc}(X)\,\mathrm{Enc}(A))\,\mathrm{Enc}(B)$, where $X \in \mathbb{R}^{128\times768}$, $A \in \mathbb{R}^{768\times n_{\mathrm{LoRA}}}$, and $B \in \mathbb{R}^{n_{\mathrm{LoRA}}\times768}$ (n_{LoRA} is the LoRA rank), is listed in Table 15. [13]

D.2 FHE squared l_2-norm computation in [28]

In this section we include the FHE algorithms in [28] for computing the square matrix S_h given input as encrypted Q_h, K_h in head h. More concretely, in each head h they compute

$$U_h[i][j] := \exp\left(-\frac{1}{2\sqrt{n}}\|Q_h[i][:] - K_h[j][:]\|_2^2\right),$$

[12] The number of each HE operation of CCMM is taken from Table 2 in JKLS [16]. The number of each HE operation of CPMM is derived by considering B as a plaintext matrix and adjusting the number of each HE operation in step 1-2, step 2 and step 3 in Table 2.

[13] Numbers in the table is taken from Table 9 in [28].

Table 15: The number of each HE operation for LoRA-style CCMMs for a single input in [28].

Operations	Rot	ct-ct Mult	Relinearization	ct-pt Mult	Level
LoRA-style CCMMs	$40n_{\mathrm{LoRA}} - 1$	$6n_{\mathrm{LoRA}}$	$6n_{\mathrm{LoRA}}$	$4n_{\mathrm{LoRA}}$	3

where $Q, K \in \mathbb{R}^{n_1 \times n_3}$, $Q_h[i][:], K_h[j][:] \in \mathbb{R}^{n_3}$ represent the i-th row of Q and the j-th row of K respectively. [28] did not provide the algorithm of computing $\exp\left(-\frac{1}{2\sqrt{n}}\|Q_h[i][:] - K_h[j][:]\|_2^2\right)$ in their paper. This section is following their GitHub implementation.

[28] packs multiple heads together in computing the Gaussian kernel. Let the query and key matrices be $\bar{Q}, \bar{K}$ which have shape $n_1 \times n_2$. Therefore, packing $\bar{Q}$ or $\bar{K}$ will require $s := \lceil \frac{n_1 n_2}{N} \rceil$ ciphertexts. More concretely, with the assumption $n_1 | N$, [28] first divide $\bar{Q}$ into $s := \lceil \frac{n_1 n_2}{N} \rceil$ sub-matrices, each of which has size $n_1 \times \frac{N}{n_1}$. Then we can write $\bar{Q} := (\bar{Q}_0 \| \bar{Q}_1 \| ... \| \bar{Q}_{s-1})$, where $\bar{Q}_i$ has size $n_1 \times \frac{N}{n_1}$ and $i \in [s]$. The notations are similar for $\bar{K} := (\bar{K}_0 \| \bar{K}_1 \| ... \| \bar{K}_{s-1})$. [28] also assume that $2n_3 | \frac{N}{n_1}$ so that each head will be packed and can be computed within the same ciphertext.

The matrix packing of [28]. [28] follows the *row-wise packing* from [16]. In this packing, each row of the matrix is *sequentially* packed into a single long vector with size $N/2$. When the number of entries of the matrix exceeds the number of ciphertext slots, multiple ciphertexts are used. To pack $\bar{Q}$, [28] uses row-wise packing for each $\bar{Q}_i, \bar{K}_i, i \in [s]$. Each head has n_3 columns and the number of heads is $H = \frac{n_2}{n_3}$. Let $t := \frac{N}{2n_1 n_3}$. Each matrix $\bar{Q}_i$ or $\bar{K}_i$ will have t heads packed. For any $i \in [s]$, Q_i, K_i contain $2t$ heads each:

$$\bar{Q}_i = (Q_{2it} \| Q_{2it+1} \| ... \| Q_{2it+2t-1})$$

$$\bar{K}_i = (K_{2it} \| K_{2it+1} \| ... \| K_{2it+2t-1})$$

Please refer to Algorithm 19 for the detailed algorithm of [28]. All the matrices are in the encrypted form. For the sake of simplicity, we avoid using the notation $\mathrm{Enc}(\cdot)$ in Algorithm 19. Their computations are based on the decomposition:

$$\|Q_h[i][:] - K_h[j][:]\|_2^2 = \|Q_h[i][:]\|_2^2 + \|K_h[j][:]\|_2^2 - 2Q_h[i][:]K_h^\intercal[j][:] \ .$$

Analysis. For each $i \in [s]$:

- Generating $\{Q_{i,L,sum}, Q_{i,R,sum}, K_{i,L,sum}, K_{i,R,sum}\}$ costs $4(\log n_3 + \log(n_3/2))$ rotations, 4 ct-ct multiplications and 8 ct-pt multiplications.
- Generating $\{Q_{i,L,\mathbb{C}}, Q_{i,R,\mathbb{C}}, K_{i,L,\mathbb{C}}, K_{i,R,\mathbb{C}}\}$ costs 8 rotations, 0 ct-ct multiplications and 12 ct-pt multiplications.
- 4 matrix transposition are required, each costs $2\sqrt{n_1 + n_3} - 1$ rotations and $n_1 + n_3 - 1$ ct-pt multiplications.
- 2 matrix multiplications to generate $Q_{i,L,\mathbb{C}} K_{i,L,\mathbb{C}}^T, Q_{i,R,\mathbb{C}} K_{i,R,\mathbb{C}}^T$, each costs $\frac{3n_3}{2} - 3 + 4\sqrt{n_3}$ rotations, $n_3/2$ ct-ct multiplications (with lazy relinearization) and $3n_3 - 5$ ct-pt multiplications.
- Extracting $Q_{i,L} K_{i,L}^T, Q_{i,R} K_{i,R}^T$ costs 2 conjugations, which has similar cost as HE rotations.

In summary, the required number of each HE operation for kernel computation is listed in Table 16.

Table 16: The number of each HE operation for Kernel CCMM for a single input in [28].

Operations	Rot	ct-ct Mult	Relinearization	ct-pt Mult	Level
Kernel CCMM in [28]	$s\big[4\log_2(n_3^2/2) + 8\lceil\sqrt{n_1 + n_3}\rceil + 3n_3 + 8\sqrt{n_3}\big]$	$s\,(4 + n_3)$	$8s$	$s(4n_1 + 10n_3 + 6)$	3

Algorithm 19 Kernel computation ($\{S_h\}_{h\in[H]}$) in [28].

Input: Encrypted row-packed matrices $\{\bar{Q}_i, \bar{K}_i\}_{i\in[s]}$.
Output: Encrypted row-packed matrices S.

For any $i \in [s]$, split Q_i and K_i into two parts: ▷ Cost 1 level

$$\bar{Q}_i = Q_{i,L} + Q_{i,R} := (Q_{2it}\|\boldsymbol{O}\|...\|Q_{2it+2t-2}\|\boldsymbol{O}) + (\boldsymbol{O}\|Q_{2it+1}\|...\|\boldsymbol{O}\|Q_{2it+2t-1})$$

$$\bar{K}_i = K_{i,L} + K_{i,R} := (K_{2it}\|\boldsymbol{O}\|...\|K_{2it+2t-2}\|\boldsymbol{O}) + (\boldsymbol{O}\|K_{2it+1}\|...\|\boldsymbol{O}\|K_{2it+2t-1})$$

Step I: compute the square l_2-norm for each row in $Q_{i,L}, Q_{i,R}, K_{i,L}, K_{i,R}, i \in [s]$:
Each head has shape $n_1 \times n_3$.
$Q_{i,L,sum} := Q_{i,L} \odot Q_{i,L}$, $Q_{i,R,sum} := Q_{i,R} \odot Q_{i,R}$. ▷ Rotate and Sum.
for $col \in \{1, 2, 4, ..., n_3/2\}$ **do**
  $tmp :=$ Left rotate the ciphertext of $Q_{i,L,sum}$ for col slots.
  $Q_{i,L,sum} += tmp$.
  $tmp :=$ Right rotate the ciphertext of $Q_{i,R,sum}$ for col slots.
  $Q_{i,R,sum} += tmp$.
end for
Masking:
  Set $Q_{i,L,sum}$ to 0 except the $2n_3l$-th column, $l \in [t]$. ▷ Cost 1 level
  Set $Q_{i,R,sum}$ to 0 except the $(2n_3(l+1)-1)$-th column, $l \in [t]$. ▷ Cost 1 level
for $col \in \{1, 2, 4, ..., n_3\}$ **do**
  $tmp :=$ Right rotate the ciphertext of $Q_{i,L,sum}$ for col slots.
  $Q_{i,L,sum} += tmp$.
  $tmp :=$ Left rotate the ciphertext of $Q_{i,R,sum}$ for col slots.
  $Q_{i,R,sum} += tmp$.
end for
Compute $K_{i,L,sum}, K_{i,R,sum}$ using the same steps.
Transpose $K_{i,L,sum}, K_{i,R,sum}$ by [16].
Step II: Compute $Q_{i,L}K_{i,L}^T$:
Repack the columns in $Q_{i,L}$ in complex number field $\mathbb{C}$:
  $Q_{i,L,\mathbb{C}} := Q_{i,L}$.
for $j \in [\frac{N}{n_1}]$ **do**
  The $2j$-th column of $Q_{i,L,\mathbb{C}}$:= the $2j$-th column of $Q_{i,L}$ + $\big($the $(2j+1)$-th column of $Q_{i,L}\big)\sqrt{-1}$. ▷ Cost 1 level
  The $2j+1$-th column of $Q_{i,L,\mathbb{C}}$:= the $2j$-th column of $Q_{i,L,\mathbb{C}}$.
end for
Repack the columns in $K_{i,L}$ in complex number field $\mathbb{C}$:
  $K_{i,L,\mathbb{C}} := K_{i,L}$.
for $j \in [\frac{N}{n_1}]$ **do**
  The $2j$-th column of $K_{i,L,\mathbb{C}}$:= the $2j$-th column of $K_{i,L}$ − $\big($the $(2j+1)$-th column of $K_{i,L}\big)\sqrt{-1}$.
  The $2j+1$-th column of $K_{i,L,\mathbb{C}}$:= the $2j$-th column of $K_{i,L,\mathbb{C}}$.
end for
Transpose $K_{i,L,\mathbb{C}}$ by [16].
A modified matrix multiplication from [16] to compute $Q_{i,L,\mathbb{C}}K_{i,L,\mathbb{C}}^T$.
$Q_{i,L}K_{i,L}^T := \mathrm{Re}(Q_{i,L,\mathbb{C}}K_{i,L,\mathbb{C}}^T)$. ▷ Cost 1 level
Similarly, compute $Q_{i,R,\mathbb{C}}K_{i,R,\mathbb{C}}^T$.
Return $2s$ ciphertexts:

$$Q_{i,L,sum} + K_{i,L,sum} - 2 \times Q_{i,L}K_{i,L}^T, Q_{i,R,sum} + K_{i,R,sum} - 2 \times Q_{i,R}K_{i,R}^T, \quad \forall i \in [s]\ .$$

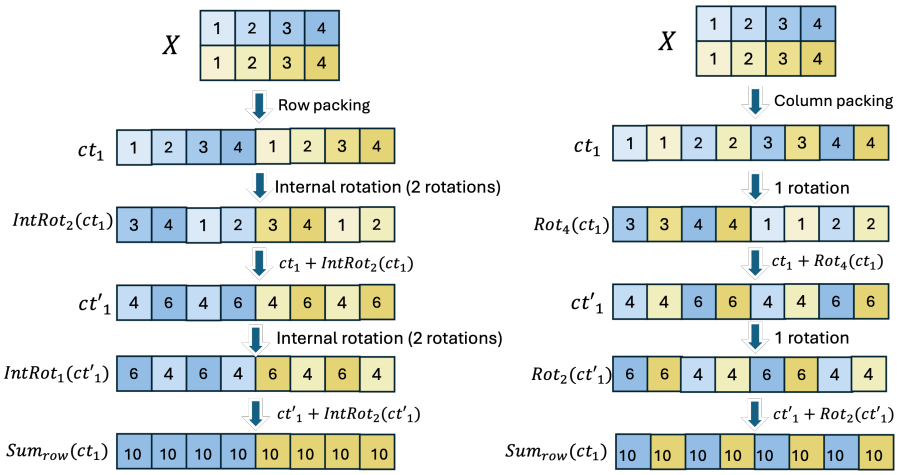

Figure 3: Toy example of summing elements within each row under row packing and column packing

E Details of non-linear evaluations

E.1 Layernorm evaluations

Our Layernorm evaluation algorithm takes a set of ciphertexts in flexible interleaved column packing that encrypts $\{X^i\}_{i\in[b]}$, where b is the batch size as input, and outputs a set of ciphertexts with the same packing format. Figure 3 illustrates a toy example of summing elements within each row under row packing [28] and our packing scheme. [14] In [28], a generic row-packing strategy is applied across transformer operations, which requires "internal rotations" within each row for such evaluations. Each internal rotation is implemented via two ciphertext rotations, leading to a $2\times$ rotation cost compared to our column-based packing, which avoids such internal rotations.

For higher precision of the inverse square root approximation, we perform the following transformation to $\mathrm{LayerNorm}$ for vector $\mathbf{x}$:

$$\begin{aligned}\mathrm{LayerNorm}(\mathbf{x})[i] &= \gamma_i \frac{x_i - \mathrm{mean}(\mathbf{x})}{\sqrt{\mathrm{var}(\mathbf{x})}} + \beta_i \\ &= \frac{\gamma_i}{\sqrt{c}} \frac{x_i - \mathrm{mean}(\mathbf{x})}{\sqrt{\mathrm{var}(\mathbf{x})/c}} + \beta_i\end{aligned}$$

where c (as in [36, 28] is a constant to ensure the input to the inverse square root is in an appropriate range.

As discussed in [38], achieving sufficiently accurate Layernorm typically requires around 20 multiplicative levels, which exceeds our setting where only 12 levels remain after bootstrapping. To address this, we introduce two additional bootstrapping steps: one applied to $\mathrm{var}(\mathbf{x})/c$ before evaluating the inverse square root, and another within the inverse square root computation itself, ensuring sufficient multiplicative depth for accurate evaluation. Notably, both bootstrapping operations are performed on a single aggregated ciphertext (after summation across inputs), rather than on all input ciphertexts. As a result, each Layernorm evaluation incurs a runtime of less than 2 seconds.

E.2 Functional bootstrapping for element-wise functions

RBOOT framework [33] accelerates secure neural network inference by fusing the evaluation of non-linear activation functions like ReLU directly into the CKKS bootstrapping process. The core innovation lies in exploiting the trigonometric structure of the modular reduction circuit. Normal CKKS bootstrapping usually uses $\sin(\cdot)$ and $\arcsin(\cdot)$ to approximate mod function $x(\mathrm{mod}\ q_0)$.

[14] For simplicity, the example considers a single input and omits interleaved batching. As discussed in our matrix operations, interleaved batching can further reduce the number of required rotations.

For example in ReLU function, the evaluation from RBOOT is integrated into the bootstrapping process using the following relationship for $x \in [-1/4, 1/4] + \mathbb{Z}$:

$$\text{ReLU}(x(\text{mod } 1)) = \frac{1}{4\pi} \arcsin(\sin(2\pi x)) + \frac{1}{8\pi} \arcsin(-\cos(4\pi x)) + \frac{1}{16} \tag{3}$$

As $\sin(\cdot)$ and $\cos(\cdot)$ can be obtained in bootstrapping process, and $\arcsin(\cdot)$ is approximated by polynomial already, we can obtain the result of ReLU using Eq. (3). Reported by [33], this reduces the multiplicative level consumption of ReLU evaluation in a targeted precision from 14 to 20 levels (typically required by high-degree polynomial approximations) down to approximately 8 levels. In general if we want to evaluate function $f(\cdot)$, we can replace $\arcsin(\cdot)$'s polynomial approximation with specialized minimax polynomials that approximate the composition of $f(cdot)$ and $\arcsin(\cdot)$.

F Full flow of FELID

Figure 4 illustrates the complete diagram of our evaluation flow to fine-tuning a 2-layer BERT-style model. The blue boxes are CPMM, while the green boxes are CKKS functional bootstrapping using [33]. The changes of multiplicative levels are indicated in each node. Finally, all the yellow boxes are the gradients of updatable weights. They will be sent to compact packing process and the AdamW-HE optimizer [28].

G Experimental details

G.1 Extended Details about Experimental Setup

G.1.1 Continued-pretraining.

For our kernel-attention models, we initialize from the same pretrained BERT checkpoint and replace softmax attention with either GK or EK. Instead of training a kernelized backbone from scratch, we start from the pretrained softmax BERT checkpoint and adapt only the components directly affected by the attention replacement. Specifically, we update the Q/K/V projections and kernel-specific parameters, while keeping the embeddings, feed-forward layers, and layer-normalization parameters frozen. The continued-pretraining objective combines masked language modeling with distillation from the original softmax BERT teacher, using hidden-state mean-squared error and logits KL divergence. Both our GK and EK models are trained for 500k steps. In our implementation, this continued-pretraining stage takes approximately 30–60 minutes, avoiding the need to train a kernelized backbone from scratch.

G.1.2 Justifications

Scope justification. We limit our primary evaluation to a 2-layer BERT-style architecture, matching the model size used in the baseline [28]. We do not modify the feedforward blocks or the classification head.[15]

This scope is architecturally sufficient because a 2-layer model encapsulates all fundamental interactions within transformer architecture, including the all the interactions between multi-head attention block, feedforward block, and sequential layer normalization blocks [21, 11, 28]. Consequently, our improvements demonstrated at this scale are operationally fundamental and reflect the effectiveness of our operation-level and algorithmic optimizations. Furthermore, because transformer layers are evaluated sequentially and refreshed via bootstrapping to a fixed level at the start of each block, the total latency is strictly linear in the number of layers. As the SOTA FHE fine-tuning baseline [28] already shows that the single-epoch latency reach hundreds of hours for tasks in GLUE, we believe that it is sufficient to demonstrate our optimizations on a 2-layer model.

Task justification. While the SOTA baseline of FHE fine-tuning [28] evaluates six GLUE tasks, we limit our primary analysis to 3 tasks: **STS-B**, **MRPC**, and **RTE**, due to the extremely high

[15] In contrast, [28] introduces two architectural modifications that we do not adopt: (1) splitting the output of the first feedforward layer into two parts and applying activation to only one part; and (2) replacing the first matrix in the classification head with two dense blocks of LoRA-style shapes with rank 32.

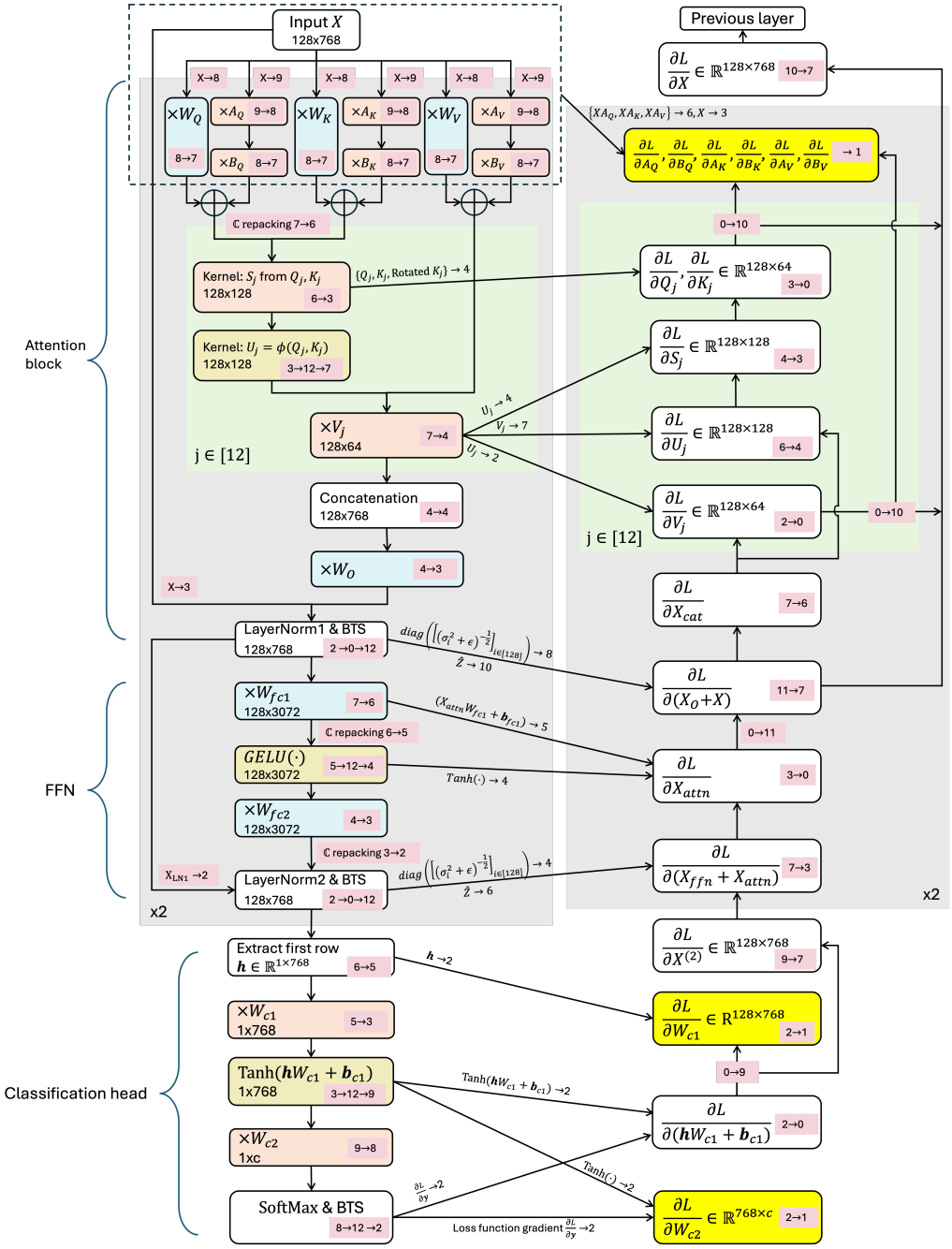

Figure 4: Full flow of our framework.

computational overhead of the other tasks. According to Table 3 in [28], fine-tuning for **COLA**, **SST-2**, and **QNLI** on 8-GPU cluster requires approximately 82.5 hours (for 5 epochs), 130 hours (1 epoch), and 202 hours (1 epoch), respectively. Such durations are even more computationally impractical for our single-GPU environment.

Furthermore, focusing on these three representative tasks is well justified, as prior studies consistently show that FHE introduces negligible numerical error and therefore has minimal impact on downstream performance. Frameworks such as [36, 25, 27] have demonstrated that encrypted results remain closely aligned with plaintext counterparts.

Limitations. Despite the proposed optimizations, while FHE enables non-interactive, privacy-preserving transformer fine-tuning, it remains computationally intensive and typically requires high-performance hardware (e.g., multiple GPUs) to mitigate latency, especially for large model and datasets.

G.2 PhantomFHE vs. HEaaN in [28]

The implementation in [28] is based on the proprietary (non-open-sourced) HEaaN library and evaluated on an NVIDIA GeForce RTX 4090 GPU, whereas our implementation is built on PhantomFHE and runs on an NVIDIA H200 GPU.

To provide a more controlled comparison, in Table 17, we benchmark the primitive operations of the CKKS scheme in both libraries under same parameters.[16]

Table 17: Runtime (ms) of basic operations in CKKS with parameters: degree of polynomial ring = 2^{16}, number of slots = 2^{15}, multiplicative level before bootstrapping = 9, security parameter = 128-bit. The result of HEaaN reported by [28].

Library	GPU	Add	PMult	Mult	Rot^{HE}	BTS/ ExtBTS	FuncBTS
HEaaN	Nvidia GeForce RTX 4090	0.02	0.1	0.6	0.49	60/137	-
PhantomFHE	NVIDIA H200	0.03	0.06	3.13	2.87	476	398
Ratio (PhantomFHE/HEaaN)		$1.5\times$	$0.6\times$	$5.2\times$	$5.8\times$	$3.5-7.9\times$	-

Under same CKKS parameters, PhantomFHE on H200 achieves comparable or faster performance for simple operations (e.g., plaintext multiplication), but incurs significantly higher costs for expensive operations: (1) being about $5-6\times$ slower for ciphertext multiplication and rotations and (2) $3.5-7.9\times$ slower for bootstrapping, compared to HEaaN on RTX 4090. The difference reflects the effects of hardware and library-level optimization differences, which are not the focus of this work.

[16] The HEaaN results are taken from Table 5 of [28]. BTS, ExtBTS, and FuncBTS denote bootstrapping, extended bootstrapping (only in HEaaN), and functional bootstrapping (only in our implementation), respectively.

G.3 Breakdown of FELID encrypted fine-tuning

Table 18 lists the runtime of all fine-tuning operations over a batch of input in 1 layer, including forward pass, backward pass and optimizer.

Table 18: Breakdown fine-tuning time (seconds) over a batch of input with batch size = 16.

Block	Operation	Input size	Time (s)
Forward: Transformer Layer	$\times W_Q, W_K, W_V$	$(128 \times 768) \times (768 \times 768) \times 3$	5.8
	$\times A*, \times B*, * = Q, K, V$	$(128 \times 768) \times (768 \times 2) \times (2 \times 768) \times 3$	0.45
	LoRA addition	$((128 \times 768) + (128 \times 768)) \times 3$	0.001
	Repack($\mathbb{C} \to \mathbb{R}$)	$(128 \times 64) \times 12 \times 2$	0.013
	Kernel S	$(128 \times 64) \times 12 \times 2$	5.97
	Kernel U (FuncBTS)	$(128 \times 128) \times 12$	38.31
	$\times V$	$(128 \times 128) \times (128 \times 64) \times 12$	4.28
	Concatenation	$(128 \times 64) \times 12$	0.071
	$\times W_o$	$(128 \times 768) \times (768 \times 768)$	2.05
	Repack($\mathbb{C} \to \mathbb{R}$)	128×768	0.005
	Layernorm	128×768	1.95
	Bootstrapping	128×768	11.45
	$\times W_{fc1}$	$(128 \times 768) \times (768 \times 3072)$	7.6
	Repack($\mathbb{C} \to \mathbb{R}$)	128×3072	0.023
	GELU (FuncBTS)	128×3072	85.75
	$\times W_{fc2}$	$(128 \times 3072) \times (3072 \times 768)$	7.2
	Repack($\mathbb{C} \to \mathbb{R}$)	128×768	0.025
	Layernorm	128×768	1.9
	Bootstrapping	128×768	11.53
Forward: Classification Head	Extract the first row	128×768	0.02
	$\times W_{c1}$	$(1 \times 768) \times (768 \times 768)$	0.16
	Tanh (FuncBTS)	1×768	0.67
	$\times W_{c2}$	$(1 \times 768) \times (768 \times c)$	0.008
	Softmax	$1 \times c$	1.86
	Bootstrapping	$1 \times c$	1.05
Backward: Classification Head	$\partial L / \partial W_{c2}$	$1 \times c$	0.004
	$\partial L / \partial (\mathbf{h} W_{c1} + \mathbf{b}_{c1})$	1×768 & $1 \times c$	0.007
	$\partial L / \partial W_{c1}$	$(768 \times 1) \times (1 \times 768)$	0.52
	$\partial L / \partial X^{(2)}$	(1×768)	0.27
Backward: Transformer Layer	$\partial L / \partial (X_{ffn} + X_{att})$	128×768	0.067
	$\partial L / \partial X_{att}$	128×768&$(128 \times 3072) \times 2$	26.7
	Bootstrapping	128×768	11.45
	$\partial L / \partial (X_o + X)$	128×768	0.097
	$\partial L / \partial X_{cat}$	128×768	3.73
	$\partial L / \partial V$	$(128 \times 128) \times (128 \times 64) \times 12$	9.82
	$\partial L / \partial U$	$(128 \times 128) \times (128 \times 64) \times 12$	5.5
	$\partial L / \partial S$	$(128 \times 128) \times 2 \times 12$	0.65
	$\partial L / \partial Q$	$(128 \times 128) \times (128 \times 64) \times 12$	0.87
	$\partial L / \partial K$	$(128 \times 128) \times (128 \times 64) \times 12$	11.8
	Bootstrapping	$(128 \times 768) \times 3$	33.43
	$\partial L / \partial A*, * = Q, K, V$	$(128 \times 768)^\intercal \times (128 \times 768) \times (768 \times 2)^\intercal \times 3$	3.7
	$\partial L / \partial B*, * = Q, K, V$	$(128 \times 2)^\intercal \times (128 \times 768) \times 3$	1.8
	$\partial L / \partial X$	$(128 \times 768) \times 4$	35.7
Optimizer	AdamW-HE	$(768 \times c)$, (768×768), $(768 \times 2) \times 3$, $(2 \times 768) \times 3$	19.9

G.4 End-to-end results across multiple batch sizes

We conducted additional experiments across batch sizes 4, 16, and 64 to showcase that the proposed flexible packing method is well-suited for transformer fine-tuning.

Matrix operations and vector-wise function. The amortized time (Table 19 decreases when batch size increases, which is consistent to our complexity overhead analysis in Appendix B and Section 3.3. Note that for element-wise functions such as exp and GELU, we evaluate them by functional encryption, and the amortized time is independent of the batch size.

Table 19: Time costs (sec) of matrix operations and vector-wise function LayerNorm across multiple batch sizes.

Operation	Batch Size 4		Batch Size 16		Batch Size 64	
	Total Time	Amortized	Total Time	Amortized	Total Time	Amortized
CPMM	0.26	0.065	0.97	0.061	3.11	0.049
CCMM-LoRA	0.018	0.0045	0.067	0.0042	0.169	0.0026
CCMM-kernel	0.88	0.22	3.42	0.21	10.2	0.16
CCMM-V	0.86	0.22	3.24	0.20	10.32	0.16
LayerNorm	1.02	0.26	1.90	0.12	6.02	0.094

2-layer BERT-style model. Across all tested batch sizes, our packing uses all available plaintext slots in the ciphertexts. The results in Table 20 show that the amortized latency decreases from 45.32s at batch size 4 to 36.67s at batch size 64. We further show that noise remains very close across all settings (Table 21).

Table 20: Time costs (sec) in 2-layer BERT-style model across multiple batch sizes.

	Sub-operation	Batch Size 4	Batch Size 16	Batch Size 64
Forward pass	Matrix	17.10	67.79	216.90
	Non-linear	4.88	9.56	28.80
	(Func) Boot	73.80	295.05	1180.20
Backward pass	Matrix	41.50	164.04	524.87
	Non-linear	1.02	1.91	6.02
	(Func) Boot	23.20	92.45	369.60
Optimizer	-	19.76	19.90	20.24
Total	-	181.26	650.60	2346.63
Amortized	-	45.32	40.66	36.67

Table 21: Mean and max of absolute noise across different batch sizes.

	Batch Size 4	Batch Size 16	Batch Size 64
Mean of Absolute Noise	2.32×10^{-8}	2.54×10^{-8}	2.64×10^{-8}
Max of Absolute Noise	1.45×10^{-6}	1.60×10^{-6}	1.68×10^{-6}

G.5 Additional cumulative component-level breakdown

We in Section 4.1 report separate improvements for matrix operations, functional bootstrapping, and bootstrapping counts. To makes the attribution clearer, we provide the following cumulative component-level breakdown in Table 22. In summary:

- Encrypted matrix evaluation contributes to largest absolute runtime reduction.
- Flexible interleaved packing and operation-specific algorithms substantially reduce expensive rotations and relinearizations (Table 2 in paper).
- Lower multiplicative depth and fewer required bootstrapping invocations reduce bootstrapping time.
- Functional bootstrapping further reduces the non-linear evaluation cost by folding element-wise function evaluation into the bootstrapping step.

G.6 Experiments across multiple hidden dimensions

We include in Table 23 and Table 24 that the amplification trend observed in Table 15, [28]. We additionally evaluate the amortized runtime with batch size 4 across multiple hidden dimensions. For CPMM and CCMM, we test them with $\mathbb{R}^{128\times d} \times \mathbb{R}^{d\times d}$. The results show that as the dimension d increases, the performance gains from LoRA fine-tuning become larger.

Table 22: Cumulative component-level breakdown of each operation, in batch size 16.

Operation	[28]	FELID **(Ours)**	**Reduction**	**Main Cause**
Forward , Matrix	1429	67.79	21.1×	Packing + matrix algorithms
Forward , Non-linear	291	9.56	30.4×	Functional bootstrapping
Forward , (Func) Boot	708	295.05	2.4×	Reduced depth + fewer bootstrappings
Backward , Matrix	2377	164.04	14.5×	Packing + matrix algorithms
Backward , Non-linear	47	1.91	24.6×	Functional bootstrapping
Backward , (Func) Boot	355	92.45	3.8×	Reduced depth + fewer bootstrappings
Optimizer	31	19.90	1.6×	Compact packing (Appendix C.3)
Total	5238	650.60	8×	Combined effect

We also follow the setting in Table 15 of [28] and evaluate the scaling behavior of Gaussian-kernel attention and softmax attention on a matrix in $\mathbb{R}^{n\times n}$. In FELID, the exponential function in Gaussian-kernel attention is evaluated by functional bootstrapping, which refreshes the ciphertext and evaluates the element-wise function in a single step. In contrast, softmax is a vector-wise function and cannot be directly evaluated by functional bootstrapping. Therefore, for a fair comparison, we compare Gaussian-kernel functional bootstrapping with the full "softmax evaluation + bootstrapping" flow, where both settings produce refreshed ciphertexts after function evaluation. The results also show that as the dimension n increases, the performance gains from GK become larger.

Table 23: Operation time cost (s) in LoRA fine-tuning across multiple hidden dimensions.

Dim(d)	**256**	**512**	**768**
CPMM	0.007	0.027	0.061
CCMM	0.067	0.39	1.13
Factor	9.57x	14.4x	18.5x

Table 24: Gaussian-kernel attention and softmax attention time cost (sec), across multiple dimensions.

Dim(n)	**128**	**256**	**512**
GK (Functional bootstrapping)	0.3	0.6	2.39
Softmax+bootstrapping	0.735	1.726	8.95
Factor	2.45x	2.88x	3.74x

G.7 CKKS approximation error during full training

We performed an additional noise-propagation experiment to evaluate how CKKS approximation error accumulates through a full training step. we use the trained 2-layer Gaussian-kernel checkpoint and cyclically replicate its layers to construct deeper models. we run forward pass, backward pass, and AdamW-HE update on the first batch of MRPC, RTE, and STS-B with batch size 16, and report the mean and maximum value of the absolute error (Table 25). These results show that FELID is robust to CKKS approximation error in the evaluated full-step training flow. The error does not blow up through forward propagation, backward propagation, or encrypted optimizer updates, and remains small compared with the scale of model activations and updates.

Table 25: The mean and maximum of the absolute error during full training.

Number of layers	**2**	**4**	**8**	**12**
MRPC	Mean: 2.54×10^{-8} Max: 1.9×10^{-6}	Mean: 3.14×10^{-8} Max: 2.9×10^{-6}	Mean: 5.47×10^{-8} Max: 9.8×10^{-6}	Mean: 3.03×10^{-7} Max: 8.6×10^{-5}
RTE	Mean: 1.96×10^{-8} Max: 1.6×10^{-6}	Mean: 2.63×10^{-8} Max: 1.4×10^{-6}	Mean: 6.97×10^{-8} Max: 5.0×10^{-5}	Mean: 2.63×10^{-7} Max: 9.3×10^{-5}
STS-B	Mean: 5.51×10^{-7} Max: 5.1×10^{-5}	Mean: 7.18×10^{-7} Max: 7.4×10^{-5}	Mean: 7.48×10^{-7} Max: 7.5×10^{-5}	Mean: 1.14×10^{-6} Max: 1.6×10^{-4}

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The main claims in the abstract are verified in Section 3.2.1, Section 4.1, and Section 4.2, respectively.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We discussed the limitations of the work in Section G.1.2.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

Justification: We listed all theoretical analysis of computation complexity in Section B, and summarize the result in Table 2.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Our implementation of proposed algorithms are attached in the GitHub link. Detailed descriptions (pseudo code) of proposed algorithms are presented in Section B.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: Our implementation of proposed algorithms are attached in the GitHub link.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: We describe the model, datasets, parameters and underlying libraries in Section 4 Experimental results and Section G.1.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: The experiments are repeated many times so that we report the average results.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)

- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: All experiments used a single NVIDIA H200 GPU, as described in Section 4.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research conducted in the paper confirm, in every aspect, with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: We discussed the potential positive societal impacts of privacy-preserving machine learning in Section 1.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.

- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The answer NA means that the paper poses no such risks.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We cite the original paper or open-source library that produced the code package or dataset.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.

- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: The paper does not release new assets.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: This paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: This paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.

- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: In this work, LLM is used only for editing grammar, spelling and word choice. LLM does not impact the core methodology, scientific rigorousness, or originality of the research.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.