

ROSETTA: Efficient and Accurate Privacy-Preserving LLM Decoding via Hybrid CKKS/TFHE Evaluation

Jiangrui Yu
Peking University
Beijing, China
jiangrui.yu@stu.pku.edu.cn

Baosheng Zhang^{*}
Xi'an Jiaotong University
Xi'an, China
2831532315@stu.xjtu.edu.cn

Liang Kong
Ant Group
Beijing, China
kongliang.kong@antgroup.com

Lin Ding
Peking University
Beijing, China
dinglin@pku.edu.cn

Yi Chen
Peking University
Beijing, China
yichen25@stu.pku.edu.cn

Ye Yu
Peking University
Beijing, China
yy3628@columbia.edu

Mingzhe Zhang
Ant Group
Beijing, China
smartzmz@gmail.com

Meng Li[†]
Peking University
Beijing, China
meng.li@pku.edu.cn

Abstract

Generative large language models (LLMs) have achieved state-of-the-art performance on many real-world tasks such as code generation and question answering. These models predominantly rely on an autoregressive decoding strategy that generates output tokens sequentially. However, their pervasive deployment raises serious privacy concerns, motivating private inference frameworks based on fully homomorphic encryption (FHE). A major limitation of existing FHE frameworks is their inefficiency in evaluating nonlinear operations, which incur substantial overhead and dominate the decode stage.

In this paper, we propose ROSETTA, a hybrid CKKS/TFHE framework that overcomes this limitation. We first observe that nonlinear operations in the decode stage exhibit heterogeneous workload patterns, which can be handled effectively via a hybrid approach. We then realize this with two key contributions: 1) an adaptive segmented lookup-table protocol based on TFHE that enables efficient and accurate evaluation of nonlinear operations; and 2) a scheme-aware operator-selection framework that automatically assigns each nonlinear operator to CKKS or TFHE to minimize end-to-end decoding latency. We demonstrate that ROSETTA achieves up to 4.8× Softmax speedup and 1.5–2.1× end-to-end speedup over the SOTA framework CacheMir.

CCS Concepts

• **Security and privacy** → **Cryptography**; **Public key encryption**; **Privacy-preserving protocols**; • **Computing methodologies** → **Machine learning**.

^{*}This work was completed while Baosheng Zhang was an intern at Peking University.
[†]Corresponding author.



Keywords

fully homomorphic encryption, CKKS, TFHE, privacy-preserving inference, large language model, programmable bootstrapping

ACM Reference Format:

Jiangrui Yu, Baosheng Zhang, Liang Kong, Lin Ding, Yi Chen, Ye Yu, Mingzhe Zhang, and Meng Li. 2026. ROSETTA: Efficient and Accurate Privacy-Preserving LLM Decoding via Hybrid CKKS/TFHE Evaluation. In *Proceedings of the 2026 ACM SIGSAC Conference on Computer and Communications Security (CCS '26)*, November 15–19, 2026, The Hague, Netherlands. ACM, New York, NY, USA, 15 pages. <https://doi.org/10.1145/3830454.3846549>

1 Introduction

Generative large language models (LLMs), such as GPT [42] and LLaMA [44], have demonstrated impressive performance across a broad range of applications, such as clinical diagnostics [43], financial analysis [17], document summarization [32], and intelligent voice assistants [10]. These models typically execute on cloud platforms and process user prompts that may contain highly sensitive information. Consequently, privacy has become a fundamental concern for the deployment of LLM inference [24, 26, 39, 49].

To address this challenge, private inference frameworks have been proposed to protect both proprietary model weights and user inputs throughout inference. Specifically, the client learns nothing beyond the final output, while the server obtains no information about the input data. Existing systems have explored cryptographic frameworks based on fully homomorphic encryption (FHE) [9, 16, 19, 21, 31, 38, 41, 53, 59, 60], secure multi-party computation (MPC) [2, 18, 23, 30, 34, 56–58], and hybrid FHE/MPC designs [24–26, 29, 37, 39, 47, 48, 50, 51, 54, 62, 64]. Among these designs, FHE-based inference achieves higher communication efficiency and significantly reduces the computational burden on the client. As a result, FHE-centric approaches are more practical for end users: the client simply uploads encrypted inputs, the untrusted server carries out all subsequent computations directly over ciphertexts, and only the encrypted output is sent back, as shown in Figure 1(a).

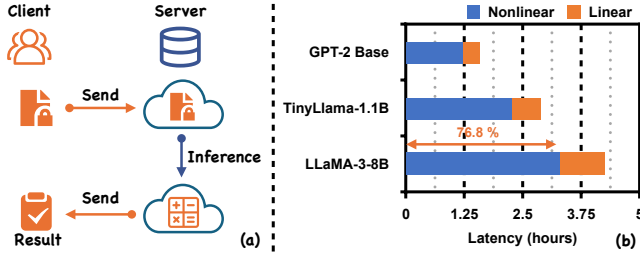


Figure 1: (a) Illustration of FHE-based private inference. (b) Latency breakdown for GPT-2 Base, TinyLlama-1.1B, and LLaMA-3-8B based on CacheMir [55].

Despite these benefits, building a fully end-to-end LLM over FHE remains substantially challenging. In particular, generative LLM inference generally involves two main stages: prefill and decode. The prefill stage processes the entire input prompt, typically hundreds to thousands of tokens, in parallel. This naturally fits SIMD-style schemes like CKKS [11] and has been well optimized by pure-CKKS frameworks [38, 41, 59, 60]. The decode stage, in contrast, processes only one token at a time, potentially yielding low data parallelism for some operators, which can leave many SIMD slots underutilized with pure-CKKS frameworks. Although CacheMir [55] has optimized its linear components, nonlinear operators remain the dominant source of latency, accounting for more than **70%** of the decode stage total runtime, as illustrated in Figure 1(b).

To meet such disparate demands, a natural idea is to leverage SISC-style schemes like TFHE [13] for nonlinear operators. TFHE supports programmable bootstrapping (PBS), a lightweight and flexible primitive that evaluates arbitrary functions through a look-up table (LUT). Operating on a per-element basis, PBS does not rely on SIMD batching and remains efficient when nonlinear operators receive only a few elements. Hybrid CKKS/TFHE frameworks such as PEGASUS [27] explore this direction by offloading **all** nonlinear layers to TFHE: they first convert CKKS ciphertexts to TFHE, evaluate all nonlinear operators via PBS, and finally convert the results back to CKKS for subsequent linear layers.

However, we observe that nonlinear operators in the decode stage are highly heterogeneous along two dimensions, **element count** and **input dynamic range**, and a one-size-fits-all approach is suboptimal. Take softmax as an example (Figure 2(a) and (b)): it comprises two nonlinear operators, $\exp$ and $1/x$. The $\exp$ operator processes n elements with a bounded input range, whereas the subsequent summation produces a single scalar with a much wider range fed into $1/x$. Evaluating both operators under CKKS, as in pure-CKKS frameworks [38, 55, 59, 60], handles $\exp$ efficiently with a low-degree polynomial approximation under SIMD batching (Figure 2(c)). However, to achieve enough precision over the wide dynamic range of $1/x$, iterative methods such as Goldschmidt require many rounds, each consuming multiplicative levels and triggering frequent bootstrapping, yielding a total latency of 44.1s.

Evaluating both operators under TFHE instead (Figure 2(d)) faces two fundamental issues. *First*, $\exp$ over n elements is inefficient because TFHE’s element-wise PBS cannot exploit batch parallelism, and the two scheme conversions add further overhead, incurring 57.9s in total. *Second*, although PBS is efficient on $1/x$, whose input

is a scalar, a single PBS encodes only a small LUT (e.g., 2,048 entries), capping precision at ~ 11 bits, which is insufficient for the wide dynamic range and can inflate the softmax perplexity by over 100% relative to the plaintext baseline (Table 8), rendering the efficient computation meaningless. *In summary, neither scheme alone is sufficient. CKKS achieves high precision but at high cost and is restricted to wide SIMD batching, whereas TFHE flexibly evaluates per-element workloads but cannot deliver high precision.*

Therefore, their complementarity suggests a per-operator hybrid. As shown in Figure 2(e), selectively offloading only $1/x$ to TFHE while keeping $\exp$ under CKKS combines the strengths of both schemes and reduces the total latency to 22.2s. However, realizing this idea raises two challenges: **(i)** the TFHE side still inherits PBS’s precision limitation on wide-range operators such as $1/x$; and **(ii)** the per-operator scheme assignment is itself non-trivial: offloading an operator to TFHE not only changes its own latency, but also introduces scheme conversion overhead, while the levels of the surrounding CKKS layers must be set accordingly. It is therefore insufficient to just compare each operator’s own latency across the two schemes alone to get the optimal assignment. Existing frameworks ignore this and resort to coarse heuristics, such as offloading all nonlinear layers (PEGASUS) or only the first k layers (LOHEN [63]), yielding suboptimal performance. Both challenges must be resolved before selective offloading can deliver efficiency and precision. Table 1 summarizes the limitations of existing frameworks along these dimensions.

1.1 Contribution

To resolve the above challenges, we propose ROSETTA, a hybrid CKKS/TFHE framework for efficient and accurate privacy-preserving LLM decoding, with two techniques each targeting one of the above challenges.

① Adaptive Segmented LUT Protocol. To address challenge (i), we propose an Adaptive Segmented LUT Protocol that delivers high-precision nonlinear evaluation while remaining lightweight under TFHE. The protocol partitions the input domain into non-uniform segments whose boundaries adapt to the function’s local behavior, with each segment carrying its own LUT and piecewise-linear interpolation, concentrating LUT entries where the function varies most rapidly. To evaluate the LUT homomorphically, we design a three-step protocol that (1) compares the encrypted input against every segment boundary and sums the homomorphic comparison results to derive the encrypted segment index, (2) multiplies the encrypted input with precomputed coefficient polynomials and uses a PBS to derive the interval index, and (3) similarly multiplies the input with packed slope/offset polynomials and uses PBS to produce the linear evaluation. The protocol relies only on RLWE plaintext-ciphertext multiplications and PBS lookups, avoiding ciphertext-ciphertext fixed-point arithmetic that TFHE does not natively support, and delivers 12–19 bit precision over wide input ranges, exceeding the ~ 11 -bit ceiling of single-PBS approaches by 6–10 bits and running 8–9× faster than CKKS-based iterative methods (§3).

② Scheme-Aware Operator Selection. To address challenge (ii), we develop a framework that jointly optimizes per-operator scheme assignment and CKKS multiplicative-level allocation, extending

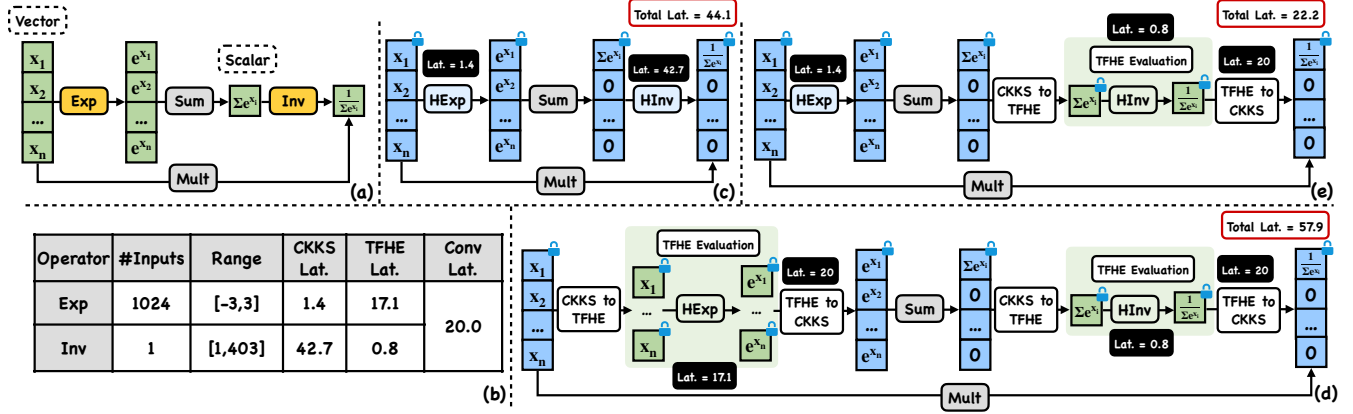


Figure 2: Softmax evaluation under different FHE schemes ($n=1024$). (a) Plaintext softmax pipeline. (b) Measured per-module latency under CKKS or TFHE. (c)–(e) Three schemes for evaluating the nonlinear operators: (c) all under CKKS, (d) all under TFHE, and (e) a hybrid assignment. “Lat.” and “Conv.” denote latency and scheme conversion, respectively.

Table 1: ROSETTA vs. prior FHE-based nonlinear evaluation frameworks. “Poly.” denotes polynomial approximation; “Seg. LUT” denotes segmented LUT.

Feature	Pure-CKKS [38, 55, 59]	PEGASUS [27]	LOHEN [63]	ROSETTA (Ours)
Scheme	CKKS	Hybrid	Hybrid	Hybrid
Evaluator	Poly.	Single LUT	Single LUT	Seg. LUT + Poly.
TFHE precision	—	Low	Low	High
Workload size	Large	Small	Small	Both
Selection strategy	All CKKS	All TFHE	First- k	Per-op*

* Jointly optimises per-operator scheme assignment and CKKS-level allocation.

CacheMir’s pure-CKKS shortest-path formulation [55] to scheme-aware decisions. Our method is driven by two key observations. First, each nonlinear layer decomposes into a mix of arithmetic and non-arithmetic primitives, and only non-arithmetic primitives (e.g., exp , $1/x$) benefit from TFHE evaluation. We therefore restrict scheme assignment to non-arithmetic primitives, dramatically shrinking the search space while leaving CKKS-friendly arithmetic operations untouched. Second, the cost of every CKKS–TFHE conversion can be expressed as an edge weight on the existing level-allocation DAG, so injecting a level-0 TFHE node alongside each non-arithmetic primitive turns the joint optimization into a single shortest-path problem on the augmented DAG. The shortest path simultaneously identifies which non-arithmetic primitives to offload and the optimal CKKS level for every layer, yielding a globally optimal schedule without separate scheme-selection and level-allocation passes (§4).

Overall, we make the following contributions:

- We design an adaptive segmented LUT protocol that achieves high-precision nonlinear evaluation over wide input ranges (§3).
- We formulate the joint optimization of TFHE operator selection and CKKS level assignment as a shortest-path problem on a DAG, enabling the proper FHE scheme selection of operators (§4).

- We evaluate ROSETTA against the SOTA pure-CKKS method CacheMir [55], and hybrid CKKS/TFHE method PEGASUS [27]. Compared to CacheMir, ROSETTA achieves up to $4.8\times$ Softmax speedup and $1.5\text{--}2.1\times$ end-to-end speedup. Compared to PEGASUS, ROSETTA achieves $3\text{--}4\times$ end-to-end speedup while reducing average PPL degradation from $+116\%$ to only $+1.4\%$.

2 Background

2.1 Notation

We use bold lower-case letters to represent vectors (e.g., $\mathbf{m}$), with $\mathbf{m}[i]$ denoting the i -th element, and hatted lower-case letters for polynomials (e.g., $\hat{m}(X)$). We write $R_{N,Q} = \mathbb{Z}_Q[X]/(X^N + 1)$ for the $2N$ -th cyclotomic ring modulo Q , and $\mathbb{Z}_q^{n_{\text{LWE}}+1}$ for an $(n_{\text{LWE}} + 1)$ -dimensional vector modulo q .

2.2 FHE Schemes

In this work, we primarily use two types of FHE schemes: CKKS [12] and TFHE [13], which can both be instantiated over standard lattice hardness problems. In what follows, we sketch the main operators for each scheme.

2.2.1 The CKKS Scheme. The CKKS scheme is constructed over the Ring Learning with Errors (RLWE) problem. It natively encodes and encrypts multiple data elements (referred to as a *packed* ciphertext) into a single RLWE ciphertext polynomial. Here, we summarize the key operators over CKKS used in this paper.

- **Encryption and Decryption:** Given a vector of plaintext messages $\mathbf{m} \in \mathbb{C}^{N/2}$, we have

$$[\mathbf{m}]_{\text{CKKS}} \leftarrow \text{CKKS-Enc}(\mathbf{m}), \quad (1)$$

where $[\mathbf{m}]_{\text{CKKS}} \in R_{N,Q}^2$ [12], for N the ring dimension and Q the ciphertext modulus, and $\mathbf{m} \approx \text{CKKS-Dec}([\mathbf{m}]_{\text{CKKS}})$. We omit how $\mathbf{m}$ is encoded into polynomials, as these details are orthogonal to our protocol.

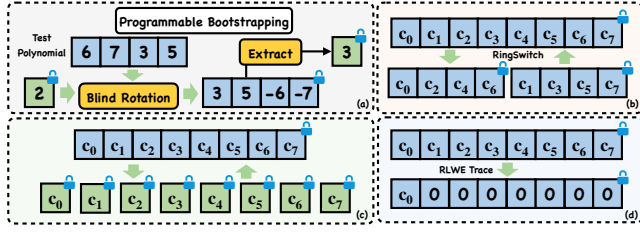


Figure 3: Operations used in our protocol: (a) PBS, (b) ring switch, (c) LWE extract / RLWE repack, (d) RLWE trace.

- **Addition and Multiplication:** CKKS ciphertexts are compatible with element-wise ciphertext addition and multiplication, e.g.,

$$[[m_0 + m_1]]_{\text{CKKS}} = \text{CKKS-Add}([m_0]_{\text{CKKS}}, [m_1]_{\text{CKKS}}). \quad (2)$$

2.2.2 TFHE Scheme. The TFHE scheme is primarily constructed over the standard Learning with Errors (LWE) problem. It natively encrypts a single scalar data element into an LWE ciphertext. It consists of the following operators:

- **Encryption and Decryption:** Given a plaintext message $m \in \mathbb{Z}_q$, we have

$$[[m]]_{\text{TFHE}} \leftarrow \text{TFHE-Enc}(m), \quad (3)$$

where $[[m]]_{\text{TFHE}} \in \mathbb{Z}_q^{n_{\text{LWE}}+1}$ [13] is an LWE ciphertext. Here, n_{LWE} denotes the LWE dimension, and q denotes the ciphertext modulus. Decryption recovers $m = \text{TFHE-Dec}([m]_{\text{TFHE}})$.

- **Addition, Subtraction, and Constant Multiplication:** TFHE ciphertexts are compatible with standard ciphertext addition, subtraction, and constant multiplication. For brevity, we take only the addition operator as an example, where it holds that

$$[[m_0 + m_1]]_{\text{TFHE}} = \text{TFHE-Add}([m_0]_{\text{TFHE}}, [m_1]_{\text{TFHE}}). \quad (4)$$

- **Programmable Bootstrapping (PBS)** [13]: PBS can homomorphically evaluate a look-up table (LUT) $f : \mathbb{Z}_q \rightarrow \mathbb{Z}_q$ while simultaneously refreshing the ciphertext noise:

$$[[f(m)]]_{\text{TFHE}} = \text{TFHE-LUT}_f([m]_{\text{TFHE}}). \quad (5)$$

This relies on a core primitive known as *blind rotation*. To evaluate the LUT f , its entries are encoded directly into the coefficients of a *test polynomial* $\hat{v}(X) \in R_{N_{\text{br}}, Q_{\text{br}}}$. Here, N_{br} denotes the ring degree (which corresponds exactly to the total number of LUT entries), and Q_{br} represents the ciphertext modulus. The blind rotation mechanism utilizes the encrypted TFHE message m as a shift amount to cyclically left-rotate the test polynomial. This operation yields an RLWE $_{Q_{\text{br}}}^{N_{\text{br}}}$ encryption of the shifted polynomial (where RLWE denotes a standard Ring-LWE ciphertext). By extracting the constant term of it, we obtain a fresh TFHE ciphertext encrypting the target value $f(m)$.

This blind rotation mechanism is illustrated in Figure 3(a). As shown, the LUT entries $\{f(0), f(1), \dots\}$ are laid out as the polynomial coefficients (e.g., 6, 7, 3, 5). The blind rotation cyclically shifts these coefficients according to the encrypted index m . For instance, given an encrypted value $m = 2$, the polynomial is left-rotated by 2 positions, moving the target entry $f(2) = 3$ directly into the constant-term position for subsequent extraction.

2.2.3 Scheme Conversion and Other Primitives. To change the underlying FHE scheme during the process of homomorphic evaluation, scheme conversion operators seamlessly bridge packed RLWE ciphertexts and scalar LWE ciphertexts [7, 27]. Below, we introduce the conversion operations utilized in our framework.

- **Ring Switch** [4, 20]: As shown in Figure 3(b), a ring switch is an operation that performs transitions between rings with different degrees N and N_{br} (large/small rings). The conversion can be processed in both directions by splitting one large ciphertext into multiple small ones, or the other way around. In our protocol, we specifically adopt the ring switching methodology from [20].
- **LWE Extract and RLWE Repack:** [8, 13] As depicted in Figure 3(c), *LWE Extract* [13] extracts each coefficient of a packed RLWE ciphertext into independent LWE scalar ciphertexts, and *RLWE Repack* [8] performs the inverse, packing scalar LWE ciphertexts back into a single RLWE ciphertext.
- **RLWE Trace Computation** [8]: As illustrated in Figure 3(d), the trace operation isolates the constant term c_0 of an RLWE ciphertext encrypting $a(X) = \sum_{i=0}^{N-1} c_i X^i$. Through a logarithmic sequence of Galois automorphisms, homomorphic additions, and a multiplication by N^{-1} , all non-constant terms cancel, leaving a ciphertext that holds only c_0 .

2.3 Modern LLM Architecture

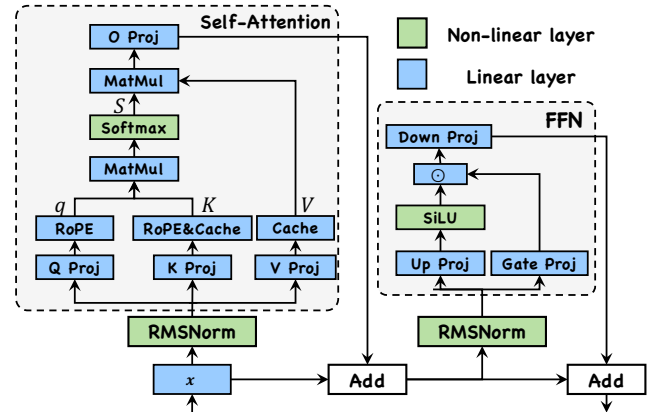


Figure 4: A modern Transformer block in LLaMA [44].

Modern LLM architecture is based on the Transformer [45], which consists of two components: the self-attention block and the feed-forward network (FFN). In this work, we focus on the decode stage of LLM and give a brief introduction below. We denote the model's hidden dimension as d_{model} (e.g., 4096), the number of attention heads as h (e.g., 32), the per-head dimension as $d = d_{\text{model}}/h$ (e.g., 128), the FFN intermediate dimension as d_{ffn} (e.g., 11008), and the current context length as n .

Self-Attention Block. Given a single-token input $\mathbf{x} \in \mathbb{R}^{1 \times d_{\text{model}}}$, it is multiplied with three weight matrices to produce the query, key, and value vectors: $q = \mathbf{x}W_Q$, $k = \mathbf{x}W_K$, and $v = \mathbf{x}W_V$, all in $\mathbb{R}^{1 \times d}$. We refer to such operation, where the activation is multiplied by a weight matrix, as a *projection*. The new key and value are

appended to the KV-cache from previous tokens to form $K \in \mathbb{R}^{n \times d}$ and $V \in \mathbb{R}^{n \times d}$. The self-attention is then computed as:

$$\mathbf{s} = \text{Softmax}\left(\frac{qK^\top}{\sqrt{d}}\right) \in \mathbb{R}^{1 \times n}, \quad \mathbf{att} = \mathbf{s} \times V \in \mathbb{R}^{1 \times d} \quad (6)$$

The attention output $\mathbf{att}$ from all h heads is concatenated and passed through an output projection $W_O \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$ to obtain the final output of the attention block.

FFN. The FFN layer is implemented using the SwiGLU activation. The input $\mathbf{x} \in \mathbb{R}^{1 \times d_{\text{model}}}$ is first projected through both the up projection $W_{\text{up}} \in \mathbb{R}^{d_{\text{model}} \times d_{\text{ffn}}}$ and the gate projection $W_{\text{gate}} \in \mathbb{R}^{d_{\text{model}} \times d_{\text{ffn}}}$, followed by a SiLU activation, and finally passed through the down projection $W_{\text{down}} \in \mathbb{R}^{d_{\text{ffn}} \times d_{\text{model}}}$ to produce the output:

$$\text{FFN}(\mathbf{x}) = \left(\text{SiLU}(\mathbf{x}W_{\text{up}}) \odot (\mathbf{x}W_{\text{gate}}) \right) W_{\text{down}} \quad (7)$$

Max-free Softmax. Let $\mathbf{z}$ denote the softmax input. Since the standard max-subtraction is expensive in FHE, recent FHE frameworks [38, 55, 59] adopt a *normalize-and-square* formulation: starting from $y_i = e^{z_i/2^k}$ on a small range, the identity $\text{Softmax}(2\mathbf{x})_i = y_i^2 / \sum_j y_j^2$ is applied k times to recover $\text{Softmax}(\mathbf{z})$. We follow this formulation throughout.

2.4 Threat Model

ROSETTA operates in a standard two-party private inference setting with a server and a client. The server hosts a proprietary LLM with confidential model weights, while the client holds sensitive input data [28, 33, 35]. The protocol guarantees that the client obtains correct inference outputs while keeping both the server’s model weights and the client’s input private. Following prior work [28, 33, 35], we assume the model architecture is publicly known to both parties and operate under the honest-but-curious security model, wherein both parties adhere to the prescribed protocol but may attempt to learn additional information beyond what is permitted.

3 Segmented LUT Evaluation

The observations from §1 drive us to design the ROSETTA Segmented LUT protocol, which enables accurate and efficient nonlinear function evaluation through a lightweight LUT-based method. We first present the limitations of the single-LUT approach and introduce the Adaptive Segmented LUT that achieves high precision efficiently (§3.1). We then realize this construction as a cryptographic lookup protocol (§3.2), which is seamlessly integrated into the LLM decoding flow via FHE scheme conversion (§3.3).

3.1 Adaptive Segmented LUT Construction

We start with the conventional single-LUT approach, which corresponds to the capability of a single TFHE PBS operation. Throughout this section, we use the evaluation of $f(x) = 1/x$ over the input domain $[\tau_{\text{start}}, \tau_{\text{end}}] = [0.1, 2.0]$ via a LUT with $N_{\text{br}} = 4$ entries as a running example. As illustrated in Figures 5(a) and (b), a standard LUT divides the domain into N_{br} equal intervals, approximating the function within each interval by its value at the midpoint. For instance, the first interval $[0.1, 0.575]$ has a midpoint of 0.338; thus, $f(0.338) = 2.963$ is stored as the first LUT entry.

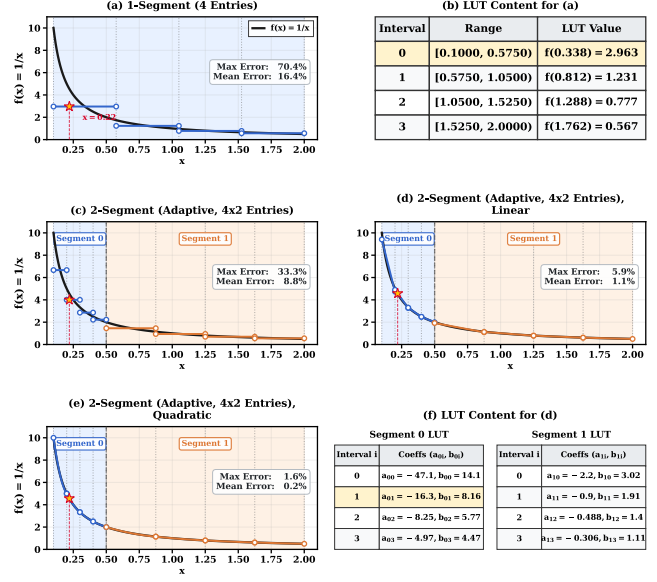


Figure 5: Segmented LUT for $f(x)=1/x$: (a) single-segment baseline with (b) its table; (c)–(e) two-segment variants using constant/linear/quadratic approximations; (f) slope/offset coefficients (a_{si}, b_{si}) of the linear variant in (d).

To evaluate an input such as $x = 0.22$, we first determine which interval it falls into, denoted by index I . Because the intervals are equally sized, the index is proportional to the input’s distance from τ_{start} . Thus, we directly compute I via a linear mapping:

$$I = \left\lfloor \frac{x - \tau_{\text{start}}}{\tau_{\text{end}} - \tau_{\text{start}}} \cdot N_{\text{br}} \right\rfloor = \left\lfloor \frac{0.22 - 0.1}{2.0 - 0.1} \cdot 4 \right\rfloor = 0$$

Using this index ($I = 0$), we fetch the pre-computed value from the table. As shown by the yellow row in Figure 5(b), the returned result is the midpoint approximation $f(0.338) = 2.963$. Overall, this basic LUT method produces a “staircase” approximation of the original function $f(x)$.

However, this single-LUT approach suffers from two main weaknesses. First, constrained by the limited capacity, a single table partitions the entire domain into only N_{br} equally spaced intervals, making each interval too wide to accurately capture rapid function variations. Second, within each interval, the function is approximated by a single constant (the midpoint value), which is accurate only near the midpoint; inputs far from it suffer from large errors. Together, these constraints significantly degrade approximation accuracy. As illustrated in Figure 5(a), the baseline LUT evaluates $f(0.22)$ to only 2.963, whereas the exact value is 4.545. This yields a 34.8% relative error at this evaluation point, and the maximum error reaches 70.4% across the entire domain.

3.1.1 Adaptive LUT Construction. To address the two aforementioned weaknesses, the table construction is optimized across two dimensions. First, to overcome the capacity limitation, an adaptive multi-segment LUT approach is adopted, as illustrated in Figure 5(c). Rather than uniformly spanning the entire domain with a single table, the domain is adaptively partitioned into $N_{\text{seg}} = 2$ segments,

delimited by $N_{\text{seg}} + 1$ boundaries $\tau_0 = \tau_{\text{start}} < \tau_1 < \dots < \tau_{N_{\text{seg}}} = \tau_{\text{end}}$, with a dedicated LUT assigned to each segment. In our example, the domain $[0.1, 2.0]$ is split into a narrow first segment $[\tau_0, \tau_1] = [0.1, 0.5]$ and a broader second segment $[\tau_1, \tau_2] = [0.5, 2.0]$. This non-uniform partition is because the function $f(x) = 1/x$ is steep near zero, necessitating a higher density of entries to maintain accuracy, whereas for larger inputs the curve flattens out and requires far fewer entries. By adapting the boundary spacing to the function's curve, the construction concentrates entries where the function varies most rapidly and reduces the maximum error to 33.3%.

Second, to mitigate the inaccuracies of a constant-value representation, each interval approximates $f(x)$ by a linear function $f(x) \approx a_{si} \cdot x + b_{si}$, where the slope a_{si} and offset b_{si} are precomputed and stored for interval i of segment s (Figure 5(d, f)). For example, given $x = 0.22$ in segment $s = 0$, interval $i = 1$ (i.e., $[0.2, 0.3]$), the stored coefficients are $a_{01} = -16.327$ and $b_{01} = 8.163$, yielding $f(0.22) \approx -16.327 \times 0.22 + 8.163 = 4.571$, close to the true value 4.545. Compared to the constant-value baseline, this linear model tracks the curve continuously within each interval, reducing the maximum error to 5.9%. While higher-order models such as quadratic approximation can further improve precision (Figure 5(e)), we adopt the linear variant as it delivers sufficient accuracy at minimal cost.

3.1.2 Online Lookup Evaluation. After constructing the adaptive LUT offline, an online lookup algorithm is required to evaluate the table for a given query x . We first demonstrate a cleartext algorithm and extend it to the fully encrypted version in §3.2. The lookup proceeds by first locating which segment and interval the query x falls in, denoted by the segment index S and local interval index I , then retrieving the corresponding coefficients and performing a linear evaluation. Accordingly, the workflow is divided into three steps: 1) Segment Index Lookup, 2) Local Interval Index Computation, and 3) Coefficient Lookup and Linear Evaluation.

Step 1: Segment Index Lookup. Given an input x , the segment index $S \in \{0, \dots, N_{\text{seg}} - 1\}$ is derived by comparing x against each segment boundary τ_j and summing the results:

$$S = \sum_{j=1}^{N_{\text{seg}}-1} \mathbb{I}(x \geq \tau_j).$$

For instance, the example input $x = 0.22$ is compared with the boundary $\tau_1 = 0.5$. Because $x < 0.5$, the indicator sum yields $S = 0$, indicating that the input falls into the 0-th segment.

Step 2: Local Interval Index Computation. Given the segment index S , the local interval index I within that specific segment is computed via a linear mapping, identical to the single-LUT approach:

$$I = \left\lfloor \frac{x - \tau_S}{\tau_{S+1} - \tau_S} \cdot N_{\text{br}} \right\rfloor = \left\lfloor \frac{0.22 - 0.1}{0.5 - 0.1} \cdot 4 \right\rfloor = \lfloor 1.2 \rfloor = 1$$

Thus, the example input $x = 0.22$ maps to the local Interval 1 (representing the sub-range $[0.2, 0.3]$) within Segment 0.

Step 3: Linear Evaluation. Using the derived indices (S, I) , the corresponding linear coefficients (a_{SI}, b_{SI}) are fetched to compute the approximation $f(x) \approx a_{SI} \cdot x + b_{SI}$. In the running example, fetching the parameters for Segment 0 and Interval 1 yields $a_{01} =$

-16.327 and $b_{01} = 8.163$. This produces $f(0.22) \approx -16.327 \cdot 0.22 + 8.163 = 4.571$, close to the true value 4.545. The relative error drops to 0.57%, significantly outperforming the 34.8% baseline.

3.2 Cryptographic Lookup Protocol

We now describe how to realize the three-step lookup under FHE. Step 1 is direct: each boundary comparison $x \geq \tau_j$ is evaluated via a homomorphic comparison primitive and the results are summed. However, Steps 2 and 3 pose the main challenge: both steps require multiplying the ciphertext $\llbracket x \rrbracket$ with real-valued parameters that depend on the segment index S and local interval index I , such as the linear coefficients (a_{SI}, b_{SI}) in Step 3. Since both indices are computed from the encrypted input $\llbracket x \rrbracket$, they also remain in ciphertext form as $\llbracket S \rrbracket_{\text{LWE}}$ and $\llbracket I \rrbracket_{\text{LWE}}$, and the server cannot directly extract the needed coefficients as in the cleartext setting. A naïve solution would use PBS to look up the required parameters with $\llbracket S \rrbracket_{\text{LWE}}$ and $\llbracket I \rrbracket_{\text{LWE}}$, but this yields them as LWE ciphertexts (e.g., $\llbracket a_{SI} \rrbracket_{\text{LWE}}$). The subsequent multiplication $\llbracket x \rrbracket_{\text{LWE}} \cdot \llbracket a_{SI} \rrbracket_{\text{LWE}}$ would then demand high-precision fixed-point ciphertext-ciphertext arithmetic, which TFHE does not natively support.

To overcome this, our protocol adopts an *online-construct-then-select* strategy (Figure 6). Instead of first looking up the coefficients (a_{SI}, b_{SI}) and then computing $a_{SI} \cdot x + b_{SI}$, our protocol inverts the computation order: it first computes all candidate results via RLWE plaintext-ciphertext multiplication to construct a ciphertext test polynomial, then selects the correct one via PBS, thereby avoiding fixed-point arithmetic on LWE ciphertexts entirely. Concretely, the server encodes all coefficients (a_{si}, b_{si}) for every segment s and interval i into plaintext polynomials and multiplies them with the ciphertext $\llbracket x \rrbracket_{\text{RLWE}}$, producing a ciphertext whose coefficients contain all candidate evaluations $a_{si} \cdot x + b_{si}$. The encrypted indices $\llbracket S \rrbracket_{\text{LWE}}$ and $\llbracket I \rrbracket_{\text{LWE}}$ are then used via PBS to select the correct result $a_{SI} \cdot x + b_{SI}$.

Based on this strategy, we now describe the protocol in detail, using the running example from §3.1 ($N_{\text{seg}} = 2$, $N_{\text{br}} = 4$, $(\tau_0, \tau_1, \tau_2) = (0.1, 0.5, 2.0)$, $x = 0.22$), as illustrated in Figure 6. The encrypted input is provided in two forms: an RLWE ciphertext $\llbracket x \rrbracket_{\text{RLWE}}$ (with x encoded in the constant coefficient) for the plaintext-ciphertext multiplications, and an LWE ciphertext $\llbracket x \rrbracket_{\text{LWE}}$ for the homomorphic comparisons in Step 1. The preparation of both ciphertexts is detailed later.

Step 1: Segment Index Lookup. This step directly follows the cleartext algorithm, which evaluates $\llbracket S \rrbracket_{\text{LWE}} = \llbracket \sum_{j=1}^{N_{\text{seg}}-1} \mathbb{I}(x \geq \tau_j) \rrbracket_{\text{LWE}}$. Each comparison $x \geq \tau_j$ is evaluated homomorphically using the high-precision comparison protocol HomComp from [5], which internally chains several PBS operations for sufficient precision. The $N_{\text{seg}} - 1$ comparison results, each an LWE ciphertext $\llbracket x \geq \tau_j \rrbracket_{\text{LWE}}$, are then summed via homomorphic addition to produce the encrypted segment index $\llbracket S \rrbracket_{\text{LWE}}$. This step requires $N_{\text{seg}} - 1$ invocations of HomComp in total. In the running example (Figure 6(a)), a single HomComp($x, \tau_1=0.5$) is evaluated; since $x = 0.22 < 0.5$, the result is $\llbracket 0 \rrbracket_{\text{LWE}}$, yielding $\llbracket S \rrbracket_{\text{LWE}} = \llbracket 0 \rrbracket_{\text{LWE}}$ (Segment 0).

Step 2: Local Interval Index Computation. As shown in Figure 6(b), we then homomorphically compute $\llbracket I \rrbracket_{\text{LWE}}$ where $I = \lfloor \frac{x - \tau_S}{\tau_{S+1} - \tau_S} \cdot N_{\text{br}} \rfloor$ via the online-construct-then-select strategy.

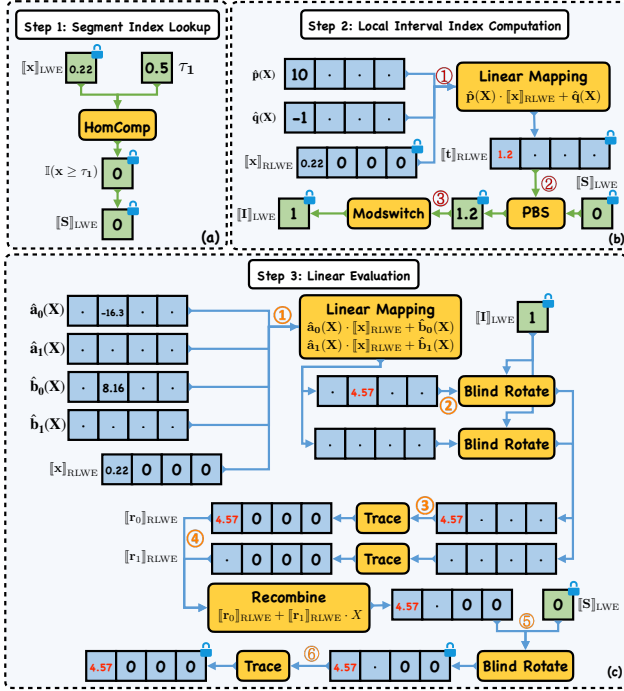


Figure 6: ROSETTA cryptographic lookup on the running example ($N_{\text{seg}}=2$, $N_{\text{br}}=4$, $(\tau_0, \tau_1, \tau_2)=(0.1, 0.5, 2.0)$, $x=0.22$). **Red** marks the value selected at each step; “.” marks entries omitted for clarity; the target LUT is Figure 5(d).

Construct. ① We precompute two plaintext polynomials, a slope $\hat{p}(X)$ and an offset $\hat{q}(X)$:

$$\begin{aligned}\hat{p}(X) &= \sum_{s=0}^{N_{\text{seg}}-1} \frac{N_{\text{br}}}{\tau_{s+1}-\tau_s} \cdot X^s & (= 10.0 + 2.667X), \\ \hat{q}(X) &= \sum_{s=0}^{N_{\text{seg}}-1} \left(-\frac{\tau_s \cdot N_{\text{br}}}{\tau_{s+1}-\tau_s} \right) \cdot X^s & (= -1.0 - 1.333X).\end{aligned}$$

A single plaintext-ciphertext multiplication (*linear mapping*) then constructs the *test polynomial ciphertext*

$$[\mathbf{t}]_{\text{RLWE}} = \hat{p}(X) \cdot [x]_{\text{RLWE}} + \hat{q}(X).$$

Since x is encoded in the constant coefficient of $[x]_{\text{RLWE}}$, multiplying by $\hat{p}(X)$ effectively broadcasts x to every coefficient position. This produces a ciphertext whose s -th coefficient holds the candidate local interval index for segment s :

$$[\mathbf{t}]_{\text{RLWE}}[s] = \frac{(x - \tau_s) \cdot N_{\text{br}}}{\tau_{s+1} - \tau_s} \quad (\text{coefficients: } [1.200, -0.747]).$$

Select and Round. ② A single PBS using $[S]_{\text{LWE}}$ then selects the S -th coefficient; in our example, coefficient 0 is selected, yielding the value 1.200. ③ The ModSwitch operation then rounds this value, giving $\lfloor 1.200 \rfloor = 1$ and producing $[I]_{\text{LWE}} = [1]_{\text{LWE}}$ (Interval 1). In total, this step requires one plaintext-ciphertext multiplication, one plaintext addition, and one PBS.

Step 3: Linear Evaluation. As shown in Figure 6(c), the final step homomorphically computes $f(x) \approx a_{SI} \cdot x + b_{SI}$, where S and I are available only as LWE ciphertexts $[S]_{\text{LWE}}$ and $[I]_{\text{LWE}}$. Since the coefficients (a_{SI}, b_{SI}) depend on both encrypted indices, we apply

the online-construct-then-select strategy twice, performing two rounds of selection, first by I , then by S :

Phase 1 Construct. ① For each segment s , we pack the slope and offset of every local interval $i \in \{0, \dots, N_{\text{br}} - 1\}$ from the per-segment LUTs (Figure 5(f)) into a slope polynomial $\hat{a}_s(X) = \sum_i a_{si} \cdot X^i$ and an offset polynomial $\hat{b}_s(X) = \sum_i b_{si} \cdot X^i$. In the running example (Segment 0), $\hat{a}_0(X) = -47.06 - 16.33X - 8.25X^2 - 4.97X^3$ and $\hat{b}_0(X) = 14.12 + 8.16X + 5.77X^2 + 4.47X^3$. Using the same broadcast property as Step 2, a *linear mapping* computes

$$[\mathbf{r}_s]_{\text{RLWE}} = \hat{a}_s(X) \cdot [x]_{\text{RLWE}} + \hat{b}_s(X),$$

so that the i -th coefficient of $[\mathbf{r}_s]_{\text{RLWE}}$ stores $a_{si} \cdot x + b_{si}$. In the running example, $[\mathbf{r}_0]$ has coefficients (3.765, 4.571, 3.959, 3.379) and $[\mathbf{r}_1]$ has (2.538, 1.714, 1.296, 1.042), where the target $a_{01} \cdot 0.22 + b_{01} = 4.571$ sits at coefficient 1 of $[\mathbf{r}_0]$.

Phase 1 Select by I . Within each ciphertext, only the I -th coefficient holds the correct result. ② A blind rotation using $[I]_{\text{LWE}}$ moves the I -th coefficient to the constant term. ③ A subsequent Trace zeros out all other coefficients:

$$[\mathbf{r}_s] \leftarrow \text{Trace}(\text{BlindRot}([\mathbf{r}_s]_{\text{RLWE}}, [I]_{\text{LWE}})).$$

Each ciphertext now contains a single value $a_{SI} \cdot x + b_{SI}$ on the constant term, with the rest filled with zeros; in the running example, $[\mathbf{r}_0] = [4.571]$ and $[\mathbf{r}_1] = [1.714]$.

Phase 2 Construct (Recombine). ④ To recombine the N_{seg} single-value ciphertexts into a single test polynomial ciphertext, we shift the s -th result into the s -th coefficient position via monomial multiplication and sum:

$$[\mathbf{R}]_{\text{RLWE}} = \sum_{s=0}^{N_{\text{seg}}-1} [\mathbf{r}_s] \cdot X^s.$$

In the running example, the combined ciphertext stores 4.571 and 1.714, with the target at position $S = 0$.

Phase 2 Select by S . ⑤ A final blind rotation using $[S]_{\text{LWE}}$ moves the S -th coefficient to the constant term. ⑥ A final Trace isolates it: $[f(x)]_{\text{RLWE}} = \text{Trace}(\text{BlindRot}([\mathbf{R}]_{\text{RLWE}}, [S]_{\text{LWE}})) \approx [4.571]_{\text{RLWE}}$.

In total, Step 3 requires N_{seg} plaintext-ciphertext multiplications, N_{seg} plaintext-ciphertext additions, $N_{\text{seg}} + 1$ blind rotations, and $N_{\text{seg}} + 1$ Trace operations. Combined with Steps 1 and 2, the entire protocol uses $N_{\text{seg}} - 1$ HomComp invocations, $N_{\text{seg}} + 1$ plaintext-ciphertext multiplications, $N_{\text{seg}} + 2$ blind rotations, and $N_{\text{seg}} + 1$ Trace operations.

3.3 Pipeline Integration

With the lookup protocol in place, we now describe how to prepare its inputs and convert its output back to the SIMD domain (Figure 7). We first describe the single-value pipeline (§3.3.1), and then extend it to the multi-value case (§3.3.2).

3.3.1 Input Preparation and Output Conversion. The lookup protocol requires the input x in two ciphertext forms: an RLWE ciphertext $[x]_{\text{RLWE}}$ in coefficient form with x encoded in the constant term, for the plaintext-ciphertext multiplications in Steps 2 and 3, and an LWE ciphertext $[x]_{\text{LWE}}$ for the homomorphic comparisons in Step 1. Since the preceding layers of the LLM pipeline produce SIMD-encoded ciphertexts, a scheme conversion is needed.

Starting from a SIMD-encoded ciphertext, we first apply SlotToCoeff (STC) [11] to convert the slot representation ciphertext into

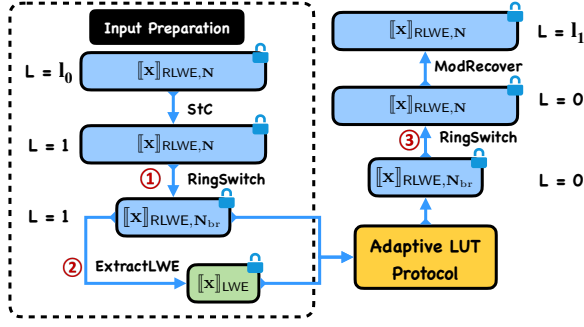


Figure 7: Input/output pipeline: StC maps the CKKS ciphertext to coefficient form, then derive the RLWE and LWE ciphertexts with RingSwitch and ExtractLWE; after the lookup we return to CKKS via RingSwitch + ModRecover.

coefficient form. The resulting coefficient-form ciphertext is then processed in two steps to produce the required inputs. ① For the RLWE input, we apply RingSwitch to reduce the ring dimension from N to N_{br} , so that the subsequent polynomial multiplications in the lookup protocol operate on a much smaller ring at reduced cost. ② We then apply ModSwitch to drop to level 0, then ExtractLWE to extract the constant term coefficient to get the LWE input. Both inputs are then sent into the lookup protocol.

After the lookup protocol completes, the output $\llbracket f(x) \rrbracket_{RLWE}$ resides on the small ring of dimension N_{br} with the result in the constant term. ③ To return to the SIMD domain for subsequent layers, we first apply RingSwitch to embed the result back into the full ring of dimension N , then apply ModRecover, which internally performs ModRaise, CoeffToSlot (CtS), and EvalMod, to restore a valid CKKS ciphertext at the required level. Algorithm 1 summarizes the complete protocol described above.

3.3.2 Multi-Value Batching. The protocol above targets the single-value case where one ciphertext carries a single element x . In certain scenarios, such as the reciprocal in Softmax normalization, K elements (e.g., $K = 16$ attention heads) each require an independent non-linear evaluation. In this case, we perform StC only once to convert the SIMD ciphertext into coefficient form. After StC, the K input values reside in K coefficients of the coefficient-form ciphertext. We then apply RingSwitch to produce one or more small-ring ciphertexts of dimension N_{br} ; and each element lands at a known coefficient position in one of these ciphertexts.

To evaluate the j -th element ($0 \leq j < K$), we take a copy of its corresponding small-ring ciphertext, multiply by the appropriate monomial X^{-c} (where c is the coefficient position of that element) to rotate it into the constant term, and apply a Trace to zero out the remaining coefficients. The resulting ciphertext serves as the RLWE input $\llbracket x_j \rrbracket_{RLWE}$ for the lookup protocol; the LWE input $\llbracket x_j \rrbracket_{LWE}$ is obtained by applying ModSwitch and ExtractLWE to this same ciphertext. Both inputs are then fed into the lookup protocol. Once all K evaluations complete, the output ciphertexts are recombined by shifting each result back to its original coefficient position and summing them. A single RingSwitch then maps the combined small-ring ciphertexts back into one full-ring ciphertext, and a single ModRecover restores it to the SIMD domain at the required level.

Algorithm 1 ROSETTA Cryptographic Lookup Protocol

Input: CKKS ciphertext $\llbracket x \rrbracket_{CKKS}$; precomputed polynomials $\hat{p}, \hat{q}, \{\hat{a}_s, \hat{b}_s\}_{s=0}^{N_{seg}-1}$; boundaries $\{\tau_j\}_{j=0}^{N_{seg}}$

Output: CKKS ciphertext $\llbracket f(x) \rrbracket_{CKKS}$

```

1: // Input Preparation
2:  $\llbracket x \rrbracket_{coeff} \leftarrow \text{SlotToCoeff}(\llbracket x \rrbracket_{CKKS})$ 
3:  $\llbracket x \rrbracket_{RLWE} \leftarrow \text{RingSwitch}(\llbracket x \rrbracket_{coeff}, N_{br})$ 
4:  $\llbracket x \rrbracket_{LWE} \leftarrow \text{ExtractLWE}(\text{ModSwitch}(\llbracket x \rrbracket_{RLWE}))$ 
5: // Step 1: Segment Index
6: for  $j = 1$  to  $N_{seg} - 1$  do
7:    $\llbracket x \geq \tau_j \rrbracket_{LWE} \leftarrow \text{HomComp}(\llbracket x \rrbracket_{LWE}, \tau_j)$ 
8: end for
9:  $\llbracket S \rrbracket_{LWE} \leftarrow \sum_{j=1}^{N_{seg}-1} \llbracket x \geq \tau_j \rrbracket_{LWE}$ 
10: // Step 2: Local Interval Index
11:  $\llbracket t \rrbracket_{RLWE} \leftarrow \llbracket x \rrbracket_{RLWE} \cdot \hat{p}(X) + \hat{q}(X)$  {Construct}
12:  $\llbracket I \rrbracket_{LWE} \leftarrow \text{PBS}(\llbracket t \rrbracket_{RLWE}, \llbracket S \rrbracket_{LWE})$  {select & round}
13: // Step 3: Linear Evaluation
14: for  $s = 0$  to  $N_{seg} - 1$  (construct) do
15:    $\llbracket r_s \rrbracket_{RLWE} \leftarrow \llbracket x \rrbracket_{RLWE} \cdot \hat{a}_s(X) + \hat{b}_s(X)$ 
16: end for
17: for  $s = 0$  to  $N_{seg} - 1$  (select by  $I$ ) do
18:    $\llbracket r_s \rrbracket \leftarrow \text{Trace}(\text{BlindRot}(\llbracket r_s \rrbracket, \llbracket I \rrbracket_{LWE}))$ 
19: end for
20:  $\llbracket R \rrbracket_{RLWE} \leftarrow \sum_{s=0}^{N_{seg}-1} \llbracket r_s \rrbracket \cdot X^s$  {Recombine}
21:  $\llbracket f(x) \rrbracket_{RLWE} \leftarrow \text{Trace}(\text{BlindRot}(\llbracket R \rrbracket, \llbracket S \rrbracket_{LWE}))$  {Select by  $S$ }
22: // Output Conversion
23:  $\llbracket f(x) \rrbracket_{CKKS} \leftarrow \text{ModRecover}(\text{RingSwitch}(\llbracket f(x) \rrbracket_{RLWE}))$ 
24: return  $\llbracket f(x) \rrbracket_{CKKS}$ 

```

4 Scheme-Aware Operator Selection

As diverse nonlinear layers exist in Transformer models, determining which to evaluate with our TFHE-based segmented LUT protocol and which to keep in CKKS is non-trivial. To address this, we propose a scheme-aware operator selection method. We first categorize the operators inside the nonlinear layers and identify the subset that benefits from TFHE evaluation. We then propose an algorithm that jointly optimizes TFHE selection and level schedule.

4.1 Operator Categorization

As shown in Table 2, we classify the operators within the nonlinear layers of a Transformer model into two categories: *arithmetic primitives* and *non-arithmetic primitives*.

Arithmetic primitives include element-wise addition and multiplication, scalar addition and multiplication, and summation. For example, the LayerNorm affine $y = \gamma \cdot x + \beta$ is realized as a scalar multiplication followed by a scalar addition. These operations can be evaluated natively and efficiently by CKKS via its SIMD arithmetic: they are depth-cheap, fully parallelized across SIMD slots, and their cost is independent of the input value range. On the contrary, TFHE is not suitable for arithmetic primitives like multiplications. Therefore, all arithmetic primitives are assigned to CKKS by default, and the selection decision focuses on non-arithmetic primitives.

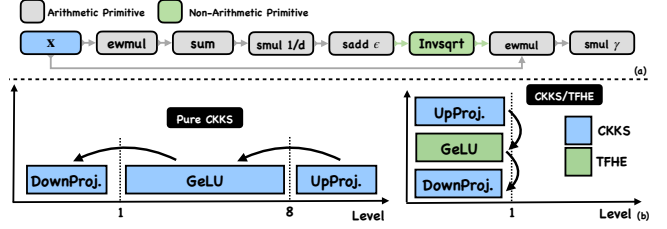
Table 2: Definition of operators. Suffix cc denotes ciphertext-ciphertext operands; cp denotes ciphertext-plaintext.

Type	Name	Description
Arithmetic Primitives		
Identity	ewadd_cc/cp	Element-wise addition of cc or cp
	ewmul_cc/cp	Element-wise multiplication of cc or cp
Expansion	sadd_cp	Scalar-broadcast addition
	smul_cp	Scalar-broadcast multiplication
Reduction	sum	Summation along a specified dimension
Non-Arithmetic Primitives		
exp		Exponential e^x , used in Softmax
silu, gelu, relu		Non-linear activations in FFN layers
inv, invsqrt		Reciprocal $1/x$ and recip. $\sqrt{1/x}$
Composite Layers		
Softmax		Attention score normalization with exp and inv
LayerNorm		Mean-variance normalization with invsqrt
RMSNorm		Root-mean-square normalization with invsqrt

Non-arithmetic primitives include exp, SiLU/GELU/ReLU, and $1/x/1/\sqrt{x}$. Under CKKS, these primitives are evaluated by a high-degree polynomial approximation or computed via an iterative method such as Goldschmidt iteration for $1/x$. Using CKKS for these primitives can be costly because such evaluations are *level-hungry*: polynomial or iterative methods consume many multiplicative levels, so a single primitive may trigger frequent bootstraps. This makes non-arithmetic primitives the dominant source of cost in nonlinear layers, and makes them potential candidates for TFHE evaluation, since TFHE evaluates arbitrary functions directly through lightweight look-up tables and thus avoids the multiplicative-depth pressure of CKKS.

Each nonlinear layer in a Transformer is a composition of arithmetic and non-arithmetic primitives. Some layers contain only a single non-arithmetic primitive (e.g., GELU and ReLU activation layers), while others are *composite*, interleaving arithmetic primitives with non-arithmetic primitives. As illustrated in Figure 8(a), the composite layer RMSNorm decomposes into a pipeline of seven primitives: given an input vector $x \in \mathbb{R}^d$, element-wise multiplication produces x_i^2 , summation followed by scalar multiplication $\frac{1}{d}$ yields the mean of squares $\mu = \frac{1}{d} \sum_i x_i^2$, scalar addition gives $\mu + \epsilon$, reciprocal square root produces the normalization factor $(\mu + \epsilon)^{-1/2}$, and a final element-wise multiplication with scalar multiplication γ yields the output $\gamma \cdot x \cdot (\mu + \epsilon)^{-1/2}$. In this pipeline, only the reciprocal square root $1/\sqrt{\cdot}$ is non-arithmetic; all remaining six primitives are arithmetic and evaluated natively by CKKS.

This decomposition reveals that selecting an entire nonlinear layer for TFHE is unnecessarily costly, since the majority of its constituent primitives are arithmetic and already efficient under CKKS. Instead, we decompose each nonlinear layer into many primitives and restrict TFHE selection to non-arithmetic primitives alone, while all arithmetic primitives remain in CKKS. This fine-grained strategy reduces the scheme-assignment search space from all operators to only the few non-arithmetic candidates, and avoids wasting TFHE resources on operations that CKKS already handles well.

**Figure 8: (a) Primitive decomposition of RMSNorm, where only invsqrt (green) is a TFHE candidate. (b) Level management of a GELU FFN: pure CKKS (left) vs. TFHE selection (right).**

4.2 Scheme-Aware Placement

The preceding analysis identifies non-arithmetic primitives as potential candidates for TFHE evaluation, but determining which ones to select is non-trivial because assigning a primitive to TFHE not only changes its own cost, but also affects the level assignment of the surrounding layers. First, selecting a non-arithmetic primitive for TFHE evaluation eliminates the multiplicative levels that its CKKS counterpart would have consumed, altering the overall level budget. Second, the input levels of the preceding and succeeding linear layers become additional decision variables that jointly determine both layer computation cost and scheme conversion cost; in particular, the higher the LWE-to-CKKS target level, the more expensive the conversion. Consequently, comparing the per-primitive cost of TFHE against CKKS in isolation is insufficient: the total cost depends on how the surrounding layers' levels are assigned, and the optimal level assignment in turn depends on how schemes are selected.

We take GELU inside a Transformer FFN as an example (Figure 8(b)). The FFN consists of an up-projection, a GELU activation, and a down-projection. Under pure CKKS, the up-projection is assigned a high input level to leave enough levels for the subsequent GELU evaluation. If GELU is instead selected for TFHE, the up-projection only needs enough levels for its own computation, since GELU no longer consumes any CKKS levels; meanwhile, the TFHE output only needs to be recovered to a level sufficient for the down-projection, which is exactly the LWE to CKKS target level that determines the conversion cost. Whether selecting GELU for TFHE thus requires jointly considering TFHE evaluation cost, conversion cost, and cost of other layers.

To address this challenge, we build on CacheMir [55], a state-of-the-art level management algorithm designed for pure-CKKS private inference, and extend it to be scheme-aware. CacheMir formulates the problem as a shortest-path problem over a Directed Acyclic Graph (DAG) derived from the network structure. As shown in Figure 9(a), CacheMir constructs a DAG by mapping each feasible input level of every layer to a vertex, and connecting the vertices of adjacent layers with edges whose weights capture the latency of computing the layer at the given input level plus any bootstrap needed to reach the next layer's input level. Since each layer is computed before any optional bootstrap or level drop [19], input levels smaller than the layer's multiplicative depth are pruned as infeasible. Under this construction, a shortest path through the

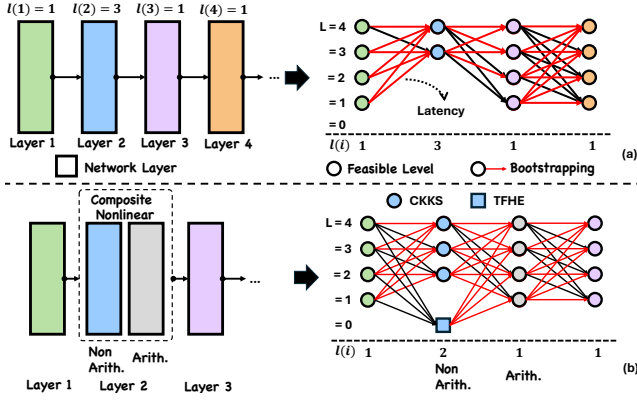


Figure 9: (a) Pure-CKKS level graph with the shortest path giving the optimal level schedule. (b) Scheme-aware extension: a TFHE node (blue, level 0) is injected at each non-arithmetic primitive, and the shortest path automatically chooses between the CKKS and TFHE paths.

DAG corresponds to a globally optimal per-layer level assignment, which in turn gives the bootstrap placement. Formally, for a network with D layers and a maximum level L , let $v[i, j]$ denote the vertex corresponding to the i -th layer with input level j , and let $e[i, x, y]$ denote the directed edge from $v[i, x]$ to $v[i + 1, y]$ with weight $w[i, x, y]$. Let $\ell(i) \leq L$ denote the multiplicative depth of layer i ; an edge $e[i, x, y]$ is feasible only when $x \geq \ell(i)$. The minimum inference latency is then

$$\min_{\{x_i\}} \sum_{i=1}^D w[i, x_i, x_{i+1}],$$

where $\{x_i\}$ is the sequence of input levels chosen along the shortest path. Given that the latency of level drop is negligible and that bootstrap performance is insensitive to the input level [19], the edge weight decomposes as

$$w[i, x, y] \approx t_i(x) + \mathbf{1}_{x-\ell(i)<y} \cdot t_{\text{boot}}(y),$$

where $t_i(x)$ is the layer- i computation latency at input level x (monotonically increasing in x), $t_{\text{boot}}(y)$ is the bootstrap latency to target level y (also monotonically increasing), and the indicator equals 1 when the residual level $x - \ell(i)$ after computation is insufficient for the desired output level y , triggering a bootstrap.

Building on this DAG formulation, our framework makes CacheMir scheme-aware in two steps.

Step 1: Primitive-level decomposition. Following §4.1, we expand each nonlinear layer in the DAG into its individual primitives. Each arithmetic primitive still becomes a standard CKKS vertex column in the level graph. Each non-arithmetic primitive is marked as a candidate for TFHE selection.

Step 2: TFHE node injection. For each tagged primitive p , we insert a special *TFHE* node into the level graph alongside the existing CKKS vertices, as illustrated in Figure 9(b). Since TFHE evaluation operates at level 0, this node is placed at level 0 in the graph; any vertex of the preceding layer can reach it via StC and ExtractLWE. Its outgoing edges connect to every feasible input level of the succeeding CKKS segment. A TFHE edge from $v[i, 0]$ to $v[i + 1, y]$

Table 3: Models used in evaluation.

Model	#Heads	Hidden dim.	#Layers
GPT-2 Base [42]	12	768	12
TinyLlama-1.1B [61]	16	2048	22
LLaMA-3-8B [22]	32	4096	32

carries a weight of the form

$$\begin{aligned} w_T[i, 0, y] &= t_{\text{StC}} + t_{\text{TFHE}}(n_p, [\ell_p, r_p]) + t_{\text{ModRecover}}(y) \\ &= t_{\text{TFHE}}(n_p, [\ell_p, r_p]) + t_{\text{boot}}(y), \end{aligned}$$

where the three terms correspond respectively to CKKS-to-LWE conversion, segmented-LUT evaluation, and LWE-to-CKKS conversion. The first term t_{StC} covers SlotToCoeff and ExtractLWE, and the second term $t_{\text{TFHE}}(n_p, [\ell_p, r_p])$ is the TFHE evaluation cost governed by the segmented LUT configuration of §3. Here, n_p denotes the number of input elements processed by primitive p , and $[\ell_p, r_p]$ denotes its input range. The third term $t_{\text{ModRecover}}(y)$ covers ModRaise, CoeffToSlot, and EvalMod to recover the output to level y . Since SlotToCoeff followed by ModRecover constitutes a full CKKS bootstrap (SlotToCoeff + ModRaise + CoeffToSlot + EvalMod), we have $t_{\text{StC}} + t_{\text{ModRecover}}(y) = t_{\text{boot}}(y)$, reusing a similar cost function from the pure-CKKS edge weight. The shortest path on this augmented DAG then gives the minimum-latency schedule, simultaneously determining which non-arithmetic primitives are evaluated by TFHE (those whose path traverses a TFHE node) and the optimal input level for every layer, without requiring a separate search over scheme selection decisions.

5 Evaluation

5.1 Experimental Setup

Implementation. ROSETTA is implemented with Lattigo [1] as the CPU backend and a customized Phantom library [52] for GPU acceleration. Since Lattigo does not expose a direct LWE API, we manage LWE ciphertexts with RLWE ciphertexts and construct our protocols on top of its blind-rotation primitive. All baselines are re-implemented in the same framework with identical cryptographic parameters for fair comparison. We measure the latency of all methods at the granularity of modules to obtain a precise estimation of end-to-end runtime. All experiments are conducted on an Intel Xeon Gold 6240R CPU (2.40 GHz, 48 threads) with 384 GB of main memory, running Go 1.24, CUDA 12.4, and Lattigo v6. An NVIDIA A100 GPU (80 GB) is also used for the end-to-end experiment in Table 7.

Cryptographic Configuration. For the CKKS scheme, we set the polynomial degree to $N=2^{16}$ with a 1,763-bit ciphertext modulus Q . Following CacheMir [55], we set the maximum level to $L=13$ and the multiplicative depth of the bootstrapping circuit to $K=15$, adopting the Coefficients-to-Slots (CtS)-first bootstrapping variant with CtS depth 4 and Slots-to-Coefficients (StC) depth 3. To support an 8-depth approximate modular reduction via a sparse secret key (Hamming weight 192), we apply sparse secret encapsulation [6]. The RNS modulus chain is configured with $\log_2 q_0 \approx 53$ and $\log_2 q_i \approx 41$ for all $i \geq 1$ to ensure sufficient noise budget. For the TFHE scheme, we set the LWE dimension to $n_{\text{LWE}}=1,024$ and

Table 4: Softmax precision (bits, top) and latency (s, bottom) across context length n and input range. “–” denotes unavailable or divergent results.

Cat.	Method	Range $[-8, 8]$					Range $[-16, 16]$					Range $[-32, 32]$				
		512	1k	2k	4k	8k	512	1k	2k	4k	8k	512	1k	2k	4k	8k
Pure CKKS	CacheMir	16.4 149 s	15.4 149 s	20.4 149 s	21.5 149 s	20.4 150 s	13.8 149 s	10.7 149 s	10.9 149 s	10.1 149 s	12.9 149 s	9.7 234 s	8.4 234 s	8.0 234 s	9.5 235 s	10.4 236 s
	NEXUS	14.7 110 s	15.6 110 s	16.6 110 s	17.6 110 s	18.6 110 s	–	–	–	–	–	–	–	–	–	–
	MOAI	14.7 110 s	15.6 110 s	16.6 111 s	17.6 113 s	18.6 117 s	–	–	–	–	–	–	–	–	–	–
Hybrid CKKS/TFHE	PEGASUS	9.0 69 s	9.0 77 s	9.3 94 s	10.5 128 s	9.2 197 s	6.7 69 s	6.8 77 s	7.9 94 s	8.0 128 s	8.6 197 s	5.9 89 s	6.3 97 s	7.0 114 s	7.6 148 s	8.2 217 s
	Ours	14.1 32 s	15.1 31 s	16.0 31 s	17.0 31 s	18.0 32 s	12.7 51 s	13.6 51 s	14.5 51 s	15.6 51 s	16.6 51 s	11.2 70 s	12.1 70 s	13.0 70 s	9.2 71 s	15.1 70 s

Table 5: Wide-range Softmax comparison with Cho et al. [14] on $[-128, 128]$. Each cell reports RMSE precision (bits, top) and latency (s, bottom).

Method	512	1k	2k	4k	8k
Cho	17 192 s	16 196 s	9 197 s	9 195 s	4 198 s
Ours	12 117 s	15 118 s	17 120 s	18 121 s	18 122 s

the blind-rotation ring degree to $N_{br}=2,048$, using a 53-bit ciphertext modulus. All parameter sets achieve 128-bit security according to the Homomorphic Encryption Standard [3]. For ROSETTA’s segmented LUT, we use $N_{seg}=4$ segments per nonlinear primitive with linear interpolation for each segment. The boundaries are logarithmically spaced for $1/x$ and $1/\sqrt{x}$, and uniformly spaced for SiLU and GELU. We report precision in bits, defined as $-\log_2$ of the root-mean-square error between the FHE-evaluated output and the plaintext reference over inputs sampled from the target input range.

Models, Datasets, and Baselines. We evaluate on three representative generative language models: GPT-2 Base [42], TinyLlama-1.1B [61], and LLaMA-3-8B [22], with key architectural details summarized in Table 3. Sequence lengths vary across experiments. We do not fine-tune any model and evaluate on several datasets. Perplexity is reported on several 8B models, which are most sensitive to errors introduced by FHE.

Our pure-CKKS baseline is CacheMir [55], which adopts the nonlinear operators from THOR [38] without modification. Since THOR targets prefill, CacheMir is the directly comparable system for our decode-stage setting. We additionally integrate the nonlinear methods of MOAI [60] and NEXUS [59] as drop-in replacements for CacheMir’s nonlinear operators. For hybrid CKKS/TFHE frameworks [27, 63] that share the same all-TFHE strategy, we use PEGASUS as a representative baseline.

5.2 Micro-benchmarks

We first evaluate ROSETTA on three main nonlinear layers in LLMs, the Softmax, RMSNorm, and LayerNorm. After being processed by ROSETTA, each layer is decomposed into a combination of CKKS and TFHE operations based on its computational characteristics. We then evaluate several non-arithmetic primitives in isolation to further demonstrate the advantage of the segmented LUT design.

Softmax. Table 4 shows the precision (bits) and latency (s) comparisons between ROSETTA and the baselines on Softmax. Following prior work [38, 55], inputs are generated from a normal distribution $\mathcal{N}(0, \sigma)$ truncated to range $[-M, M]$ with variance $\sigma^2 = M^2/3$ for $M \in \{8, 16, 32\}$, which approximates the distribution of attention scores in LLMs. For latency, ROSETTA is 2.2–4.8× faster than the pure-CKKS baselines (CacheMir, NEXUS, MOAI), which all rely on iterative methods with many bootstraps, and 1.3–6.1× faster than PEGASUS, whose latency grows with context length n because it runs a per-element PBS for every exponential function. This shows that the hybrid design of ROSETTA can handle the heterogeneous workload of nonlinear operators and achieve better latency. For precision, ROSETTA is comparable to the pure-CKKS baselines on narrow ranges and gains a 2–5-bit advantage on wide ranges where they degrade or fail to run, while consistently outperforming PEGASUS by 4–9 bits, showing that the segmented LUT keeps precision robust as the input range widens.

To further evaluate Softmax over a wider input range, Table 5 compares ROSETTA with our reimplement of Cho in Lat-tigo under identical cryptographic parameters on $[-128, 128]$. Cho achieves 17 and 16 bits of RMSE precision at $n = 512$ and 1k, but drops to 9, 9, and 4 bits as n increases to 2k, 4k, and 8k. In contrast, ROSETTA achieves 12–18 bits across these context lengths and reaches 17–18 bits at $n \geq 2k$, yielding an 8–14-bit advantage in the long-context settings. It also reduces latency from Cho’s 192–198 s to 117–122 s, corresponding to a 1.6× speedup.

RMSNorm and LayerNorm. Table 6 compares ROSETTA with the baselines on RMSNorm and LayerNorm. Inputs are drawn from zero-mean normal distribution with per-group variance σ^2 sampled from two ranges $[0.2, 150]$, and $[1.0, 8192]$ following [38], across hidden dimensions $d \in \{768, 2048, 4096\}$. For latency, ROSETTA is

Table 6: RMSNorm and LayerNorm precision (bits, top) and latency (s, bottom) across hidden dimension d and variance $\sigma_{\max}^2$.

Cat.	Method	RMSNorm						LayerNorm					
		$\sigma_{\max}^2=150$			$\sigma_{\max}^2=8192$			$\sigma_{\max}^2=150$			$\sigma_{\max}^2=8192$		
		768	2k	4k	768	2k	4k	768	2k	4k	768	2k	4k
Pure CKKS	CacheMir	10.4 65 s	10.6 65 s	11.0 65 s	11.2 66 s	10.4 66 s	9.1 65 s	8.8 68 s	9.6 68 s	10.7 68 s	6.7 68 s	8.2 68 s	10.7 69 s
	NEXUS	10.1 148 s	10.3 148 s	10.7 148 s	11.1 189 s	10.2 189 s	8.7 189 s	8.6 151 s	9.4 151 s	10.6 151 s	6.4 191 s	7.9 192 s	10.2 192 s
	MOAI	10.1 148 s	10.3 149 s	10.7 151 s	11.1 189 s	10.2 190 s	8.7 192 s	8.6 151 s	9.4 154 s	10.6 157 s	6.4 192 s	7.9 194 s	10.2 198 s
Hybrid CKKS/TFHE	PEGASUS	6.9 22 s	6.4 22 s	6.6 22 s	4.6 22 s	4.7 22 s	4.7 22 s	8.7 24 s	8.9 25 s	8.6 25 s	6.9 24 s	6.9 25 s	6.9 24 s
	Ours	14.1 26 s	11.9 26 s	14.1 26 s	11.2 26 s	14.2 26 s	15.8 27 s	14.6 28 s	14.6 29 s	14.6 28 s	12.5 29 s	10.1 29 s	14.9 28 s

Table 7: End-to-end latency on CPU (hours, top) and GPU (minutes, bottom) across models and context lengths n .

Cat.	Method	GPT-2 Base			TinyLlama-1.1B			LLaMA-3-8B		
		512	2k	8k	512	2k	8k	512	2k	8k
Pure CKKS	CacheMir	1.58 h 2.57 m	1.58 h 2.57 m	1.60 h 2.65 m	2.88 h 4.97 m	2.88 h 4.98 m	2.95 h 5.45 m	4.25 h 8.16 m	4.29 h 8.39 m	4.44 h 9.77 m
	PEGASUS	1.08 h 1.94 m	1.97 h 3.83 m	5.70 h 11.5 m	2.14 h 4.20 m	4.33 h 8.83 m	13.5 h 27.8 m	4.24 h 9.28 m	10.8 h 23.0 m	37.5 h 78.3 m
Hybrid CKKS/TFHE	Ours	0.84 h 1.31 m	0.84 h 1.31 m	0.85 h 1.39 m	1.52 h 2.71 m	1.52 h 2.71 m	1.57 h 3.18 m	2.28 h 5.18 m	2.31 h 5.41 m	2.45 h 6.79 m

Table 8: Relative PPL increase over the plaintext baseline (PPL_0) across models and datasets; lower is better.

Model	Method	WikiText-2	WikiText-103	LAMBADA	GSM8K	ShareGPT	Avg.
LLaMA-3-8B	Ours	+2.0%	+2.2%	+1.3%	+0.7%	+1.6%	+1.6%
	CacheMir	+2.0%	+1.8%	+1.3%	+0.7%	+1.1%	+1.4%
	NEXUS/MOAI	–	–	–	–	–	–
	PEGASUS	+41.4%	+40.6%	+23.0%	+19.4%	+25.8%	+30.0%
Qwen2-7B	Ours	+0.1%	−0.0%	+0.1%	−0.1%	+0.7%	+0.1%
	PEGASUS	+282.8%	+278.7%	+364.8%	+196.8%	+208.1%	+266.2%
Mistral-7B	Ours	+1.1%	+0.8%	+0.4%	+0.4%	+0.4%	+0.6%
	PEGASUS	+12.2%	+12.5%	+8.7%	+8.3%	+8.4%	+10.0%
DeepSeek-8B	Ours	+3.3%	+3.3%	+4.3%	+2.4%	+3.0%	+3.3%
	CacheMir	+0.4%	+0.5%	+0.8%	−0.1%	−0.2%	+0.3%
	NEXUS/MOAI	–	–	–	–	–	–
	PEGASUS	+166.2%	+161.0%	+139.1%	+155.0%	+167.5%	+157.8%
All-model avg.	Ours	+1.6%	+1.6%	+1.5%	+0.9%	+1.4%	+1.4%
	PEGASUS	+125.7%	+123.2%	+133.9%	+94.9%	+102.5%	+116.0%

– indicates that PPL diverges.

2.4–7.4× faster than the pure-CKKS baselines (CacheMir, NEXUS, MOAI), which rely on iterative methods with frequent bootstraps, and achieves comparable latency to PEGASUS since both evaluate inverse square root on a single scalar with lightweight PBS. For precision, ROSETTA outperforms PEGASUS by 3–11 bits across configurations, as the segmented LUT provides better precision

over the full domain due to the linear approximation and adaptive method.

Primitive Evaluation. We also benchmark non-arithmetic primitives $1/x$, $1/\sqrt{x}$, SiLU, and GELU to further demonstrate the performance of the segmented LUT design.

Table 9: Single-primitive benchmark for $1/x$ and $1/\sqrt{x}$.

Function	Range	Goldschmidt		PEGASUS		Ours	
		Prec.	Lat. (s)	Prec.	Lat. (s)	Prec.	Lat. (s)
$1/x$	[0.01, 10]	9.8	45.5	5.4	0.8	12.7	4.8
$1/x$	[0.01, 100]	12.7	41.0	5.4	0.7	12.5	4.8
$1/\sqrt{x}$	[0.01, 10]	13.0	39.0	6.4	0.8	16.7	4.7
$1/\sqrt{x}$	[0.01, 100]	15.7	41.9	6.4	0.8	12.7	4.8

Table 10: Activation function evaluation on $[-20, 20]$.

Function	Method	Prec.	Lat. (s)
SiLU	Chebyshev	12.4	13.7
SiLU	PEGASUS	5.8	0.8
SiLU	Ours	16.8	4.8
GELU	Chebyshev	15.2	12.5
GELU	PEGASUS	6.5	0.8
GELU	Ours	15.5	4.7

Table 9 reports results for $1/x$ and $1/\sqrt{x}$ over two different input ranges. For latency, ROSETTA is 8–9× faster than Goldschmidt, which requires many iterations that consume multiple multiplicative levels and trigger bootstraps frequently. For precision, ROSETTA achieves 12.5–16.9 bits and outperforms PEGASUS by 6–10 bits, whose single-LUT approach is limited to 5–7 bits.

Results for SiLU and GELU on $[-20, 20]$ appear in Table 10. ROSETTA is 2.7–2.9× faster than a degree-59 Chebyshev approximation with comparable precision (15.5–16.8 bits vs. 12.4–15.2 bits). PEGASUS achieves only 5.8–6.5 bits, which ROSETTA outperforms by 9–10 bits. Both benchmarks demonstrate the lightweight but accurate nature of the segmented LUT design, which can achieve high precision with a small number of segments and entries, while avoiding the heavy cost of iterative/polynomial methods.

We calibrate the evaluated input ranges by profiling the nonlinear inputs of LLaMA-3-8B with a fixed public prefix prepended [46] across the five datasets used in our PPL evaluation. For Softmax, 31 of 32 layers have logit spans below 64 across all datasets, while the last layer reaches 105.5–109.2. Therefore, $[-32, 32]$ captures the common case, and $[-128, 128]$ covers the observed wide-range case. For RMSNorm, $\sigma_{\max}^2 = 8192$ covers approximately 72% of the profiled normalization operators, representing the majority of practical cases. These measurements show that our benchmark ranges are representative of real-model inputs.

5.3 End-to-End Performance

To further demonstrate the effectiveness of ROSETTA, we present end-to-end latency across different models and context lengths. Table 7 shows the end-to-end latency on CPU and GPU for GPT-2 Base, TinyLlama-1.1B, and LLaMA-3-8B at $n \in \{512, 2k, 8k\}$. Latency measures one-token generation (one decode iteration), excluding prefill. All methods share identical linear-layer implementations and differ only in nonlinear operators. Compared to CacheMir, ROSETTA achieves 1.5–2.1× end-to-end speedup across all models and context

lengths. NEXUS and MOAI are excluded as their Softmax implementation diverges at LLM-scale input ranges. Compared to PEGASUS, ROSETTA is 3–4× faster, as PEGASUS relies on per-element PBS for Softmax whose cost scales linearly with n . On GPU, ROSETTA achieves 1.4–2.0× speedup over CacheMir with a per-token latency of 1.3–6.8 m, demonstrating backend-agnostic gains and practical deployment potential.

5.4 End-to-end Model Perplexity Evaluation

We further evaluate end-to-end model quality with a ciphertext-aligned plaintext PPL test, where every nonlinear operation is replaced by its FHE counterpart. We test four 7–9 B models (LLaMA-3-8B, Qwen2-7B, Mistral-7B, DeepSeek-R1-Distill-LLaMA-8B) on WikiText-2/103 [36], LAMBADA [40], GSM8K [15], and ShareGPT, comparing ROSETTA ($N_{\text{seg}}=4$ segments) with PEGASUS. As reported in Table 8, ROSETTA keeps PPL degradation below +4.3% on every pair (average +1.4%), while PEGASUS ranges from +8% to +360% (average +116%). Such large degradations render outputs incoherent in practice: a single LUT cannot cover the wide input ranges of large LLMs, whereas our segmented LUT spans the full range with sufficient precision, validating the practicality of ROSETTA. We additionally evaluate CacheMir [55], NEXUS [59], and MOAI [60] on LLaMA-3-8B and DeepSeek-8B across all five datasets. CacheMir achieves PPL comparable to ROSETTA but incurs higher latency, whereas NEXUS/MOAI diverge because their fixed offline global maximum cannot cover LLM logit ranges.

6 Conclusion

We propose ROSETTA, a hybrid CKKS/TFHE framework for efficient and accurate privacy-preserving LLM decoding, featuring an adaptive segmented LUT protocol for accurate nonlinear evaluation and a scheme-aware operator selector that jointly optimizes per-operator scheme assignment and CKKS-level allocation via shortest-path search. Experiments show that ROSETTA achieves significant speedup over prior frameworks with negligible PPL degradation, taking a meaningful step toward practical secure LLM inference.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China (NSFC) under Grants 92464104, 62495102, and 62341407; in part by the National Key Research and Development Program under Grant 2024YFB4505004; in part by Ant Group through the CCF-Ant Research Fund under Grant CCF2412628150; in part by the Beijing Advanced Innovation Center for Future Blockchain and Privacy Computing under Grant GJJ-25-014; and in part by the 111 Project under Grant B18001.

References

- [1] 2024. Lattigo v6. Online: <https://github.com/tuneinsight/lattigo>. EPFL-LDS, Tune Insight SA.
- [2] Yoshimasa Akimoto, Kazuto Fukuchi, Youhei Akimoto, and Jun Sakuma. 2023. Privformer: Privacy-preserving Transformer with MPC. *2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P) (2023)*, 392–410. <https://api.semanticscholar.org/CorpusID:260387049>
- [3] Martin Albrecht, Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Shai Halevi, Jeffrey Hoffstein, Kim Laine, Kristin Lauter, Satya Lokam, Daniele Micciancio, Dustin Moody, Travis Morrison, Amit Sahai, and Vinod

- Vaikuntanathan. 2019. Homomorphic Encryption Standard. Cryptology ePrint Archive, Paper 2019/939. <https://eprint.iacr.org/2019/939>
- [4] Youngjin Bae, Jung Hee Cheon, Jaehyung Kim, Jai Hyun Park, and Damien Stehlé. 2023. HERMES: Efficient ring packing using MLWE ciphertexts and application to transciphering. In *Annual International Cryptology Conference*. Springer, 310–340.
 - [5] Song Bian, Zhou Zhang, Haowen Pan, Ran Mao, Zian Zhao, Yier Jin, and Zhenyu Guan. 2023. HE3DB: An Efficient and Elastic Encrypted Database Via Arithmetic-And-Logic Fully Homomorphic Encryption. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*. 2930–2944.
 - [6] Jean-Philippe Bossuat, Juan Ramón Troncoso-Pastoriza, and Jean-Pierre Hubaux. 2022. Bootstrapping for Approximate Homomorphic Encryption with Negligible Failure-Probability by Using Sparse-Secret Encapsulation. Cryptology ePrint Archive, Paper 2022/024. doi:10.1007/978-3-031-09234-3_26
 - [7] Christina Boura, Nicolas Gama, Mariya Georgieva, and Dimitar Jetchev. 2020. CHIMERA: Combining ring-LWE-based fully homomorphic encryption schemes. In *Journal of Mathematical Cryptology*, Vol. 14. De Gruyter, 316–338.
 - [8] Hao Chen, Wei Dai, Miran Kim, and Yongsoo Song. 2021. Efficient Homomorphic Conversion Between (Ring) LWE Ciphertexts. In *Applied Cryptography and Network Security: 19th International Conference, ACNS 2021, Kamakura, Japan, June 21–24, 2021, Proceedings, Part I* (Kamakura, Japan). Springer-Verlag, Berlin, Heidelberg, 460–479. doi:10.1007/978-3-030-78372-3_18
 - [9] Tianyu Chen, Hangbo Bao, Shaohan Huang, Li Dong, Binxing Jiao, Daxin Jiang, Haoyi Zhou, Jianxin Li, and Furu Wei. 2022. THE-X: Privacy-Preserving Transformer Inference with Homomorphic Encryption. In *Findings of the Association for Computational Linguistics: ACL 2022*, Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (Eds.). Association for Computational Linguistics, Dublin, Ireland, 3510–3520. doi:10.18653/v1/2022.findings-acl.277
 - [10] Peng Cheng and Utz Roedig. 2022. Personal Voice Assistant Security and Privacy—A Survey. *Proc. IEEE* 110 (2022), 476–507. <https://api.semanticscholar.org/CorpusID:247416823>
 - [11] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. 2018. Bootstrapping for Approximate Homomorphic Encryption. In *Advances in Cryptology – EUROCRYPT 2018*, Jesper Buus Nielsen and Vincent Rijmen (Eds.). Springer International Publishing, Cham, 360–384.
 - [12] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. 2017. Homomorphic encryption for arithmetic of approximate numbers. In *Advances in Cryptology – ASIACRYPT 2017*. Springer, 409–437.
 - [13] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. 2020. TFHE: fast fully homomorphic encryption over the torus. *Journal of Cryptology* 33, 1 (2020), 34–91.
 - [14] Wonhee Cho, Guillaume Hanrot, Taeseong Kim, Minje Park, and Damien Stehlé. 2024. Fast and Accurate Homomorphic Softmax Evaluation. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security* (Salt Lake City, UT, USA) (CCS '24). Association for Computing Machinery, New York, NY, USA, 4391–4404. doi:10.1145/3658644.3670369
 - [15] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168* (2021).
 - [16] Xianglong Deng, Shengyu Fan, Zhicheng Hu, Zhuoyu Tian, Zihao Yang, Jiangrui Yu, Dingyuan Cao, Dan Meng, Rui Hou, Meng Li, Qian Lou, and Mingzhe Zhang. 2024. Trinity: A General Purpose FHE Accelerator. arXiv:2410.13405 [cs.AR] <https://arxiv.org/abs/2410.13405>
 - [17] Han Ding, Yinhang Li, Junhao Wang, Hang Chen, Doudou Guo, and Yunbai Zhang. 2024. Large Language Model Agent in Financial Trading: A Survey. arXiv:2408.06361 [q-fin.TR] <https://arxiv.org/abs/2408.06361>
 - [18] Ye Dong, Wen-Jie Lu, Yancheng Zheng, Haoqi Wu, Derun Zhao, Jin Tan, Zhicong Huang, Cheng Hong, Tao Wei, Wen-Guang Chen, and Jianying Zhou. 2025. PUMA: Secure inference of LLaMA-7B in five minutes. *Security and Safety* 4 (2025), 2025014. doi:10.1051/sands/2025014
 - [19] Austin Ebel, Karthik Garimella, and Brandon Reagen. 2025. Orion: A Fully Homomorphic Encryption Framework for Deep Learning. arXiv:2311.03470 [cs.CR] <https://arxiv.org/abs/2311.03470>
 - [20] Craig Gentry, Shai Halevi, Chris Peikert, and Nigel P Smart. 2013. Field switching in BGV-style homomorphic encryption. *Journal of Computer Security* 21, 5 (2013), 663–684.
 - [21] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. 2016. CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy. In *Proceedings of The 33rd International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 48)*, Maria Florina Balcan and Kilian Q. Weinberger (Eds.). PMLR, New York, New York, USA, 201–210. <https://proceedings.mlr.press/v48/giladbachrach16.html>
 - [22] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. 2024. The Llama 3 Herd of Models. arXiv:2407.21783 [cs.AI] <https://arxiv.org/abs/2407.21783>
 - [23] Kanav Gupta, Neha Jawalkar, Ananta Mukherjee, Nishanth Chandran, Divya Gupta, Ashish Panwar, and Rahul Sharma. 2023. SIGMA: Secure GPT Inference with Function Secret Sharing. Cryptology ePrint Archive, Paper 2023/1269. <https://eprint.iacr.org/2023/1269>
 - [24] Meng Hao, Hongwei Li, Hanxiao Chen, Pengzhi Xing, Guowen Xu, and Tianwei Zhang. 2022. Iron: Private inference on transformers. *Advances in Neural Information Processing Systems* 35 (2022), 15718–15731.
 - [25] Zhicong Huang, Wen-jie Lu, Cheng Hong, and Jiansheng Ding. 2022. Cheetah: Lean and fast secure Two-Party deep neural network inference. In *31st USENIX Security Symposium (USENIX Security 22)*. 809–826.
 - [26] Wen jie Lu, Zhicong Huang, Zhen Gu, Jingyu Li, Jian Liu, Cheng Hong, Kui Ren, Tao Wei, and WenGuang Chen. 2023. BumbleBee: Secure Two-party Inference Framework for Large Transformers. Cryptology ePrint Archive, Paper 2023/1678. <https://eprint.iacr.org/2023/1678>
 - [27] Wen jie Lu, Zhicong Huang, Cheng Hong, Yiping Ma, and Hunter Qu. 2021. PEGASUS: Bridging Polynomial and Non-polynomial Evaluations in Homomorphic Encryption. Cryptology ePrint Archive, Paper 2020/1397. <https://eprint.iacr.org/2020/1397>
 - [28] Jae Hyung Ju, Jaiyoung Park, Jongmin Kim, Minsik Kang, Donghwan Kim, Jung Hee Cheon, and Jung Ho Ahn. 2024. NeuJeans: Private Neural Network Inference with Joint Optimization of Convolution and Bootstrapping. arXiv:2312.04356 [cs.CR] <https://arxiv.org/abs/2312.04356>
 - [29] Chirag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. 2018. Gazelle: A Low Latency Framework for Secure Neural Network Inference. arXiv:1801.05507 [cs] (2018). <http://arxiv.org/abs/1801.05507>
 - [30] Andes Y. L. Kei and Sherman S. M. Chow. 2025. SHAFT: Secure, Handy, Accurate, and Fast Transformer Inference. In NDDS.
 - [31] Dongwoo Kim and Cyril Guyot. 2023. Optimized Privacy-Preserving CNN Inference With Fully Homomorphic Encryption. *IEEE Transactions on Information Forensics and Security* 18 (2023), 2175–2187. doi:10.1109/TIFS.2023.3263631
 - [32] Huan Yee Koh, Jiaxin Ju, Ming Liu, and Shirui Pan. 2022. An Empirical Survey on Long Document Summarization: Datasets, Models and Metrics. *ACM Comput. Surv.* (jun 2022). doi:10.1145/3545176
 - [33] Eunsang Lee, Joon-Woo Lee, Junghyun Lee, Young-Sik Kim, Yongjune Kim, Jong-Seon No, and Woosuk Choi. 2021. Low-Complexity Deep Convolutional Neural Networks on Fully Homomorphic Encryption Using Multiplexed Parallel Convolutions. Cryptology ePrint Archive, Paper 2021/1688. <https://eprint.iacr.org/2021/1688>
 - [34] Dacheng Li, Rulin Shao, Hongyi Wang, Han Guo, Eric P. Xing, and Hao Zhang. 2023. MPCFormer: fast, performant and private Transformer inference with MPC. arXiv:2211.01452 [cs.LG] <https://arxiv.org/abs/2211.01452>
 - [35] Qian Lou and Lei Jiang. 2021. HEMET: A Homomorphic-Encryption-Friendly Privacy-Preserving Mobile Neural Network Architecture. In *Proceedings of the 38th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 139)*, Marina Meila and Tong Zhang (Eds.). PMLR, 7102–7110. <https://proceedings.mlr.press/v139/lou21a.html>
 - [36] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer Sentinel Mixture Models. arXiv:1609.07843 [cs.CL]
 - [37] Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. 2020. Delphi: A Cryptographic Inference Service for Neural Networks. In *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, 2505–2522. <https://www.usenix.org/conference/usenixsecurity20/presentation/mishra>
 - [38] Jungho Moon, Dongwoo Yoo, Xiaoqian Jiang, and Miran Kim. 2025. THOR: Secure Transformer Inference with Homomorphic Encryption. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security (Taipei, Taiwan) (CCS '25)*. Association for Computing Machinery, New York, NY, USA, 3765–3779. doi:10.1145/3719027.3765150
 - [39] Qi Pang, Jinhao Zhu, Helen Möllering, Wenting Zheng, and Thomas Schneider. 2023. BOLT: Privacy-Preserving, Accurate and Efficient Inference for Transformers. Cryptology ePrint Archive, Paper 2023/1893. <https://eprint.iacr.org/2023/1893>
 - [40] Denis Paperno, Germán Kruszewski, Angeliki Duchi, and Marco Baroni. 2016. The LAMBADA dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*.
 - [41] Dongjin Park, Eunsang Lee, and Joon-Woo Lee. 2024. Powerformer: Efficient and High-Accuracy Privacy-Preserving Language Model with Homomorphic Encryption. Cryptology ePrint Archive, Paper 2024/1429. <https://eprint.iacr.org/2024/1429>
 - [42] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language Models are Unsupervised Multitask Learners. *OpenAI* (2019). https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf Accessed: 2024-11-15.
 - [43] Fahad Shamsah, Salman Hameed Khan, Syed Waqas Zamir, Muhammad Haris Khan, Munawar Hayat, Fahad Shahbaz Khan, and H. Fu. 2022. Transformers in Medical Imaging: A Survey. *Medical Image Analysis* 88 (2022), 102802. <https://api.semanticscholar.org/CorpusID:246240729>

- [44] Hugo Touvron et al. 2023. Llama 2: Open Foundation and Fine-Tuned Chat Models. arXiv:2307.09288 [cs.CL] <https://arxiv.org/abs/2307.09288>
- [45] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (Long Beach, California, USA) (NIPS'17). Curran Associates Inc., Red Hook, NY, USA, 6000–6010.
- [46] Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2024. Efficient Streaming Language Models with Attention Sinks. arXiv:2309.17453 [cs.CL] <https://arxiv.org/abs/2309.17453>
- [47] Tianshi Xu, Meng Li, and Runsheng Wang. 2024. HEQuant: Marrying Homomorphic Encryption and Quantization for Communication-Efficient Private Inference. arXiv:2401.15970 [cs.CR] <https://arxiv.org/abs/2401.15970>
- [48] Tianshi Xu, Meng Li, Runsheng Wang, and Ru Huang. 2023. Falcon: Accelerating Homomorphically Encrypted Convolutions for Efficient Private Mobile Network Inference. *arXiv preprint arXiv:2308.13189* (2023).
- [49] Tianshi Xu, Wen-jie Lu, Jiangrui Yu, Yi Chen, Chenqi Lin, Runsheng Wang, and Meng Li. 2025. Breaking the layer barrier: remodeling private transformer inference with hybrid CKKS and MPC. In *Proceedings of the 34th USENIX Conference on Security Symposium* (Seattle, WA, USA) (SEC '25). USENIX Association, USA, Article 137, 20 pages.
- [50] Tianshi Xu, Lemeng Wu, Runsheng Wang, and Meng Li. 2024. PrivCirNet: Efficient Private Inference via Block Circulant Transformation. *arXiv preprint arXiv:2405.14569* (2024).
- [51] Tianshi Xu, Shuzhang Zhong, Wenxuan Zeng, Runsheng Wang, and Meng Li. 2024. PrivQuant: Communication-Efficient Private Inference with Quantized Network/Protocol Co-Optimization. *arXiv preprint arXiv:2410.09531* (2024).
- [52] Hao Yang, Shiyu Shen, Wangchen Dai, Lu Zhou, Zhe Liu, and Yunlei Zhao. 2023. Phantom: A CUDA-Accelerated Word-Wise Homomorphic Encryption Library. *Cryptology ePrint Archive, Paper 2023/049*. doi:10.1109/TDSC.2024.3363900
- [53] Jiangrui Yu, Yi Chen, and Meng Li. 2026. PEFT: a near-memory processing-enabled heterogeneous accelerator for BatchPBS TFHE. In *2026 IEEE International Symposium on Circuits and Systems (ISCAS)*. 4093–4097. doi:10.1109/ISCAS66217.2026.11562611
- [54] Jiangrui Yu, Wenxuan Zeng, Tianshi Xu, Renze Chen, Yun Liang, Runsheng Wang, Ru Huang, and Meng Li. 2025. *FlexHE: A flexible Kernel Generation Framework for Homomorphic Encryption-Based Private Inference*. Association for Computing Machinery, New York, NY, USA. <https://doi.org/10.1145/3676536.3676739>
- [55] Ye Yu, Yifan Zhou, Yi Chen, Pedro Soto, Wenjie Xiong, and Meng Li. 2026. Cachemir: Fully Homomorphic Encrypted Inference of Generative Large Language Model with KV Cache. arXiv:2602.11470 [cs.CR] <https://arxiv.org/abs/2602.11470>
- [56] Wenxuan Zeng, Ye Dong, Jinjin Zhou, Jin Tan, Lei Wang, Tao Wei, Runsheng Wang, and Meng Li. 2025. MPCache: MPC-Friendly KV Cache Eviction for Efficient Private LLM Inference. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*. <https://openreview.net/forum?id=kd6hcHU9C>
- [57] Wenxuan Zeng, Meng Li, Wenjie Xiong, Tong Tong, Wen-jie Lu, Jin Tan, Runsheng Wang, and Ru Huang. 2023. MPCViT: Searching for Accurate and Efficient MPC-Friendly Vision Transformer with Heterogeneous Attention. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 5052–5063.
- [58] Wenxuan Zeng, Tianshi Xu, Meng Li, and Runsheng Wang. 2024. EQO: Exploring Ultra-Efficient Private Inference with Winograd-Based Protocol and Quantization Co-Optimization. arXiv:2404.09404 [cs.CR] <https://arxiv.org/abs/2404.09404>
- [59] Jiawen Zhang, Xinpeng Yang, Lipeng He, Kejia Chen, Wen-jie Lu, Yinghao Wang, Xiaoyang Hou, Jian Liu, Kui Ren, and Xiaohu Yang. 2024. Secure Transformer Inference Made Non-interactive. *Cryptology ePrint Archive, Paper 2024/136*. <https://eprint.iacr.org/2024/136>
- [60] Linru Zhang, Xiangning Wang, Jun Jie Sim, Zhicong Huang, Jiahao Zhong, Huaxiong Wang, Pu Duan, and Kwok Yan Lam. 2025. MOAI: Module-Optimizing Architecture for Non-Interactive Secure Transformer Inference. *Cryptology ePrint Archive, Paper 2025/991*. <https://eprint.iacr.org/2025/991>
- [61] Peiyuan Zhang, Guangtao Zeng, Tianduo Wang, and Wei Lu. 2024. TinyLlama: An Open-Source Small Language Model. arXiv:2401.02385 [cs.CL] <https://arxiv.org/abs/2401.02385>
- [62] Tengyu Zhang, Chenqi Lin, Jiangrui Yu, Yi Chen, Shuwen Deng, and Meng Li. 2025. (Invited) FENIX: Flexible and Efficient Hybrid HE/MPC Acceleration with Near-Memory Processing. In *2025 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. 1–9. doi:10.1109/ICCAD66269.2025.11240701
- [63] Z. Zhang et al. 2024. LOHEN: Layer-wise Optimizations for Neural Network Inferences over Encrypted Data with High Performance or Accuracy. In *USENIX Security Symposium*.
- [64] Yifan Zhou, Tianshi Xu, Jue Hong, Ye Wu, and Meng Li. 2025. CryptoMoE: Privacy-Preserving and Scalable Mixture of Experts Inference via Balanced Expert Routing. In *The Thirty-ninth Annual Conference on Neural Information Processing*

Systems. <https://openreview.net/forum?id=8pEqyGri>

A Open Science

To promote availability, all pre-trained models used in this work (GPT-2, TinyLlama-1.1B, LLaMA-3-8B, Qwen2-7B, Mistral-7B, and DeepSeek-R1-Distill-LLaMA-8B) and the WikiText-2 evaluation corpus are publicly accessible through their original sources, as referenced in the paper. Our source code and materials for replicating the experiments are publicly available at https://anonymous.4open.science/r/ROSETTA_artifact-B77F/. We welcome feedback and contributions to improve the implementation and further the research in privacy-preserving inference.

B Ethical Considerations

This work focuses on improving the efficiency of privacy-preserving inference frameworks for generative large language models. Our contributions are intended to advance private computation techniques without creating new risks to data security or user privacy. By enhancing the practicality and scalability of privacy-preserving machine learning, we hope to encourage the responsible and broader uptake of privacy-centric technologies.

We strictly comply with the ACM Code of Ethics. In particular, our work is guided by the principles of respecting user privacy (Principle 1.6) and avoiding harm (Principle 1.2). All models and datasets employed in our methodology, including GPT-2, TinyLlama-1.1B, LLaMA-3-8B, Qwen2-7B, Mistral-7B, DeepSeek-R1-Distill-LLaMA-8B, WikiText, LAMBADA, GSM8K, and ShareGPT, are publicly accessible. Our experiments do not intentionally collect or process proprietary, sensitive, or personally identifiable information. ROSETTA runs entirely under FHE on ciphertext-aligned activations and weights, and does not introduce additional privacy risk beyond that of the underlying CKKS and TFHE primitives or the threat model assumed by prior FHE-LLM systems. We have carefully considered the ethical implications of our segmented LUTs and scheme-aware operator selector, and confirm that no privacy violations arise from our experimentation or proposed methods.

In summary, because this research does not involve human subjects and does not intentionally collect or process proprietary, sensitive, or personally identifiable information, we identify no need for formal human-subject review. By advancing the practicality of privacy-preserving machine learning, we believe this work helps promote the responsible adoption of privacy-enhancing technologies and reinforces data minimization and confidentiality.

C Generative AI Disclosure

Generative AI tools such as GPT, Codex, and Gemini were employed to support code implementation, benchmark configuration, $\LaTeX$ typesetting, and the editing and polishing of portions of the manuscript. All AI-assisted content, including prose, code, experimental results, and citations, was thoroughly inspected, validated, and revised by the authors, who assume full responsibility for the accuracy, originality, and integrity of the work presented in this paper.