SoK: Private LLM Inference using Approximate Homomorphic Encryption

Ahmad Al Badawi, Andreea Alexandru, Yuriy Polyakov
Duality Technologies
Email: {aalbadawi, aalexandru, ypolyakov}@dualitytech.com

Vinod Vaikuntanathan
Duality Technologies and MIT
Email: vinodv@mit.edu

***Abstract*—Although recent surveys on privacy-enhancing technologies concluded that FHE cannot feasibly evaluate non-linear activation functions in modern ML architectures, 21 CKKS-based frameworks have since demonstrated end-to-end private inference of LLMs with up to 8 B parameters. However, as the field grows rapidly, the literature has become fragmented. Frameworks differ in ciphertext packing layouts, model fidelity, software and hardware stacks, and reported metrics, that hinder direct comparison and reproducibility. This paper presents the first systematization of knowledge for non-interactive, CKKS-based private LLM inference. We categorize the design space across two axes: a model-level axis (packing layout for linear blocks and model preservation for non-linear blocks) and a system-level axis (covering hardware, compilers, bootstrapping management, and hybrid execution). To standardize reporting framework configurations and results among researchers, we propose a Private LLM Card System (PLCS). Furthermore, we introduce POLARIS, a model-preserving reference framework for CKKS-based private LLM inference and provide it as an open-source proof-of-concept[1]. POLARIS evaluates encrypted inference for the entire BERT family, from BERT-Tiny to BERT-Base, on CPU and GPU backends, the first framework to demonstrate this.**

Our analysis shows that only 5 out of the 21 surveyed frameworks are model-preserving, that is, they evaluate standard, unmodified LLMs without retraining or architectural substitutions while keeping the entire prompt encrypted. We also show that model-preserving CKKS approximations maintain high downstream task accuracy from BERT-Tiny up to Llama-3-8 B. We identify a runtime gap of roughly four orders of magnitude between the best encrypted and plaintext inference as the primary barrier to practical use. While our analysis confirms that CKKS-based inference is now algorithmically feasible for non-linear functions and large models, we conclude that it remains operationally impractical for many human-facing applications until the efficiency gap is narrowed. We outline promising research directions to close this gap.

1. Introduction

Large Language Models (LLMs) have moved from research artifacts to critical infrastructure: medical professionals apply them to patient records [1], lawyers upload case files into chat interfaces [2], and financial analysts query proprietary market reports to automate risk assessment [3]. In these interactions, the user typically sends a prompt, and often additional context material, to the model host, which raises privacy concerns that neither usage agreements nor access controls can fully address [4].

Among the various privacy-enhancing technologies (PETs) proposed to mitigate these privacy concerns, cryptographic approaches offer attractive solutions; in particular, *non-interactive* private inference via Fully Homomorphic Encryption (FHE). In this approach, the user encrypts her private query into a ciphertext and sends it to the server, and then the server computes on the ciphertext to generate an encrypted response that is transmitted to the user for decryption. Unlike other PETs, there is no online interaction beyond a single round-trip as in the multi-round protocols typically used in Multi-Party Computation (MPC), no privacy budget to deplete (as in Differential Privacy (DP)), and no trusted hardware prone to side-channel attacks (as in Trusted Execution Environments (TEEs)). However, the computational overhead associated with FHE is typically much higher than for other PETs. Within FHE, the Cheon-Kim-Kim-Song (CKKS) scheme [5] has become the de facto choice for LLM inference, as it natively supports Single Instruction, Multiple Data (SIMD)-packed approximate homomorphic arithmetic over floating-point numbers [6], [7], [8], [9], [10].

Over the past 3 years, we have identified more than 21 distinct CKKS-based private LLM inference frameworks, scaling from Bidirectional Encoder Representations from Transformers (BERT)-Tiny on CPU [11], [12], [13] to BERT-Base on GPU clusters [14], [15], [16], and from Generative Pre-trained Transformer (GPT)-2 [17] to Large Language Model Meta AI (Llama)-3 [18], [19], [20]. New algorithmic innovations have provided efficient solutions to components previously declared intractable, such as the normalize-and-square Softmax [21], and the logarithmic-depth $\arg\max$ of NEXUS [15]. On the systems side, GPU-accelerated CKKS bootstrapping has dropped from tens of seconds on CPU [22] to under 10 milliseconds on a multi-GPU cluster [18], and open-source CKKS-on-GPU backends [17], [23] now make these speedups reproducible for the broader research community.

The rapid growth, however, has produced a fragmented

1. Code will be released as open-source upon formal publication.

evaluation landscape that is hard to navigate via direct comparison. Proposed frameworks differ along many axes, including: 1) ciphertext packing designs (token/row vs. column/diagonal vs. complex/cyclic vs. algorithmic shortcuts), 2) model fidelity (model-preserving approximation vs. retrained architectural replacements), 3) cryptographic parameters (security level, secret key distributions), 4) software stacks (OpenFHE, SEAL, Lattigo, HEaaN, Phantom, FIDESlib, Liberate), 5) hardware platforms (consumer CPUs to data center GPU clusters), 6) evaluation metrics (amortized vs. unamortized latency, time-to-first-token vs. per-token latency, with or without sequence lengths), 7) licensing and distribution models (permissive open-source vs. proprietary closed-source binaries), and 8) implementation levels (out-of-the-box end-to-end pipeline vs. modular, low-level building blocks).

Existing surveys do not provide the cross-framework synthesis needed to close the gap in comparative evaluation. The most recent Privacy-Preserving Machine Learning (PPML) surveys or Systematization of Knowledge (SoK) papers [24], [25], [26], [27], [28], [29], [30], [31], [32], [33] cover MPC, TEE, DP, and FHE together and explicitly identify the lack of standardized LLM-inference benchmarks as an open problem [26], [34]. In their *lifecycle* survey [31], the authors treat both FHE and MPC at equal depth and do not analyze per-operator CKKS design choices. Earlier surveys [24], [25], [29], [30], [35] were published before the LLM era and concluded that standard activation functions (such as ReLU, Softmax, GELU) cannot be realized efficiently in FHE [29], [30], [31] or that pure FHE inference is "progressively losing interest" in favor of hybrid protocols [24]. Notably, a recent measurement study [36] questions the ability of CKKS to perform a deep computation mixing linear and non-linear operations, which they argue using a notion of "computational universality" and experimental evaluation for machine learning workloads. However, a large number of recent frameworks implementing end-to-end FHE evaluations of large-scale models (e.g., Llama-3 and GPT-2) claim that CKKS polynomial approximations can evaluate complex LLMs fully under encryption without prohibitive accuracy degradation.

To address this gap, we present the first SoK dedicated exclusively to CKKS-based, non-interactive private LLM inference. Our work is driven by the following research questions:

- **RQ1** What approaches lead to model-preserving[2] private evaluation of LLM inference and can ultimately be used for production-ready frameworks?
- **RQ2** Can CKKS-based solutions achieve practically acceptable accuracy in evaluating widely adopted LLMs?
- **RQ3** What is the current efficiency slowdown of the state-of-the-art (SoA) CKKS-based LLMs compared to in-the-clear GPU implementations?
- **RQ4** What are the main challenges and open problems that prevent CKKS-based LLMs from becoming practical?

The overarching goals of our study are to (1) assess the current state of FHE-based solutions in terms of accuracy, efficiency, and privacy, and (2) develop research recommendations for building practical private LLM frameworks using FHE.

To answer **RQ1**, we develop a taxonomy of the linear and non-linear building blocks of LLM architectures. We identify which non-linear blocks lead to model-preserving designs and which do not. Finally, we discuss hybrid public-private methods that deviate from the privacy goals of private LLM inference.

To address **RQ2**, we review approximate CKKS arithmetic and explain its differences from conventional floating-point LLM compute models used in the clear. Our analysis of existing private LLM solutions shows that CKKS is able to achieve acceptable accuracy for 8-billion-parameter models because 1) the precision requirements of LLMs are low, making 16-bit floats acceptable (in cleartext implementations) for many scenarios, and 2) the linear blocks perform aggregation that can often "stabilize" the stochastic error due to averaging effects. Therefore, we anticipate that acceptable accuracy can be achieved for even larger ($\geq$ 100 B) models that are used in practice.

After identifying model-preserving methods for private LLM inference and showing that acceptable accuracy can be achieved, we present POLARIS, an open-source reference implementation for model-preserving private LLM inference. To our knowledge, POLARIS is the first framework that evaluates the entire BERT family, from BERT-Tiny to BERT-Base [37]. The motivation for POLARIS is to provide a reference framework that other researchers can use to develop implementations for more advanced LLMs and to benchmark their accuracy and efficiency.

To address **RQ3**, we identify the challenges in standardized benchmarking of existing solutions, and propose a four-cards reporting scheme, a concept we adopt from the machine learning domain to categorize parameters by model, CKKS scheme, system deployment, and downstream application. After that, we present a table evaluating the performance of all LLM solutions based on the classes and categories selected using our methodology. Finally, we compare the experimental results for the largest model-preserving implementations with cleartext benchmarks. Our analysis shows that the best private LLM inference results are currently four orders of magnitude slower than the optimized cleartext implementations on the same GPU systems.

Finally, to address **RQ4**, we identify efficiency and robustness as two major practicality challenges for private LLM inference using CKKS. We discuss potentially promising research directions to advance the algorithms and highlight specialized hardware acceleration systems as ways to reduce the efficiency gap between private LLM inference using CKKS and the cleartext instantiation.

Our main contributions can be summarized as follows:

2. An approach that works with a standard cleartext model and does not require model retraining or fine-tuning.

1) **A two-axis taxonomy of model-level design choices**: linear blocks and non-linear blocks. We provide recommendations for selecting these components and place every surveyed framework on this grid (§3).
2) **A system-level systematization** covering bootstrapping management, hardware/compiler stacks, and hybrid public-private inference (§3.3), each accompanied by recommended selection criteria.
3) **POLARIS**, an **open-source reference** CKKS framework that combines SoA's most stable and model-preserving building blocks (§3.4.1). To support reproducibility, we will release POLARIS as open-source upon publication.
4) **The Private LLM Card System (PLCS)**, a four-card reporting scheme (Model, Cryptographic, System, Application) that comprehensively specifies framework configurations and evaluation experiments (§4).
5) **A comprehensive comparison** of 21 frameworks organized by downstream task and model scale (§4.3, Tables 1 and 2), and **cross-framework insights** (§4.4) that can be drawn from our analysis.

2. Background and Preliminaries

This section introduces the core concepts the paper builds on.

2.1. LLM Architecture: Overview

The LLMs considered in our SoK are primarily based on the *Transformer* architecture [38], which stacks a number of identical transformer layers. Each layer combines a multi-head attention sub-layer with a Feed-Forward Neural Network (FFNN) sub-layer. The surveyed frameworks share this basic architecture but differ in their attention masking, layer normalization, and activation functions. Specifically, they diverge into three distinct architectures: *encoder-only* (e.g., BERT), *decoder-only* (e.g., GPT), and *optimized decoder* (e.g., Llama). The surveyed frameworks target two downstream tasks: sequence classification and autoregressive token generation. Further details on the architectures can be found in Appendix A.

2.2. System and Threat Model

We consider a two-party privacy-preserving Machine Learning as a Service (MLaaS) framework, where a *Client* aims to perform private inference using a public LLM model hosted by a not necessarily trusted *Server*. The main goals are to offload computations to the server and protect the privacy of the client's prompts by ensuring the server performs inference on encrypted data without accessing the underlying plaintext. The framework assumes a single-round communication protocol: the client transmits the encrypted prompt along with the necessary public key material, and the server computes and returns the encrypted results without ever accessing the underlying data. We adopt the standard *semi-honest* threat model for outsourced computation. Concrete details of this model can be found in Appendix B.

2.3. CKKS Machinery

Ordinary cleartext computation easily and efficiently handles 64-, 32-, or 16-bit floating-point numbers. In contrast, CKKS performs *fixed-point* arithmetic with truncation, i.e., it cannot manipulate the mantissa and exponent separately. Emulating true floating-point arithmetic using FHE has historically been extremely challenging [39]. The following are some of the computational constraints of CKKS. The formal CKKS algorithmic description is given in Appendix C.

2.3.1. SIMD and Basic Operations. CKKS packs vectors of real or complex numbers into high-degree polynomials (e.g., 2^{15} or 2^{16}). This provides SIMD parallelism, meaning an operation on a ciphertext applies simultaneously to thousands of packed *slots*. However, CKKS has a limited instruction set comprising only three native vector operations: element-wise addition, element-wise (Hadamard) multiplication, and cyclic slot rotation. The latter can be used to access individual vector elements.

The computational complexity of these operations varies significantly. While additions are computationally cheap, homomorphic multiplications (Ct-Ct) and rotations are expensive because they require a costly maintenance operation called *Key-Switching*. Additionally, multiplications incur the highest noise growth and are often followed by *Rescaling* to reduce the scaling factor degree from Δ^2 back to Δ. For a comprehensive CKKS tutorial, the reader is referred to [40].

2.3.2. Polynomial Approximation. Since it only supports addition and multiplication, CKKS does not natively allow computing non-polynomial functions like division, exponentiation, and square root. Instead, these functions are usually approximated as polynomials using *Minimax* or *Chebyshev* expansions [41]. Using the Paterson-Stockmeyer algorithm [42], a polynomial of degree t can be evaluated in a circuit of depth $O(\log t)$. The accuracy of a polynomial approximation depends on the input interval, target precision, and polynomial degree. The main challenge is balancing these factors: higher degrees improve accuracy but increase homomorphic depth and noise, while the approximation itself must avoid function singularities and optimize the input interval.

2.3.3. Multiplicative Depth and Bootstrapping. CKKS is a leveled FHE scheme. Every homomorphic multiplication increases ciphertext noise and consumes a multiplicative level. For instance, multiplying two ciphertexts ct_a at level l_a and ct_b at level l_b results in a ciphertext at level $l_{out} = \min(l_a, l_b) - 1$ due to rescaling. When the depth budget runs out, bootstrapping is called to restore some levels[3]. CKKS bootstrapping is extremely expensive and should be used sparingly. Therefore, a primary design principle in FHE development is the minimization of the target computation's multiplicative depth.

3. Note that bootstrapping itself consumes L_{BT} levels.

2.3.4. Arithmetic Precision. Unlike exact FHE schemes (such as Brakerski-Gentry-Vaikuntanathan (BGV) [43] and Brakerski-Fan-Vercauteren (BFV) [44], [45]), where the Learning with Errors (LWE) noise is independent from the plaintext message, CKKS is inherently approximate by design. The message and LWE noise are combined and cannot be separated. Hence, precision loss in CKKS is irreversible. Each homomorphic operation introduces additional computational noise into the ciphertext. The precision loss is estimated to be nearly one bit after each Rescaling operation [5]; however, in practice, we rely on the probabilistic nature of the CKKS noise. As deep computations include large-scale aggregations, such as matrix-vector operations, the noise benefits from statistical averaging effects and grows more slowly than what worst-case noise analysis predicts [46].

2.4. LLM Building Blocks

Contemporary Transformer architectures such as BERT, GPT, and Llama consist of stacked blocks that alternate between massive linear transforms and non-linear activation functions:

- **Linear Blocks**: high-dimensional matrix multiplications account for most of the model's parameters and computations. These occur in Query, Key, and Value projections, attention score calculations, and FFNNs.
- **Non-Linear Blocks**: Transformers use non-polynomial activation and normalization functions. For example, Softmax requires exponentiation and division, LayerNorm and RMSNorm require evaluating inverse square roots, and GELU, SiLU, or $\tanh$ require evaluating the error function or Sigmoid.

2.5. How CKKS Meets LLM Requirements

To perform private LLM inference, standard LLM components must be translated into CKKS-supported operations.

2.5.1. Handling Linear Blocks. Linear blocks can be computed natively in CKKS. However, because CKKS ciphertexts are 1D vectors, matrix operations must be packed into 1D SIMD slots. Matrix multiplication can then be realized by using homomorphic rotations, element-wise multiplications, and additions. Tiling techniques might be required to partition large matrices across multiple ciphertexts. In CKKS, it is important to distinguish between two types of matrix multiplications based on what is being computed:

- **Ciphertext-Plaintext**: this occurs when multiplying encrypted data by unencrypted model weights, such as in Query, Key, and Value projections, and FFNN. This operation is computationally cheaper and adds less noise.
- **Ciphertext-Ciphertext**: this occurs when multiplying two encrypted operands together, such as when calculating attention scores from encrypted Queries and Keys. This operation is significantly more expensive as it requires relinearization.

2.5.2. Handling Non-Linear Blocks. Non-linear functions (e.g., Softmax, GELU, and normalization) cannot be computed natively in CKKS. Typically, they are either approximated using polynomials or replaced entirely with simpler polynomial alternatives. Note that there are more recent techniques to evaluate non-linear functions using lookup tables [47], but their practical efficiency has yet to be proven.

2.5.3. Managing Multiplicative Depth. The forward pass of LLMs (typically 1 to 80 layers) can easily exceed the multiplicative depth budgets of practical CKKS instantiations. This requires careful placement of bootstrapping operations to restore the multiplicative depth budget. It is well established that the general bootstrapping placement problem is NP-hard [48]. Therefore, the majority of the surveyed frameworks use manual placement of bootstrapping, given that the transformer architecture is both iterative and limited in computational branches.

3. Systematization of Framework Designs

CKKS limits computation to approximate arithmetic: addition, multiplication, and cyclic rotations. To run private LLM inference under CKKS, frameworks must overcome limitations at two levels: at the model level, where the core mapping of linear and non-linear blocks determines feasibility, and at the system level, which addresses bootstrapping, hardware, and deployment. This section systematizes the design space of existing CKKS-based private LLM inference frameworks by analyzing key design choices across 21 SoA frameworks. Finally, we present POLARIS, our open-source reference framework for CKKS-based private LLM inference.

3.1. Selection Process

As mentioned earlier, we have identified 21 relevant frameworks for our study. We included frameworks that satisfy all of the following conditions:

1) the framework is primarily based on the CKKS scheme,
2) adopts a non-interactive protocol (one client-server round trip), and
3) reports performance runtime of encrypted Transformers-based LLM inference.

Our focus on CKKS stems from its widespread adoption in encrypted machine learning applications, thanks to its capabilities in handling floating-point arithmetic and rescaling efficiently. Other schemes such as BFV and BGV support exact integer arithmetic and require special and costly treatment for handling floating-point applications.

3.2. Model-Level Systematization

The model level addresses how LLM computation is mapped to CKKS. This includes mapping both linear and non-linear blocks to CKKS operations.

3.2.1. Systematization of Linear Blocks. Linear operations require mapping matrix multiplications into 1D ciphertext slots. The specific requirements are Ciphertext-Plaintext Matrix Multiplication (CPMM) for static weights (e.g., $\mathbf{Q}$, $\mathbf{K}$, $\mathbf{V}$, and FFNN linear blocks) and Ciphertext-Ciphertext Matrix Multiplication (CCMM) for dynamic attention scores $\mathbf{S}$. The primary goal across SoA frameworks is to design *Packing Layouts* to optimize expensive ciphertext rotations and multiplicative depth. We provide the following taxonomy of linear blocks based on data packing layouts. Note that some frameworks employ a combination of the following classes.

[Class L-ROW] Token/Row Packing. In this layout, each d-vector token is mapped to a separate ciphertext. While intuitive, it underutilizes the available SIMD slot capacity of the ciphertext. Frameworks such as RL [11], SecretVoice [49], PRITRAN [12], and ATLAS [50] use this layout and apply various heuristic optimizations to increase packing density. An important advantage of this approach is its KV-cache[4]-friendly nature, because tokens reside in separate ciphertexts, they can be easily reused during the decoding phase of autoregressive token generation [51].

[Class L-COL] Column/Diagonal Packing. To reduce the number of inner rotations during CPMMs, some frameworks (such as ARION [13], MOAI [14], THOR [16], and Powerformer [52]) pack data by columns or diagonals [53] across the feature level.

[Class L-CYC] Complex/Cyclic Encoding. Some frameworks apply innovative optimizations to exploit unused ciphertext dimensions. STIP [20] computes parallel outputs using both real and imaginary slots. TRICYCLE [54], CACHEMIR [55], and NEXUS [15] use cyclic or interleaved encodings to process multiple attention heads at once.

[Class L-ALG] Algorithmic Shortcuts. Some frameworks shift their focus to mathematical optimizations rather than the packing layout. For instance, ENSI [56] uses *BitNet*'s ternary weights [57] $\{-1, 0, 1\}$ to replace costly homomorphic multiplications with cheap additions. Similarly, ELLMO [58], EUSTON [19], and SplitPrompt [59] use pre-computed plaintext transformations, inspired by [60], or Singular Value Decomposition (SVD) to minimize the multiplicative depth.

3.2.2. Systematization of Non-Linear Blocks. Non-linear operations that are not CKKS-friendly (such as division, exponentiation, and error function) incur a significant cost to compute natively in CKKS. Frameworks solve this challenge either by mathematically approximating the target function in a *model-preserving* manner or by entirely modifying the non-linear operations in a *model-modifying* manner.

Based on our analysis of SoA frameworks, we systematize the design space of non-linear blocks using a taxonomy categorized by model preservation:

[Class N-PRE] Model-Preserving Frameworks. Frameworks in this class aim to preserve the original LLM architecture by using numerical methods to evaluate non-linear operations homomorphically. Because this approach does not alter the original activation functions, it eliminates the need for costly retraining or fine-tuning typically caused by training and inference-time activation mismatches.

- **The Softmax$(\cdot)$ Block:** SplitPrompt solves the Softmax problem over large input intervals by using an iterative *normalize-and-square* algorithm [21]. By scaling inputs down by a negative power of two, it reduces the evaluation of $\exp(\cdot)$ to a stable interval before recursively squaring the results to retain the full original interval. Similarly, THOR optimizes the multiplicative depth of normalize-and-square using Goldschmidt's division algorithm paired with Adaptive Successive Over-Relaxation (aSOR).
- **The Inverse $\sqrt{\cdot}$ Block:** The inverse $\sqrt{\cdot}$ function is primarily used in LayerNorm to divide by the standard deviation of the input vector. Frameworks (like THOR, CEP [17], MOAI, and EFLA [61]) use Newton-Raphson iterations and Goldschmidt's algorithm to approximate this function. Note that the normalize-and-square adaptation of Softmax introduces this function as well.
- **The GELU$(\cdot)$ and $\tanh(\cdot)$ Blocks:** Most frameworks (such as ARION, ELLMO, TRICYCLE, and RL) rely on high-degree polynomials to approximate these functions. In contrast, THOR uses function composition to achieve lower-degree polynomials, improving efficiency without deteriorating accuracy.
- **The SiLU$(\cdot)$ Block:** The SiLU$(\cdot)$ activation function is an essential building block in Llama architectures. Frameworks use different polynomial approximation algorithms to evaluate it homomorphically. For instance, CACHEMIR uses composite polynomials while SplitPrompt relies on standard Chebyshev approximation techniques to approximate this function.
- **The $\arg\max(\cdot)$ Block:** This block is only used in token generation tasks. A core advancement in optimizing this operation is due to NEXUS, which improves upon the baseline algorithm in Phoenix [62] by reducing the computational complexity from $\mathcal{O}(\mathcal{V})$ to $\mathcal{O}(\log \mathcal{V})$. NEXUS's $\arg\max$ is used in other generative frame-

Recommended Selection Criteria for Linear Blocks

1) **Versatility**: the framework must efficiently handle large matrix multiplications: CCMM and CPMM. The layout must match the target task, e.g., L-ROW keeps each token within its own set of ciphertexts facilitating smooth token-level operations.
2) **Rotation and Depth Optimization**: minimize the total number of homomorphic rotations and the number of distinct rotation keys the server keeps per client, as these keys are large. Additionally, the multiplicative depth must be kept to a minimum.
3) **Ciphertext Count**: Bootstrapping cost is linear in ciphertext count, minimizing the total ciphertext count reduces the overhead of bootstrapping operations.

4. Implementation technique used to accelerate inference by caching previously computed Key and Value tensors to avoid redundant computation.

works such as CEP, MOAI, and EUSTON.

[Class N-MOD] Model-Modifying Frameworks. Frameworks in this class argue that approximating off-the-shelf non-linear blocks with polynomials is not efficient. Instead, they co-design the model architecture to be FHE-friendly. Hence, they may require retraining the model to recover the representational power lost when substituting the non-linear blocks. Frameworks primarily modify the model to replace the Softmax and LayerNorm blocks. These include:

- **Native Polynomial Attention:** Polyformer [63] uses a scaled-σ attention mechanism and *Range Minimization Training* to replace Softmax entirely. This results in a 100% polynomial transformer by substituting non-linear blocks with pure polynomials.
- **Gaussian Kernel Attention:** EFLA and STIP use Gaussian Kernels (GKs) [64] to replace Softmax. GKs eliminate the need for inversion or the $\max(\cdot)$ function. Moreover, the evaluation of $\exp(\cdot)$ is only required over a negative domain, making the numerical approximation more stable.
- **Maximum Estimation and Replacement:** some frameworks focus on the $\max(\cdot)$ function in Softmax (2). Instead of computing an exact $\max(\cdot)$ via deep comparison trees, several works use approximate $\max(\cdot)$. Both TRICYCLE and ELLMO use *statistical-max* estimation techniques, where the $\max(\cdot)$ function is approximated using low-cost statistics such as the mean and standard deviation, based on certain assumptions regarding the attention scores distribution. A few frameworks analyze the plaintext inference and derive constant maximum values that are provided to the encrypted model (e.g., MOAI, NEXUS, and ATLAS).
- **Sigmoid Attention:** ENSI replaces Softmax attention with Sigmoid Attention [65] and computes the sigmoid function over the absolute scores without normalization, which can affect the model's representational capacity.
- **Precomputed and Inverse-Free Layer Normalization:** to minimize the costly inverse square root operations in layer normalization, some frameworks either precompute layer normalization or eliminate division. RL and PRITRAN use a static *Precomputed Layer Normalization* derived from statistics of plaintext training on the target dataset. Other frameworks such as STIP and ELLMO employ dynamic *Delayed Layer Normalization*, which exploits the scale-invariance property of LayerNormto eliminate the division in the first LayerNorm of a transformer block, as the propagated σ factor is canceled out by the second LayerNorm. This requires the FFNN's activation function to be scale-invariant (STIP) or *scale-tolerant* (ELLMO).

Definitions. A framework is *model-preserving* if it preserves the model architecture, and evaluates it up to a bounded numerical error that arises from the approximate nature of CKKS, not the adaptation of the model itself. A framework is *model-modifying* if it requires retraining, fine-tuning, distillation, or architectural substitution of any non-linear block resulting in a different model whose accuracy must be re-established.

Recommended Selection Criteria for Non-Linear Blocks

- **Softmax Evaluation:** use input interval reduction and model-preserving approximation algorithms such as normalize-and-square, which is successfully used in SplitPrompt and THOR. This is the only path we are aware of that scales to Llama-3-8 B without costly retraining.
- **Activations and Normalization:** for LayerNorm, GELU, $\tanh(\cdot)$, and SiLU, frameworks should rely on model-preserving polynomial approximation techniques. This is possible and accurate at Billion-scale models.
- $\arg\max$ **Optimization:** for efficient maximum index retrieval, we recommend integrating low-cost circuits such as the optimized $\arg\max$ in NEXUS.

3.3. System-Level Systematization

At the system level, we focus on the operational design choices that enable the physical execution of encrypted LLM workloads. To overcome the fundamental bottlenecks of private LLM inference with CKKS, such as noise growth and enormous compute and memory footprints, SoA frameworks systematize this workload across three dimensions. First, bootstrapping management is employed to schedule or fuse BT operations. Second, specialized hardware and compilers are leveraged to accelerate CKKS operations and encrypted tensor arithmetic. Finally, to reduce the computational requirements of full encrypted LLM inference, some frameworks resort to hybrid public-private inference, which divides model execution between expensive secure ciphertext operations and highly efficient cleartext operations.

3.3.1. Bootstrapping Management. Bootstrapping is the most computationally expensive operation in CKKS, often accounting for 50% to 70% of the total end-to-end private inference latency [13], [14], [16], [20], [54], [55]. As a result, a critical system-level design choice is when and how to refresh ciphertexts. We identify two main optimization trends among the surveyed frameworks: 1) solutions that strive to optimize bootstrapping placement within the workflow, and 2) solutions that modify bootstrapping internals to fuse specific model operations.

Bootstrapping Placement. The vast majority of surveyed frameworks rely on the static, manual placement of bootstrapping. The general rule of thumb is to invoke bootstrapping only once the multiplicative budget is exhausted. However, some frameworks opt for more sophisticated placement strategies:

- **Graph-based Placement:** CACHEMIR extends the shortest-path algorithm, which was proposed originally in *Orion* [66], by incorporating non-linear blocks

alongside linear ones to optimize private LLM inference. It applies several graph algorithms to control the massive search space of private LLM inference.
- **Ciphertext-Count Driven Placement:** ENSI observes that RMSNorm compresses the number of ciphertexts from d to 1, which reduces the frequency of bootstrapping operations. Similarly, NEXUS positions bootstrapping after dimension-reduction operations, such as matrix multiplication, rather than following high-dimensional GELU evaluations.
- **Depth-Minimization Placement:** MOAI places bootstrapping inside Softmax immediately before the division step, which reduces the required levels from 20 down to 10. This lets the preceding matrix multiplications execute at lower ciphertext levels, where the smaller modulus makes homomorphic multiplication significantly faster.

Operation Fusion. To reduce the impact of bootstrapping latency, SecretVoice fuses some linear operations directly into the process itself. This is achieved in the Slots To Coefficients (StC)-first CKKS bootstrapping variant, whose pre-computed constant linear transforms can absorb preceding linear blocks, folding their computational overhead into bootstrapping. This trade-off has two practical implications. First, the memory footprint grows because additional large fused StC matrices must be defined and cached. Second, non-standard fusions may be incompatible with ongoing optimizations in FHE compilers and hardware accelerators, generally optimized for standard bootstrapping.

3.3.2. Hardware and Compilers. The surveyed frameworks implement their solutions across different hardware platforms, categorized as follows:

Multi-Core CPU-Accelerated CKKS. CPU implementations are crucial for validating algorithmic efficiency and maintaining compatibility with most public CKKS libraries. Additionally, they are not subject to the strict memory constraints often encountered in specialized CKKS hardware accelerators. Frameworks such as ARION, TRICYCLE, PRITRAN, and RL exclusively deploy multi-threaded implementations designed to distribute parallelizable private LLM tasks across the available cores of CPU systems. These frameworks use the CKKS implementation provided by the OpenFHE [67] and Lattigo [68] libraries.

GPU-Accelerated CKKS. More practical solutions require offloading basic CKKS operations to hardware accelerators such as GPUs. CEP provides a notable system-level contribution by releasing the first open-source, GPU-accelerated backend for CKKS bootstrapping in OpenFHE, providing nearly 150× speedup in private GPT-2 inference on an NVIDIA A100 GPU compared with a CPU baseline. ELLMO employs the FIDESlib library [23], which is a server-side GPU-accelerated CKKS implementation that is compatible with OpenFHE ciphertexts. Several frameworks, including EUSTON, CACHEMIR, and ENSI, use a custom extension of the Phantom GPU library [69], whereas THOR is implemented using the closed-source, publicly available Liberate binary [70] (currently deprecated).

Multi-GPU Clusters and Compiler Orchestration. According to CERIUM [18], the encrypted-inference memory footprint of Llama-3-8B exceeds single-GPU memory by several orders of magnitude (see §4.3.1). CERIUM introduces many system-level optimizations, including the first end-to-end multi-GPU CKKS compiler, horizontal and vertical kernel fusions, multi-GPU communication topologies, and compressed sparse plaintext encoding to optimize the computational requirements of private LLM workloads. Notably, CERIUM is not open-source at the time of writing; however, the authors indicate a future release. FHE compilers such as HEIR [71] and Orion [66] provide similar lowering capabilities, but to our knowledge, have not published end-to-end encrypted LLM inference results.

3.3.3. Hybrid Public-Private Inference. To reduce the enormous computational requirements inherent to the encrypted LLM inference workload, some frameworks adopt different use cases. An increasingly popular use case is to partition the workload into public (plaintext) and private (ciphertext) domains. Existing formulations of this use case can be categorized into two approaches: model partitioning and prompt partitioning.

Model Partitioning. Some frameworks restrict the homomorphic evaluation to only sensitive components of the LLM model. For instance, RZW [72] splits the model into a large, cleartext base model deployed at the client and encrypted smaller Low-Rank Adaptation (LoRA) modules hosted on the server. Only the fine-tuned LoRA matrices, which contain the specialized proprietary knowledge, reside on the server. During inference, the client processes the data locally until it reaches the fine-tuned layers. At this point, the client encrypts the intermediate features and sends them to the server for homomorphic evaluation.

Prompt Partitioning. A more recent approach adopts a specialized use case, where the prompt is partitioned based on data sensitivity. A substantial fraction of the prompt is treated as public tokens (ptok) and can be processed in the clear during the *prefill* phase of autoregressive generative LLM tasks. However, a smaller segment is treated as sensitive and must be processed under encryption (etok). Following this approach, SplitPrompt scales to process up to 4096 input tokens (with only the final 128 encrypted). Running the Llama-2-7B model on a cluster of eight NVIDIA RTX-4090 GPUs, SplitPrompt achieves an 85-second summarization latency and a 33-second per-token generation latency.

3.4. Open-Source Reference Framework

Our systematization identifies a practical limitation: nearly every SoA framework is a paper-tailored prototype. To address this gap, we release POLARIS (*POLynomial Approximation-based Reference for Inference Systematization*), an open-source reference implementation that directly applies the model- and system-level design principles established in §3.2 and §3.3. It supports evaluating encrypted

Recommended Selection Criteria for System Design

- **Bootstrapping Management:** we recommend explicit bootstrapping placement and scheduling to minimize bootstrapping frequency. Fusing linear layers into bootstrapping should be balanced against the memory, compiler, and hardware accelerators trade-offs noted in §3.3.1.
- **Execution Platform:** we suggest starting with a CPU platform during prototyping to ensure reproducibility, and avoiding hardware-accelerator memory constraints. Once the prototype is stable, migrate to hardware accelerators, especially when high speed becomes the priority.
- **Hybrid Public-Private Inference:** we recommend adopting more prevalent use cases where the entire model is hosted by the server and the entire prompt is encrypted. Hybrid architectures should be reserved for scenarios where the threat model and functional requirements allow it.

LLM inference for the entire BERT family: BERT-Tiny, BERT-Mini, BERT-Small, BERT-Medium, and BERT-Base.

Despite its strong performance, POLARIS is *not* intended to compete with the frameworks surveyed on latency. Its primary goal is to provide an open-source, extensible research tool that separates algorithmic research from low-level systems engineering to facilitate reproducible benchmarks across diverse hardware.

3.4.1. Overview. In line with the taxonomy in (§3.2), POLARIS commits to **model-preserving** approximations only (**Class N-PRE**): no fine-tuning, no architectural substitutions, and no hybrid public/private execution split. Our approach is driven by the principle that architectural modifications must be validated within the standard machine learning domain before being applied to the FHE context. This offers two primary advantages: it ensures that any observable accuracy degradation downstream is solely attributable to the CKKS approximation rather than architectural modifications, and it eliminates the need for costly retraining.

POLARIS implements model-preserving, end-to-end private inference for sequence classification on the entire BERT family, from BERT-Tiny to BERT-Base, using the Stanford Sentiment Treebank (SST)-2 dataset[5]. It adopts a non-interactive, full server outsourcing setup where the client performs tokenization and embedding in the clear, encrypts the token embeddings, and transmits the ciphertexts to the server, which homomorphically evaluates the encoder layers, pooling, and classification (see Figure 2a in Appendix A).

3.4.2. Linear Blocks. POLARIS extends the Token/Row Packing (**Class L-ROW**) layout used in RL [11], whose original implementation constrains the model dimension (d) to 128. This constraint is practical for 128×128 projections of BERT-Tiny, but fails with larger BERT variants. POLARIS provides *tiled* variants of the VecMatER and VecMatRC routines of RL, wherein a matrix is partitioned into a grid of 128×128 sub-matrices, each encoded in its own ciphertext. We adopt this layout as it is simple and easily extendable to support KV-cache, a capability we envision future versions of POLARIS should include. While **Class L-ROW** does not satisfy the slot-utilization criterion we recommend in §3.2.1, we accept this cost deliberately as POLARIS optimizes for extensibility and reproducibility rather than latency.

5. Appendix E lists BERT models dimensions and measurements.

3.4.3. Non-Linear Blocks. For non-linear blocks, POLARIS incorporates model-preserving approximations (**Class N-PRE**) that we found to be the most stable across the surveyed frameworks. Specifically:

- **Softmax:** we use the iterative normalize-and-square algorithm [21], which is numerically stable and adaptable to various attention score ranges.
- $\frac{1}{\sqrt{\cdot}}$**,** $\mathbf{GELU}(\cdot)$**,** $\tanh(\cdot)$**:** we use Chebyshev polynomial approximations whose degrees and intervals are derived from the plaintext reference implementation.

3.4.4. Concrete Parameters. As a concrete example, we present the instantiation of POLARIS using BERT-Mini on the SST-2 dataset. Table 3 (Appendix D) summarizes the model, CKKS, and non-linear function approximation parameters, along with system and application details. Larger models require a different set of parameters, we refer the reader to POLARIS codebase for complete instantiation of all models.

3.4.5. Software and Hardware Backends. We implemented POLARIS using the OpenFHE and FIDESlib libraries. The framework supports two execution backends:

- **CPU-Only Backend (OpenFHE):** a multi-threaded reference implementation built on OpenFHE [67].
- **GPU Backend (OpenFHE + FIDESlib):** an accelerated backend using a single NVIDIA GPU. It reuses the circuit in the CPU reference implementation but offloads CKKS operations to FIDESlib [23].

The CPU backend can be used for verification and running large models that cannot fit in GPU memory. However, CPU latencies are 2 orders of magnitude slower, so we only report the GPU latencies.

4. Evaluation of Framework Implementations

In this section, we evaluate SoA private LLM inference frameworks. We identify current benchmarking challenges, introduce the *Private LLM Card System* (PLCS) reporting scheme, and provide a comprehensive literature comparison.

4.1. Challenges in Standardized Benchmarking

The rapid development of CKKS-based private LLM inference has resulted in a highly fragmented evaluation landscape. As a result, it has become more challenging to compare SoA frameworks fairly due to the following factors:

1) **Model Scale Disparity:** evaluated models range from 4.4 M (BERT-Tiny) to 8 B-parameter models (Llama).
2) **Computation Roles:** most frameworks let the server evaluate the entire model, but some adopt a more relaxed evaluation to bypass FHE bottlenecks.
3) **Cryptographic Parameters Disparity:** in CKKS, end-to-end latency, accuracy, and security are driven by the cryptographic parameters: (N, Q, Δ, L). Frameworks heavily tailor these parameters to fit their architectural optimizations.
4) **Software and Hardware Disparity:** hardware spans consumer multi-core CPUs, single GPU, and multi-GPU clusters, while software spans CPU libraries (OpenFHE [67], Microsoft SEAL [73], Lattigo [68], HELayers [74]), and GPU libraries (Phantom [75], Liberate [70], FIDESlib [23], HEaaN [5]).
5) **Downstream Task Divergence:** most frameworks target sequence classification using encoder-only models, whereas generative models need autoregressive decoding with KV cache, token sampling logic such as $\arg\max$ and multi-token prefill phase.
6) **Runtime Performance Metrics:** classification-focused frameworks report amortized or unamortized latency. Some frameworks (e.g., ELLMO) do not report the sequence length in their benchmarks. Autoregressive generation tasks report time to first token (or prefill time) for prompt processing (CERIUM and CACHEMIR), generation Time per Output Token (TPOT) for subsequent decoding steps (SplitPrompt), and occasionally the forward pass runtime at a specific sequence position (CEP).
7) **Inconsistency in Model Utility Metrics:** due to approximation errors inherent in CKKS evaluations, encrypted models often deviate from their non-encrypted counterparts. Some frameworks measure utility through downstream task performance, reporting classification accuracy or F1-scores (e.g., ARION, EFLA, ENSI, RL, NEXUS, MOAI, EUSTON, Powerformer, STIP, ELLMO, THOR). For frameworks that focus on autoregressive generation tasks, utility is measured by the perplexity[6] metric (e.g., CEP, Polyformer, CACHEMIR, SplitPrompt). A few frameworks do not consider any model utility metrics (e.g., SecretVoice, PRITRAN).
8) **Reproducibility Barrier:** closed-source implementations and hardware scarcity create a major verification challenge. Moreover, some frameworks provide only building blocks of LLM components, making end-to-end testing impossible.

4.2. Recommended Evaluation Scheme

As seen in the preceding section, the evaluation landscape for CKKS-based private LLM inference frameworks is highly disjointed. Direct performance comparisons in current literature are not accurate, as frameworks omit critical details such as hardware specifications, custom cryptographic libraries and parameters, and model modifications. To systematize the evaluation scheme of private LLM inference with CKKS, we propose a four-dimensional reporting scheme termed the PLCS.

6. A measure of the predictive accuracy of an LLM on a test sample.

4.2.1. Model Card. To address architectural modifications, researchers must explicitly define the target model. This card must report the exact model architecture, parameter count, and token sequence length (which is sometimes unreported in current literature). Additionally, it must state whether the framework requires model retraining due to knowledge distillation, polynomial substitutions, or any other reason.

4.2.2. CKKS Cryptographic Parameters Card. Reported runtime performance is irrelevant if the framework fails to meet robust security standards. This card must report the CKKS cryptographic parameters, alongside the chosen protocol paradigm and bootstrapping strategy.

4.2.3. System Deployment Card. Since runtime performance is sensitive to hardware and software backends, this card standardizes the system environment. It must document the exact hardware system used (e.g., commodity CPUs, single GPUs, and massive multi-GPU clusters). Additionally, it must specify the underlying FHE library (e.g., Microsoft SEAL [73], OpenFHE [67], Lattigo [68], Phantom [75]). Finally, the card should define the client-server computational boundary, explicitly noting if any plaintext processing (beyond initial tokenization and embedding) is performed by the client.

4.2.4. Downstream Application Card. Frameworks should describe the target evaluation task (sequence classification vs. autoregressive generation) and any task-specific assumptions related to the inference phase (e.g., only part of the prompt is encrypted). Additionally, this card standardizes both the *runtime performance* and the *model utility* of the framework.

Runtime Performance Metrics. Based on our analysis of the surveyed frameworks, we propose the following standardized runtime metrics:

- **Unamortized (End-to-End) Sequence Latency:** wall-clock time of a single sequence, excluding the setup time.
- **Amortized Per-Sequence/Token Latency:** average per-sequence/token latency under full batching.
- **Time to First Token (TTFT) / *Prefill* Time:** time to process the entire input prompt and generate the first token in generative tasks.
- **Time per Output Token (TPOT):** steady-state latency of generating subsequent tokens using KV cache.
- **Forward Pass Runtime at Specific Position / Per Position Time (PPT):** wall-clock of one forward pass at a given sequence position (e.g., the 128th token), excluding the processing overhead of the prompt.

Model Utility Metrics. Finally, to measure the quality of encrypted models, frameworks should report utility across a combination of the following metrics:

- **Downstream Task Performance:** direct comparison of the encrypted model against the plaintext model using benchmark-specific metrics (e.g., classification accuracy or F1-score).
- **Generative Modeling Quality:** for autoregressive generative models, degradation in word or byte perplexity can be used.

For a complete illustration of PLCS's four cards, instantiated for SplitPrompt, see Figure 4 in Appendix D.

4.3. Comparison of State of the Art Frameworks

Guided by the PLCS evaluation framework defined in §4.2, we present a comparison of 21 SoA CKKS-based private LLM inference frameworks in Tables 1 and 2. Rows are first grouped by *non-linear model preservation* (**Class N-PRE** vs. **Class N-MOD**) and then by model scale. We keep the original reported metric rather than deriving a single uniform metric due to the challenges identified in §4.1. We report the accuracy *loss* of each encrypted model relative to its own plaintext counterpart, categorized as Mild, Moderate, and Major. We exclude RZW, as its server-side LoRA adaptation deviates significantly from the computation model adopted by other frameworks.

4.3.1. Sequence Classification. Table 1 compares sequence classification tasks. The first notable observation is the imbalance in non-linear block classes: only THOR (BERT-Base) and POLARIS (BERT-Tiny to BERT-Base) occupy **Class N-PRE** (Model-Preserving), and the large-scale band is empty, a clear gap we identify for future work. On the other hand, **Class N-MOD** (Model-Modifying) is widely adopted; 12 distinct frameworks out of 14 follow this approach to reduce the complexity of the FHE circuit. The employed techniques can result in an order-of-magnitude improvement. For instance, EFLA observes a 6.24× improvement by using a Gaussian kernel instead of Softmax, while Powerformer achieves a 90% reduction in multiplicative depth (3 vs. 30) when using *Power Max*. Note that we consider methods involving Softmax replacement [20], [52], [61], assumed Softmax's maximum value [14], [15], [19], [50], statistical maximum value estimation [13], [54], [58], or non-standard LayerNorm [11], [12] as model-modifying. PRITRAN states that it follows RL's LayerNorm treatment without explicitly committing to a model-preservation level; hence we mark it as Class N-MOD.

We also observe a shift from compact models with a few million parameters (e.g., BERT-Tiny) towards much larger models with hundreds of millions to several billion parameters (e.g., BERT-Base, GPT-2, and Llama-3), which reflects the trend in unencrypted LLM use. This suggests that CKKS can realistically support more complex tasks. Notably, GPU-accelerated solutions outperform CPU-only implementations. Focusing on BERT-Base, we observe at least an order of magnitude improvement when comparing NEXUS vs. ARION. While it is unfair to attribute such speedups to the GPU alone when comparing frameworks with different implementations and hardware/software environments, several more uniform studies conclude similar results [14], [15].

We also note the effect of batching on latency. Focusing on model-modifying GPU-accelerated BERT-Base solutions, amortized (per 32 or 256 sequences) latencies can be an order of magnitude faster than unamortized latencies (over a single sequence): (NEXUS 37 s, EUSTON 46 s, STIP 64 s, MOAI 141 s, vs. Powerformer 345 s). A hypothetical unamortized latency of NEXUS would exceed $37 \times 32 = 1184$ s.

To understand how CKKS-based private LLM inference implementations scale with model size, it is informative to focus on EUSTON's results on a single RTX 6000 GPU: BERT-Base (46 s), GPT-2-1.5B (136 s), and Llama-3-8B (484 s). Notably, a $\approx 70\times$ model-size range maps to an $\approx 10\times$ latency range, demonstrating sub-linear scaling. In terms of memory footprint, CERIUM reports linear scaling between model size and FHE memory (≈14 KB per parameter, ≈7000× the FP16 plaintext footprint). This means scaling pure FHE-based solutions to larger models cannot rely solely on hardware upgrades (encrypted large models will not fit into any available GPU memory or CPU memory, a constraint that may require streaming solutions such as memory-mapped files [76]).

4.3.2. Autoregressive Generation. Table 2 focuses on frameworks tackling Next Word Prediction (NWP) or token generation. Unlike sequence classification frameworks, the model-preserving class includes 3 frameworks: CEP in the Small/Medium band, and CACHEMIR and CERIUM in the Large band. CACHEMIR runs Llama-3-8B end-to-end on a single A100 GPU at 97 s TPOT with 512 context tokens, and CERIUM reports 134 s prefill latency on Llama-3-8B using 8× B200 GPUs. To our knowledge, these are the first reported results of large-scale, model-preserving, non-interactive CKKS-based private LLM inference, demonstrating that deep and complex computations are achievable with CKKS. We note that CERIUM is a system- and compiler-level solution, unlike other cryptographic solutions such as CACHEMIR and CEP. Its main contribution is an end-to-end multi-GPU CKKS compiler. Similarly, CACHEMIR is not a self-contained implementation, as it reuses THOR's non-linear blocks and layers its bootstrapping scheduling algorithm on top.

We assign SplitPrompt to a dedicated Class N-PRE† sub-band because it achieves model preservation under a relaxed threat model, this places it between the model-preserving and model-modifying classes. It reports the lowest TPOT latency at 33 s and 34 s on Llama-2-7B and Llama-3-8B, respectively, using 8×RTX 4090 GPUs. The significant performance gain stems from encrypting only the last 128 tokens of a 4096-token sequence, which substantially reduces prefill latency. While the encrypted portion is evaluated under CKKS in a model-preserving manner, this hybrid public-private split deviates significantly from the client-server model adopted by the majority of frameworks and provides weaker privacy guarantees than fully

TABLE 1. COMPARISON OF CKKS-BASED PRIVATE TRANSFORMER INFERENCE FOR **SEQUENCE CLASSIFICATION**. ROWS ARE GROUPED BY NON-LINEAR MODEL PRESERVATION (**CLASS N-PRE** VS. **CLASS N-MOD**; SEC. 3), THEN BY MODEL SCALE.

Framework	Lin.	Model	Hardware	Software	N	L	Task (Sequences) (Tokens)	Performance Metric	Utility Metric	Latency	Acc. Loss [ll]	OS
Non-linear: Model-Preserving (Class N-PRE)												
Small ($\lesssim 50\,M$ parameters)												
POLARIS (Ours)	L-ROW	BERT-Tiny	A100 GPU	OpenFHE / FIDESlib	2^{16}	26	SST-2 (1, 24)	Unamort. latency	Downstream acc./F1	18 s	○	✓
POLARIS (Ours)	L-ROW	BERT-Mini	A100 GPU	OpenFHE / FIDESlib	2^{16}	26	SST-2 (1, 24)	Unamort. latency	Downstream acc./F1	83 s	○	✓
POLARIS (Ours)	L-ROW	BERT-Small	A100 GPU	OpenFHE / FIDESlib	2^{16}	26	SST-2 (1, 4)	Unamort. latency	Downstream acc./F1	49 s	○	✓
POLARIS (Ours)	L-ROW	BERT-Medium	A100 GPU	OpenFHE / FIDESlib	2^{16}	27	SST-2 (1, 4)	Unamort. latency	Downstream acc./F1	92 s	○	✓
Medium ($\approx 100\,M$ parameters)												
THOR [16]	L-COL	BERT-Base	A100 GPU	Liberate	2^{16}	13^c	GLUE (1, 128)	Unamort. latency	Downstream acc./F1	602 s	●	✓
POLARIS (Ours)	L-ROW	BERT-Base	A100 GPU	OpenFHE / FIDESlib	2^{17}	32	SST-2 (1, 4)	Unamort. latency	Downstream acc./F1	372 s[x]	○	✓
Non-linear: Model-Modifying (Class N-MOD)												
Small ($\lesssim 50\,M$ parameters)												
ARION [13]	L-COL	BERT-Tiny	64-Core CPU	Lattigo	2^{16}	14^c	GLUE (256, 128)	Amort. per-seq.	Downstream acc./F1	9.3 s	○	✓
ELLMo [58]	L-ALG	BERT-Tiny	H200 GPU	OpenFHE / FIDESlib	2^{16}	26	GLUE (–, –)	Unamort. latency	Downstream acc./F1	8 s	●	✓*
RL [11]	L-ROW	BERT-Tiny	M1 Pro CPU	OpenFHE	2^{16}	24	SST-2 (1, 32)	Unamort. latency	Downstream acc.	≈672 s	▲	✓
TRICYCLE [54]	L-CYC	BERT-Tiny	128-core CPU	OpenFHE	2^{16}	29	SST-2 (1, 27)	Unamort. latency	Downstream acc.	322 s	○	✘
EFLA [61]	L-ROW	BERT-Like	RTX 4090 GPU	HEaaN	2^{16}	9^c	GLUE (1, 128)	Latency per layer	Downstream acc./F1	27 s	▲	✓
PRITRAN [12]	L-ROW	BERT-Tiny	64-Core CPU	OpenFHE	2^{16}	24	MNLI (1, 24)	Unamort. latency	–	398 s	–	✘
PRITRAN [12]	L-ROW	BERT-Mini	64-Core CPU	OpenFHE	2^{16}	24	MNLI (1, 32)	Unamort. latency	–	≈2000 s	–	✘
Medium ($\approx 100\,M$ parameters)												
ARION [13]	L-COL	BERT-Base	64-Core CPU	Lattigo	2^{16}	14^c	GLUE (256, 128)	Amort. per-seq.	Downstream acc./F1	225 s	○	✓
ATLAS [50]	L-ROW	BERT-Base	RTX 4090 GPU	Phantom	2^{16}	14^c	GLUE (1, 128)	Unamort. latency	Downstream acc.	845 s	○	✓
EUSTON [19]	L-ALG	BERT-Base	RTX 6000 GPU	SEAL / Phantom	2^{16}	35	SST-2 (32, 128)	Amort. per-seq.	Downstream acc./F1	46 s	○	✓
Powerformer [52]	L-COL	BERT-Base	A100 GPU	Liberate	2^{16}	11^c	GLUE (1, 128)	Unamort. latency	Downstream acc./F1	345 s	▲	✓
MOAI [14]	L-COL	BERT-Base	H200 GPU	SEAL / Phantom	2^{16}	34	GLUE (256, 128)	Amort. per-seq.	Downstream acc./F1	141 s	○	✓
NEXUS [15]	L-CYC	BERT-Base	4×A100 GPU	SEAL / Phantom	2^{16}	35	GLUE (32, 128)	Amort. per-seq.	Downstream acc./F1	37 s	○	✓
STIP [20]	L-CYC	BERT-Base	A100 GPU	SEAL / Phantom	2^{16}	14^c	GLUE (32, 128)	Amort. per-seq.	Downstream acc./F1	64 s	▲	✓*
Large ($\geq 1\,B$ parameters)												
EUSTON [19]	L-ALG	GPT-2-1.5B	RTX 6000 GPU	SEAL / Phantom	2^{16}	35	SST-2 (32, 128)	Amort. per-seq.	–	136 s	–	✓
EUSTON [19]	L-ALG	Llama-3-8B	RTX 6000 GPU	SEAL / Phantom	2^{16}	35	SST-2 (32, 128)	Amort. per-seq.	Downstream acc./F1	484 s	●	✓
STIP [20]	L-CYC	GPT-2-1.5B	A100 GPU	SEAL / Phantom	2^{16}	14^c	GLUE (32, 128)	Amort. per-seq.	Downstream acc./F1	142 s	○	✓*
STIP [20]	L-CYC	Llama-3-8B	A100 GPU	SEAL / Phantom	2^{16}	14^c	GLUE (32, 128)	Amort. per-seq.	Downstream acc./F1	407 s	▲	✓*

Lin.: Linear packing layout classes: **L-ROW** token/row packing; **L-COL** column/diagonal packing; **L-CYC** complex/cyclic encoding; **L-ALG** algorithmic shortcuts (§3.2.1). Band headers (**Class N-PRE** vs. **Class N-MOD**) refer to non-linear model preserving or modifying classes (§3.2.2). **OS:** Open-source. [c] Compute levels after Bootstrapping. * Partial / artifact-level open release. [ll] **Accuracy loss**: encrypted-vs-plaintext utility gap. ○ $\leq 0.5\%$; ● 0.5–1.5%; ▲ $> 1.5\%$ or requires retraining. [x] Extrapolated. The problem size exceeded A100 memory, so we measured at $N = 2^{16}$ (186 s) and doubled it to account for the 2× bootstrapping latency scaling between these ring sizes. "–" = not reported.

encrypted solutions. Notably, CACHEMIR, SplitPrompt, and SecretVoice support KV cache pipelines, which are crucial for efficient autoregressive NWP.

Among model-modifying implementations on large models, MOAI and NEXUS target Llama-3-8B end-to-end under the standard non-interactive threat model, while ENSI explores a smaller BitNet b1.58 variant. The performance disparity between NEXUS and MOAI (52 s vs. 225 s) on an A100 GPU vs. 4× A100 GPUs can be attributed to hardware scaling. Lastly, we note that the cryptographic parameters of ENSI are questionable; specifically, a configuration of $L = 48$ with only $N = 2^{15}$ fails to provide ≈128-bit security level [77].

4.3.3. Accuracy Loss. The accuracy loss columns in Tables 1 and 2 show a clear advantage for Class N-PRE over Class N-MOD. The former never reaches Major (▲) loss: its worst entry is Moderate (●) at THOR and CEP, and every large model exhibits Mild (○) loss. In Class N-MOD, every retraining- or distillation-based framework (such as EFLA, Powerformer, STIP, Polyformer, and ENSI) shows Major loss. We note that the architectural techniques employed in Class N-MOD are primarily driven by inference latency rather than accuracy. Hence, some frameworks suggest using retraining to improve accuracy.

4.3.4. Positioning POLARIS. POLARIS fills a gap in Table 1 as the first *open-source*, model-preserving, GPU-accelerated encrypted classifier for the entire BERT family. Within **Class N-PRE**, it is the only "small-scale" entry and reports unamortized latencies of 18 s (BERT-Tiny) and 83 s (BERT-Mini) for 24-token sequences, and 49 s (BERT-Small), 92 s (BERT-Medium), and 372 s (BERT-Base) for four-token sequences on one A100. Compared with same-scale **Class N-MOD** BERT-Tiny implementations on GPU/CPU (e.g., ELLMO at 8 s, ARION at 9.3 s amortized, and TRICYCLE at 322 s), POLARIS achieves comparable or enhanced unamortized performance while preserving the model architecture.

TABLE 2. COMPARISON OF CKKS-BASED PRIVATE TRANSFORMER INFERENCE FOR **AUTOREGRESSIVE TOKEN GENERATION**. ALL FRAMEWORKS USE NON-INTERACTIVE FHE EXCEPT THOSE MARKED WITH †. MODEL-PRESERVING FRAMEWORKS OPERATING UNDER A RELAXED THREAT MODEL (HYBRID PUBLIC-PRIVATE COMPUTATION) ARE LISTED IN A DEDICATED SUB-BAND.

Framework	Lin.	Model	Hardware	Software	N	L	Task (Sequences) (Tokens)	Performance Metric	Utility Metric	Latency	Acc. Loss [ϑ]	OS
Non-linear: Model-Preserving (Class N-PRE)												
Small / Medium (≤ 100 M parameters)												
CEP [17]	L-ROW	GPT-2	A100 GPU	Custom	2^{17}	≈ 40	Decode (1, 128)	PPT	PPL	≈100 s	●	✘
Large (≥ 1 B parameters)												
CACHEMIR [55]	L-CYC	Llama-3-8B	A100 GPU	Lattigo / Phantom	2^{16}	13[c]	Decode (1, 64–512)*	TPOT	PPL	97 s	○	✘
CERIUM [18]	–	Llama-3-8B	8×B200 GPU	Custom	2^{16}	18[c]	Decode (1, 128)	Prefill Time	PPL	134 s	○	✘[$]
Non-linear: Model-Preserving: Hybrid Public-Private Split (Class N-PRE†)												
Large (≥ 1 B parameters)												
SplitPrompt† [59]	L-ALG	Llama-2-7B	8×RTX 4090 GPU	HEaaN2	2^{16}	–	Prompt part. (4096 / $\overline{128}$)	TPOT	PPL	33 s	○	✘
SplitPrompt† [59]	L-ALG	Llama-3-8B	8×RTX 4090 GPU	HEaaN2	2^{16}	–	Prompt part. (4096 / $\overline{128}$)	TPOT	PPL	34 s	○	✘
Non-linear: Model-Modifying (Class N-MOD)												
Small / Medium (≤ 100 M parameters)												
Polyformer [63]	–	BERT-6L	A100 GPU	HELayers	2^{15}	12	NWP (1, 128)	Unamort. latency	PPL	305 s	▲	✘
SecretVoice [49]	L-ROW	Transf. Enc–Dec	1-Core CPU	SEAL	2^{15}	19	Encoder step (1, 32)	Unamort. latency	–	1488 s[e]	–	✘
Large (≥ 1 B parameters)												
ENSI‡ [56]	L-ALG	Llama-3-700M	A100 GPU	OpenFHE / Phantom	2^{15}	48	Decode (32, 2048)	TTFT	Acc.	958 s	▲	✓
MOAI [14]	L-COL	Llama-3-8B	A100 GPU	SEAL / Phantom	2^{16}	34	Decode (1, 8)	Unamort. latency	–	225 s	–	✓
NEXUS [15]	L-CYC	Llama-3-8B	4×A100 GPU	SEAL / Phantom	2^{16}	35	GLUE (32, 8)	TPOT	PPL	52 s	▲	✓

Lin.: Linear packing layout classes: **L-ROW** token/row packing; **L-COL** column/diagonal packing; **L-CYC** complex/cyclic encoding; **L-ALG** algorithmic shortcuts (§3.2.1). Band headers (**Class N-PRE** vs. **Class N-MOD**) refer to non-linear model preserving or modifying classes (§3.2.2). The **Class N-PRE**† sub-band denotes model-preserving frameworks but computation is split between encrypted and plaintext domains. PPL: Perplexity. **OS:** Open-source. † Hybrid public–private split instead of fully non-interactive FHE on the full model. [e] Single encoder block. [c] Compute levels after Bootstrapping. ‡ BitNet b1.58 variant. Transf. Enc–Dec Original Transformer Model (6 encoders and 6 decoders). * Partial / artifact-level open release. ⋆ Initial input of 64 tokens, and a context window of 512 tokens. (4096 / $\overline{128}$) 4096 tokens with only last 128 tokens encrypted. [$] Will be open-source in future. [ϑ] **Accuracy loss**: encrypted-vs-plaintext utility gap. ○ ≤ 0.5%; ● 0.5–1.5%; ▲ > 1.5% or requires retraining. "–" = not reported.

4.4. Cross-Framework Insights

We summarize here the main insights that can be drawn from the previous comparison of SoA encrypted LLM inference frameworks.

Cross-Framework Insights

- **I1 Model preservation is the exception.** Only 5 out of 21 frameworks are model-preserving under full encryption. The majority of existing research prioritizes runtime performance over accuracy.
- **I2 Model preservation protects accuracy.** Class N-PRE implementations exhibit Mild to Moderate accuracy loss. In contrast, a significant portion of Class N-MOD implementations show Major loss. Modifying the model to fit FHE can improve efficiency at the cost of higher accuracy loss.
- **I3 Bootstrapping is the main bottleneck.** Bootstrapping takes 50% to 70% of latency across many frameworks. The number of bootstrapping in an encrypted LLM inference workload can predict the end-to-end latency.
- **I4 The memory wall limits scalability.** Runtime scales linearly: a 2.6× increase in model size costs 4.6× latency (POLARIS BERT-Tiny 18 s to BERT-Mini 83 s). Similarly, encrypted memory grows from 1.5 TB (BERT-Base 110 M parameters) to 112 TB (Llama-3-8B) in CERIUM. Although the scaling is linear, the memory footprint exceeds the capacity of current devices.

5. Discussion and Open Challenges

The results presented in Sections 3 and 4 show that only 5 out of 21 private LLM frameworks are model-preserving *under the standard threat model in which the entire prompt is encrypted*. We do not count SplitPrompt (Class N-PRE†), which encrypts a small part of the prompt. We also observe that the model-preserving solutions achieve acceptable practical accuracy (i.e., $\leq 0.5\%$) even for models with 8 B parameters. This gives us confidence in the expectation that models with more than 100 B parameters can also be supported with sufficient accuracy. In summary, we give a conditional positive answer to **RQ2**, which can become unconditional once models beyond 100 B parameters are implemented with CKKS thanks to efficiency advances.

FHE overhead over cleartext frameworks. Next, we turn our attention to **RQ3**, basing our analysis on the model-preserving FHE implementations for the largest models, i.e., BERT-Base and Llama-3-8B [16], [18], [55]. We examine the works in chronological order, demonstrating the runtime improvements over a span of 15 months.

The THOR study [16] reports the runtime of 602 s on a system with an A100 GPU for the sentiment analysis of a sequence with 128 tokens and batch size of 1. The cleartext FP16 benchmarks [78], [79] on an A100 GPU for the same setup suggest that the optimized cleartext runtime is in the range from 1.0 to 3.2 milliseconds, implying that the THOR solution is 5-6 orders of magnitude slower than

the optimized FP16 GPU baseline production LLMs use.

The CERIUM study [18] reports benchmarking results for BERT-Base and Llama-3-8B models on A100, H100, and B200, supporting multi-GPU setups from 1 to 8 GPUs. Their A100 BERT-Base runtime is 66 s, i.e., roughly one order of magnitude faster than THOR. A cleartext benchmarking study [80] reports the TTFT of 21.29 milliseconds for Llama-2-7B (which is very close in size to Llama-3-8B) on an A100 GPU. The corresponding Llama-3-8B TTFT in [18] is 675 s. The NVIDIA TTFT FP16 benchmark (v1.3.0) for Llama-3.1-8B with 200 input tokens on H100 reports 11.96 milliseconds [81], while the corresponding TTFT for the input sequence of 128 tokens is 698 s [18]. All of the above suggests that the latency results presented in Table 1 of [18] are at the border of 4 and 5 orders of magnitude higher than the cleartext results.

The CACHEMIR study [55] reports the TPOT of ≈97 s for Llama-3-8B on an A100 GPU system. The cleartext TPOT for Llama-2-7B is approximately 11 milliseconds [80]. As Llama-3-8B is roughly 14% larger and TPOT scales linearly with model size, the estimated cleartext runtime should be 12-13 milliseconds. The NVIDIA ITL (≈ TPOT) FP16 benchmark (v1.3.0) for Llama-3.1-8B with 200 input and output tokens on an H100 GPU is 6.52 milliseconds [81]. As A100 has lower bandwidth and number of transistors than H100, the TPOT for A100 cannot be smaller than 6.52 ms. The above analysis implies that CACHEMIR has a slowdown of 4 orders of magnitude w.r.t. cleartext benchmarking results.

Current challenges. As the overhead discussion implies, a major practicality challenge is efficiency. A latency that is four orders of magnitude higher than in the clear reduces the practical applicability of FHE-protected autoregressive generation for human-facing applications. Similar to the challenges for LLM autoregressive generation in the clear, where the prefill phase is typically compute-bound and the decoding phase employing the KV cache is memory-bound, the overhead of FHE computation is also determined by both factors. Moreover, in the case of CKKS, we work with ciphertexts and plaintexts that are larger in size than their cleartext instances (often by an order of magnitude). As the non-linear computations and bootstrapping consume many levels, the expansion of ciphertexts/plaintexts w.r.t. cleartext can lead to a four-orders-of-magnitude increase in memory footprint [18], which increases both bandwidth and compute requirements.

The second major challenge, which will become critical for the production use of private LLM inference once the efficiency challenge is adequately addressed, is the robustness of FHE-based frameworks in supporting various models and datasets. The polynomial approximations for non-linear functions require specifying the input domain interval, and the computational complexity of polynomial evaluation increases with interval size and precision. Many FHE implementations discussed in our paper used the input domain intervals tailored for specific models (and datasets). ATLAS demonstrates this issue experimentally by failing to reproduce NEXUS and THOR accuracy results on different model weights, indicating that these methods are calibrated for the pretrained model used.

Areas for future research. The increased memory footprint of FHE solutions directly depends on the efficiency of CKKS bootstrapping, specifically the number of levels needed, the number of evaluation keys, and, to a lesser degree, the computational complexity of primitive FHE operations. Some recent works presenting cheaper bootstrapping designs that may reduce this efficiency gap [82], [83]. Besides that, improved bootstrapping management, which was already discussed in § 3.3.1, can further reduce the FHE overhead. Finally, hardware acceleration may further close the efficiency gap. Preliminary estimates suggest that custom FHE chips are expected to offer a speedup of four to five orders of magnitude [84], [85], [86], [87].

6. Conclusion

In this work, we presented the first systematization of knowledge for CKKS-based, non-interactive private LLM inference. We conducted a comprehensive comparison of 21 frameworks, focusing on their scale, deployment configurations, model preservation, utility measures, and performance. We also release POLARIS, an open-source proof-of-concept reference framework that combines the most stable model-preserving building blocks identified by our analysis. Our study establishes that model-preserving CKKS-based private LLM inference is feasible and maintains high downstream accuracy for models with up to 8 B parameters (e.g., Llama-3-8B). However, we also identify a significant efficiency bottleneck that restricts its practical use: a runtime gap of four orders of magnitude compared to cleartext inference under comparable system configurations.

Acknowledgments

This research was funded, in part, by the Advanced Research Projects Agency for Health (ARPA-H). The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government.

References

[1] A. J. Thirunavukarasu, D. S. J. Ting, K. Elangovan, L. Gutierrez, T. F. Tan, and D. S. W. Ting, "Large language models in medicine," *Nature medicine*, vol. 29, no. 8, pp. 1930–1940, 2023. 1

[2] I. Cheong, K. Xia, K. J. K. Feng, Q. Z. Chen, and A. X. Zhang, "(a)i am not a lawyer, but...: Engaging legal experts towards responsible llm policies for legal advice," in *Proceedings of the 2024 ACM Conference on Fairness, Accountability, and Transparency*, ser. FAccT '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 2454–2469. [Online]. Available: https://doi.org/10.1145/3630106.3659048 1

[3] S. Mahdavi, P. K. Joshi, L. H. Guativa, U. Singh *et al.*, "Integrating large language models in financial investments and market analysis: A survey," 2025. 1

[4] R. Staab, M. Vero, M. Balunovic, and M. Vechev, "Beyond memorization: Violating privacy via inference with large language models," in *The Twelfth International Conference on Learning Representations*, 2024. [Online]. Available: https://openreview.net/forum?id=kmn0BhQk7p 1

[5] J. H. Cheon, A. Kim, M. Kim, and Y. S. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *Advances in Cryptology – ASIACRYPT 2017, Part I*, ser. Lecture Notes in Computer Science, T. Takagi and T. Peyrin, Eds., vol. 10624. Hong Kong, China: Springer, Cham, Switzerland, Dec. 3–7, 2017, pp. 409–437. 1, 2.3.4, 4, C

[6] J. Yuan, W. Liu, J. Shi, and Q. Li, "Approximate homomorphic encryption based privacy-preserving machine learning: a survey," *Artificial Intelligence Review*, vol. 58, no. 3, p. 82, 2025. 1

[7] R. Geva, A. Gusev, Y. Polyakov, L. Liram, O. Rosolio, A. Alexandru, N. Genise, M. Blatt, Z. Duchin, B. Waissengrin, D. Mirelman, F. Bukstein, D. T. Blumenthal, I. Wolf, S. Pelles-Avraham, T. Schaffer, L. A. Lavi, D. Micciancio, V. Vaikuntanathan, A. A. Badawi, and S. Goldwasser, "Collaborative privacy-preserving analysis of oncological data using multiparty homomorphic encryption," *Proceedings of the National Academy of Sciences*, vol. 120, no. 33, p. e2304415120, 2023. [Online]. Available: https://www.pnas.org/doi/abs/10.1073/pnas.2304415120 1

[8] A. Al Badawi, L. Hoang, C. F. Mun, K. Laine, and K. M. M. Aung, "Privft: Private and fast text classification with homomorphic encryption," *IEEE Access*, vol. 8, pp. 226 544–226 556, 2020. 1

[9] A. Al Badawi, "Private svm inference on encrypted data," in *Federated Learning-A Systematic Review*. IntechOpen, 2024. 1

[10] A. Al Badawi and M. Faizal Bin Yusof, "Private pathological assessment via machine learning and homomorphic encryption," *BioData Mining*, vol. 17, no. 1, p. 33, 2024. 1

[11] L. Rovida and A. Leporati, "Transformer-based language models and homomorphic encryption: An intersection with bert-tiny," in *Proceedings of the 10th ACM International Workshop on Security and Privacy Analytics*, ser. IWSPA '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 3–13. [Online]. Available: https://doi.org/10.1145/3643651.3659893 1, 3.2.1, 3.4.2, 4.3.1, 1

[12] Y. Mu, G. Li, S. Chen, and J. Xue, "Pritran: Privacy-preserving inference for transformer-based language models under fully homomorphic encryption," in *2026 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, 2026, pp. 57–69. 1, 3.2.1, 4.3.1, 1

[13] L. Yang, J. Chen, W. Dai, S. Wang, W. Wu, and Y. Feng, "ARION: Attention-optimized transformer inference on encrypted data," Cryptology ePrint Archive, Report 2025/2271, 2025. [Online]. Available: https://eprint.iacr.org/2025/2271 1, 3.2.1, 3.3.1, 4.3.1, 1

[14] L. Zhang, X. Wang, J. J. Sim, Z. Huang, J. Zhong, H. WANG, P. Duan, and K.-Y. Lam, "MOAI: Module-optimizing architecture for non-interactive secure transformer inference," in *The Fourteenth International Conference on Learning Representations*, 2026. [Online]. Available: https://openreview.net/forum?id=qJn4HtTzhH 1, 3.2.1, 3.3.1, 4.3.1, 1, 2

[15] J. Zhang, X. Yang, L. He, K. Chen, W. jie Lu, Y. Wang, X. Hou, J. Liu, K. Ren, and X. Yang, "Secure transformer inference made non-interactive." in *NDSS*. The Internet Society, 2025. [Online]. Available: http://dblp.uni-trier.de/db/conf/ndss/ndss2025.html#ZhangYH0LWH00025 1, 3.2.1, 4.3.1, 1, 2

[16] J. Moon, D. Yoo, X. Jiang, and M. Kim, "THOR: Secure transformer inference with homomorphic encryption," in *ACM CCS 2025: 32nd Conference on Computer and Communications Security*, C.-Y. Huang, J.-C. Chen, S.-P. Shieh, D. Lie, and V. Cortier, Eds. Taipei, Taiwan: ACM Press, Oct. 13–17, 2025, pp. 3765–3779. 1, 3.2.1, 3.3.1, 1, 5

[17] L. de Castro, A. Polychroniadou, and D. Escudero, "Privacy-preserving large language model inference via gpu-accelerated fully homomorphic encryption," in *Neurips Safe Generative AI Workshop 2024*, 2024. 1, 3.2.2, 2

[18] S. Jayashankar, J. Kim, M. B. Sullivan, W. Zheng, and D. Skarlatos, "A scalable multi-gpu framework for encrypted large-model inference," 2025. 1, 3.3.2, 2, 5

[19] X. Gao, S. Fu, L. Liu, Z. Liu, Y. Luo, and Y. Wang, "Euston: Efficient and user-friendly secure transformer inference with non-interactivity," 2026. 1, 3.2.1, 4.3.1, 1

[20] Z. Wang, R. Chen, X. Gao, Y. Wang, L. Liu, Z. Lan, Z. Wang, S. Fu, Q. Wang, and X. Huang, "Stip: Efficient and secure non-interactive transformer inference via compact packing," 2026. 1, 3.2.1, 3.3.1, 4.3.1, 1

[21] W. Cho, G. Hanrot, T. Kim, M. Park, and D. Stehlé, "Fast and accurate homomorphic softmax evaluation," in *ACM CCS 2024: 31st Conference on Computer and Communications Security*, B. Luo, X. Liao, J. Xu, E. Kirda, and D. Lie, Eds. Salt Lake City, UT, USA: ACM Press, Oct. 14–18, 2024, pp. 4391–4404. 1, 3.2.2, 3.4.3, D, 3

[22] A. A. Badawi and Y. Polyakov, "Demystifying bootstrapping in fully homomorphic encryption," Cryptology ePrint Archive, Report 2023/149, 2023. [Online]. Available: https://eprint.iacr.org/2023/149 1

[23] C. Agulló-Domingo, Ó. Vera-López, S. Guzelhan, L. Daksha, A. El Jerari, K. Shivdikar, R. Agrawal, D. Kaeli, A. Joshi, and J. L. Abellán, "Fideslib: A fully-fledged open-source fhe library for efficient ckks on gpus," in *2025 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2025, pp. 1–3. 1, 3.3.2, 3.4.5, 4

[24] J. Cabrero-Holgueras and S. Pastrana, "SoK: Privacy-preserving computation techniques for deep learning," *Proceedings on Privacy Enhancing Technologies*, vol. 2021, no. 4, pp. 139–162, Oct. 2021. 1

[25] L. K. L. Ng and S. S. M. Chow, "SoK: Cryptographic neural-network computation," in *2023 IEEE Symposium on Security and Privacy*. San Francisco, CA, USA: IEEE Computer Society Press, May 21–25, 2023, pp. 497–514. 1

[26] D. Andreoletti, A. Rudi, E. Carpanzano, and T. Leidi, "Privacy-preserving llm inference in practice: A comparative survey of techniques, trade-offs, and deployability," 2026. 1

[27] M. Zbudila, A. Suresh, H. Yalame, O. Mirzamohammadi, A. Abidin, and B. Preneel, "SoK: Connecting the dots in privacy-preserving ML - systematization of MPC protocols and conversions between secret sharing schemes," Cryptology ePrint Archive, Paper 2025/1679, 2025. [Online]. Available: https://eprint.iacr.org/2025/1679 1

[28] Y. Chen, T. Lu, Z. Tang, B. Zhang, Z. Shi, Y. Luan, and Z. Wang, "SoK: Private transformer-based model inference," Cryptology ePrint Archive, Paper 2026/491, 2026. [Online]. Available: https://eprint.iacr.org/2026/491 1

[29] M. S. Riazi, B. Darvish Rouani, and F. Koushanfar, "Deep learning on private data," *IEEE Security & Privacy*, vol. 17, no. 6, pp. 54–63, 2019. 1

[30] M. Azraoui, M. Bahram, B. Bozdemir, S. Canard, E. Ciceri, O. Ermis, R. Masalha, M. Mosconi, M. Önen, M. Paindavoine, B. Rozenberg, B. Vialla, and S. Vicini, "Sok: Cryptography for neural networks," in *Privacy and Identity Management. Data for Better Living: AI and Privacy: 14th IFIP WG 9.2, 9.6/11.7, 11.6/SIG 9.2.2 International Summer School, Windisch, Switzerland, August 19–23, 2019, Revised Selected Papers*, M. Friedewald, M. Önen, E. Lievens, S. Krenn, and S. Fricker, Eds. Cham: Springer International Publishing, 2020, pp. 63–81. [Online]. Available: https://doi.org/10.1007/978-3-030-42504-3_5 1

[31] J. Luo, Y. Zhang, Z. Zhang, S. Liu, Y. Dong, H. Li, Y. Yu, H. Wang, X. Zhou, and Z. Xu, "Cryptography-based privacy-preserving large language models: a lifecycle survey of frameworks, methods, and future directions," *Artificial Intelligence Review*, 2025. 1

[32] W. Zeng, T. Xu, Y. Chen, Y. Zhou, M. Zhang, J. Tan, C. Hong, and M. Li, "Towards efficient privacy-preserving machine learning: A systematic review from protocol, model, and system perspectives," Cryptology ePrint Archive, Paper 2025/1365, 2025. [Online]. Available: https://eprint.iacr.org/2025/1365 1

[33] B. Yan, K. Li, M. Xu, Y. Dong, Y. Zhang, Z. Ren, and X. Cheng, "On protecting the data privacy of large language models (llms) and llm agents: A literature review," *High-Confidence Computing*, vol. 5, no. 2, p. 100300, 2025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2667295225000042 1

[34] T. Khan, M. Budzys, K. Nguyen, and A. Michalas, "SoK: Wildest dreams: Reproducible research in privacy-preserving neural network training," *Proceedings on Privacy Enhancing Technologies*, vol. 2024, no. 3, pp. 144–164, Jul. 2024. 1

[35] N. Papernot, P. McDaniel, A. Sinha, and M. P. Wellman, "Sok: Security and privacy in machine learning," in *2018 IEEE European symposium on security and privacy (EuroS&P)*. IEEE, 2018, pp. 399–414. 1

[36] J. Xue, X. Xin, W. Zhang, M. Zheng, Q. Song, M. Zhou, Y. Dong, D. Wang, X. Chen, J. Xie *et al.*, "Measuring computational universality of fully homomorphic encryption," in *Proceedings on Privacy Enhancing Technologies (PoPETs)*, 2026. 1

[37] I. Turc, M.-W. Chang, K. Lee, and K. Toutanova, "Well-read students learn better: On the importance of pre-training compact models," 2020. [Online]. Available: https://openreview.net/forum?id=BJg7x1HFvB 1, E

[38] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf 2.1

[39] L. Bergerat, I. Chillotti, D. Ligier, J.-B. Orfila, and S. Tap, "TFHE gets real: an efficient and flexible homomorphic floating-point arithmetic," Cryptology ePrint Archive, Report 2025/257, 2025. [Online]. Available: https://eprint.iacr.org/2025/257 2.3

[40] A. Kholod, Y. Polyakov, and M. Schlottke-Lakemper, "Secure numerical simulations using fully homomorphic encryption," *Computer Physics Communications*, vol. 318, p. 109868, 2026. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0010465525003704 2.3.1

[41] J. H. Cheon, J. Jeong, J. Lee, and K. Lee, "Privacy-preserving computations of predictive medical models with minimax approximation and non-adjacent form," in *FC 2017 Workshops*, ser. Lecture Notes in Computer Science, M. Brenner, K. Rohloff, J. Bonneau, A. Miller, P. Y. A. Ryan, V. Teague, A. Bracciali, M. Sala, F. Pintore, and M. Jakobsson, Eds., vol. 10323. Sliema, Malta: Springer, Cham, Switzerland, Apr. 7, 2017, pp. 53–74. 2.3.2

[42] M. S. Paterson and L. J. Stockmeyer, "On the number of nonscalar multiplications necessary to evaluate polynomials," *SIAM Journal on Computing*, vol. 2, no. 1, pp. 60–66, 1973. 2.3.2

[43] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) fully homomorphic encryption without bootstrapping," in *ITCS 2012: 3rd Innovations in Theoretical Computer Science*, S. Goldwasser, Ed. Cambridge, MA, USA: Association for Computing Machinery, Jan. 8–10, 2012, pp. 309–325. 2.3.4

[44] Z. Brakerski, "Fully homomorphic encryption without modulus switching from classical GapSVP," in *Advances in Cryptology – CRYPTO 2012*, ser. Lecture Notes in Computer Science, R. Safavi-Naini and R. Canetti, Eds., vol. 7417. Santa Barbara, CA, USA: Springer Berlin Heidelberg, Germany, Aug. 19–23, 2012, pp. 868–886. 2.3.4

[45] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," Cryptology ePrint Archive, Report 2012/144, 2012. [Online]. Available: https://eprint.iacr.org/2012/144 2.3.4

[46] A. Costache, B. R. Curtis, E. Hales, S. Murphy, T. Ogilvie, and R. Player, "On the precision loss in approximate homomorphic encryption," in *SAC 2023: 30th Annual International Workshop on Selected Areas in Cryptography*, ser. Lecture Notes in Computer Science, C. Carlet, K. Mandal, and V. Rijmen, Eds., vol. 14201. Fredericton, Canada: Springer, Cham, Switzerland, Aug. 14-18, 2024, pp. 325–345. 2.3.4

[47] A. Alexandru, A. Kim, and Y. Polyakov, "General functional bootstrapping using ckks," in *Advances in Cryptology – CRYPTO 2025*, Y. Tauman Kalai and S. F. Kamara, Eds. Cham: Springer Nature Switzerland, 2025, pp. 304–337. 2.5.2

[48] S. Cheon, Y. Lee, D. Kim, J. M. Lee, S. Jung, T. Kim, D. Lee, and H. Kim, "DaCapo: Automatic bootstrapping management for efficient fully homomorphic encryption," in *USENIX Security 2024: 33rd USENIX Security Symposium*, D. Balzarotti and W. Xu, Eds. Philadelphia, PA, USA: USENIX Association, Aug. 14–16, 2024. [Online]. Available: https://www.usenix.org/conference/usenixsecurity24/presentation/cheon 2.5.3

[49] W. Yang, Z. Liu, Y. Zhang, X. Bie, and D. Yao, "Secretvoice1.0: Non-interactive secure transformer inference," 2026. 3.2.1, 2

[50] J. Xie, S. Tan, V. N. Boddeti, and Z. Lu, "ATLAS: Automated approximation of transformers for efficient homomorphic inference in one hour," 2026. 3.2.1, 4.3.1, 1

[51] M. Ott, S. Edunov, A. Baevski, A. Fan, S. Gross, N. Ng, D. Grangier, and M. Auli, "fairseq: A fast, extensible toolkit for sequence modeling," in *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics (Demonstrations)*, 2019, pp. 48–53. 3.2.1

[52] D. Park, E. Lee, and J.-W. 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)*, 2025, pp. 11 090–11 111. 3.2.1, 4.3.1, 1

[53] S. Halevi and V. Shoup, "Algorithms in HElib," in *Advances in Cryptology – CRYPTO 2014, Part I*, ser. Lecture Notes in Computer Science, J. A. Garay and R. Gennaro, Eds., vol. 8616. Santa Barbara, CA, USA: Springer Berlin Heidelberg, Germany, Aug. 17–21, 2014, pp. 554–571. 3.2.1

[54] L. Lim, V. Kalagi, D. Agrawal, and A. El Abbadi, "Tricycle: Private transformer inference with tricyclic encodings," Cryptology ePrint Archive, Report 2025/1200, 2025. [Online]. Available: https://eprint.iacr.org/2025/1200 3.2.1, 3.3.1, 4.3.1, 1

[55] Y. Yu, Y. Zhou, Y. Chen, P. Soto, W. Xiong, and M. Li, "Cachemir: Fully homomorphic encrypted inference of generative large language model with kv cache," 2026. 3.2.1, 3.3.1, 2, 5

[56] Z. He, M. Wang, X. Gao, Y. Luo, L. Liu, and S. Fu, "Ensi: Efficient non-interactive secure inference for large language models," in *2025 44th International Symposium on Reliable Distributed Systems (SRDS)*, 2025, pp. 83–94. 3.2.1, 2

[57] H. Wang, S. Ma, L. Dong, S. Huang, H. Wang, L. Ma, F. Yang, R. Wang, Y. Wu, and F. Wei, "Bitnet: Scaling 1-bit transformers for large language models," 2023. 3.2.1

[58] S. N. Guzelhan, L. Daksha, C. A. Domingo, G. Jonatan, J. Kim, J. L. Abellan, D. Kaeli, and A. Joshi, "Ellmo: Packing-and depth-aware encrypted transformer inference," 2026. 3.2.1, 4.3.1, 1

[59] J. Park, S. Park, J. H. Park, J. H. Ahn, J. H. Cheon, G. Hanrot, J. W. Kim, M. Park, and D. Stehlé, "Scaling up privacy-preserving ml: A ckks implementation of llama-2-7b," 2026. 3.2.1, 2

[60] X. Jiang, M. Kim, K. E. Lauter, and Y. Song, "Secure outsourced matrix computation and application to neural networks," in *ACM CCS 2018: 25th Conference on Computer and Communications Security*, D. Lie, M. Mannan, M. Backes, and X. Wang, Eds. Toronto, ON, Canada: ACM Press, Oct. 15–19, 2018, pp. 1209–1222. 3.2.1

[61] D. Rho, T. Kim, M. Park, J. W. Kim, H. Chae, E. K. Ryu, and J. H. Cheon, "Encryption-friendly LLM architecture," in *The Thirteenth International Conference on Learning Representations*, 2025. [Online]. Available: https://openreview.net/forum?id=pbre0HKsfE 3.2.2, 4.3.1, 1

[62] N. Jovanovic, M. Fischer, S. Steffen, and M. T. Vechev, "Private and reliable neural network inference," in *ACM CCS 2022: 29th Conference on Computer and Communications Security*, H. Yin, A. Stavrou, C. Cremers, and E. Shi, Eds. Los Angeles, CA, USA: ACM Press, Nov. 7–11, 2022, pp. 1663–1677. 3.2.2

[63] I. Zimerman, M. Baruch, N. Drucker, G. Ezov, O. Soceanu, and L. Wolf, "Converting transformers to polynomial form for secure inference over homomorphic encryption," in *Proceedings of the 41st International Conference on Machine Learning*, ser. ICML'24. Vienna, Austria: JMLR.org, 2024. 3.2.2, 2

[64] Y. Chen, Q. Zeng, H. Ji, and Y. Yang, "Skyformer: Remodel self-attention with gaussian kernel and nystr\"om method," in *Advances in Neural Information Processing Systems*, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., 2021. [Online]. Available: https://openreview.net/forum?id=pZCYG7gjkKz 3.2.2

[65] J. Ramapuram, F. Danieli, E. G. Dhekane, F. Weers, D. Busbridge, P. Ablin, T. Likhomanenko, J. Digani, Z. Gu, A. Shidani, and R. Webb, "Theory, analysis, and best practices for sigmoid self-attention," in *The Thirteenth International Conference on Learning Representations*, 2025. [Online]. Available: https://openreview.net/forum?id=Zhdhg6n2OG 3.2.2

[66] A. Ebel, K. Garimella, and B. Reagen, "Orion: A fully homomorphic encryption framework for deep learning," in *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, 2025, pp. 734–749. 3.3.1, 3.3.2

[67] A. Al Badawi, J. Bates, F. Bergamaschi, D. B. Cousins, S. Erabelli, N. Genise, S. Halevi, H. Hunt, A. Kim, Y. Lee *et al.*, "Openfhe: Open-source fully homomorphic encryption library," in *proceedings of the 10th workshop on encrypted computing & applied homomorphic cryptography*, 2022, pp. 53–63. 3.3.2, 3.4.5, 4, 4.2.3

[68] T. Insight, "Lattigo v6," Online: https://github.com/tuneinsight/lattigo, Aug. 2024, ePFL-LDS, Tune Insight SA. 3.3.2, 4, 4.2.3

[69] H. Yang, S. Shen, W. Dai, L. Zhou, Z. Liu, and Y. Zhao, "Phantom: A cuda-accelerated word-wise homomorphic encryption library," *IEEE Trans. Dependable Secur. Comput.*, vol. 21, no. 5, pp. 4895–4906, 2024. [Online]. Available: https://doi.org/10.1109/TDSC.2024.3363900 3.3.2

[70] DESILO, "Liberate.FHE: A New FHE Library for Bridging the Gap between Theory and Practice with a Focus on Performance and Accuracy," 2023, https://github.com/Desilo/liberate-fhe. 3.3.2, 4

[71] HEIR Contributors, "HEIR: Homomorphic encryption intermediate representation," https://heir.dev, 2024, accessed August 2026. 3.3.2

[72] R. Zhang, Z. Zheng, and W. Bao, "Practical secure inference algorithm for a fine-tuned large language model based on fully homomorphic encryption," *IEEE Transactions on Information Forensics and Security*, vol. 21, pp. 17–29, 2026. 3.3.3

[73] H. Chen, K. Laine, and R. Player, "Simple encrypted arithmetic library - SEAL v2.1," in *FC 2017 Workshops*, ser. Lecture Notes in Computer Science, M. Brenner, K. Rohloff, J. Bonneau, A. Miller, P. Y. A. Ryan, V. Teague, A. Bracciali, M. Sala, F. Pintore, and M. Jakobsson, Eds., vol. 10323. Sliema, Malta: Springer, Cham, Switzerland, Apr. 7, 2017, pp. 3–18. 4, 4.2.3

[74] E. Aharoni, A. Adir, M. Baruch, N. Drucker, G. Ezov, A. Farkash, L. Greenberg, R. Masalha, G. Moshkowich, D. Murik, H. Shaul, and O. Soceanu, "HeLayers: A tile tensors framework for large neural networks on encrypted data," *Proceedings on Privacy Enhancing Technologies*, vol. 2023, no. 1, pp. 325–342, Jan. 2023. 4

[75] H. Yang, S. Shen, W. Dai, L. Zhou, Z. Liu, and Y. Zhao, "Phantom: A cuda-accelerated word-wise homomorphic encryption library," *IEEE Transactions on Dependable and Secure Computing*, vol. 21, no. 5, pp. 4895–4906, 2024. 4, 4.2.3

[76] Z. Yuan, H. Zhang, Y. E. Zhou, A. Mohan, I. Chung, S. Seelam, J. Huang *et al.*, "Cost-efficient llm training with lifetime-aware tensor offloading via gpudirect storage," 2025. 4.3.1

[77] M. Albrecht, M. Chase, H. Chen, J. Ding, S. Goldwasser, S. Gorbunov, S. Halevi, J. Hoffstein, K. Laine, K. Lauter, S. Lokam, D. Micciancio, D. Moody, T. Morrison, A. Sahai, and V. Vaikuntanathan, "Homomorphic encryption standard," Cryptology ePrint Archive, Report 2019/939, 2019. [Online]. Available: https://eprint.iacr.org/2019/939 4.3.2

[78] Y. Zhai, C. Jiang, L. Wang, X. Jia, S. Zhang, Z. Chen, X. Liu, and Y. Zhu, " ByteTransformer: A High-Performance Transformer Boosted for Variable-Length Inputs ," in *2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. Los Alamitos, CA, USA: IEEE Computer Society, May 2023, pp. 344–355. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/IPDPS54959.2023.00042 5

[79] ——, "ByteTransformer: A high-performance transformer boosted for variable-length inputs," https://github.com/bytedance/ByteTransformer, 2022, accessed: 2026-04-24. 5

[80] J. Li, J. Xu, S. Huang, Y. Chen, W. Li, J. Liu, Y. Lian, J. Pan, L. Ding, H. Zhou, Y. Wang, and G. Dai, "Large language model inference acceleration: A comprehensive hardware perspective," 2025. [Online]. Available: https://arxiv.org/abs/2410.04466 5

[81] NVIDIA Corporation, "NVIDIA NIM LLMs benchmarking: Performance – Llama-3.1-8B-Instruct results," https://docs.nvidia.com/nim/benchmarking/llm/latest/performance.html#version-1-3-0, 2025, accessed: 2026-04-24. 5

[82] J. H. Cheon, G. Hanrot, J. Kim, and D. Stehlé, "Ship: A shallow and highly parallelizable ckks bootstrapping algorithm," in *Advances in Cryptology – EUROCRYPT 2025*, S. Fehr and P.-A. Fouque, Eds. Cham: Springer Nature Switzerland, 2025, pp. 398–428. 5

[83] J.-S. Coron and T. Seuré, "Paco: Bootstrapping for ckks via partial coefftoslot," in *Advances in Cryptology – ASIACRYPT 2025*, G. Hanaoka and B.-Y. Yang, Eds. Singapore: Springer Nature Singapore, 2026, pp. 36–67. 5

[84] N. Samardzic, A. Feldmann, A. Krastev, S. Devadas, R. Dreslinski, C. Peikert, and D. Sanchez, "F1: A fast and programmable accelerator for fully homomorphic encryption," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 238–252. 5

[85] A. Golder, R. Kumar, S. Taneja, K. Race, P. Aseron, J. Greensky, W. Wang, H. Gong, L. Kethareswaran, V. Suresh *et al.*, "Heracles: 8192-way simd programmable scalable fully-homomorphic encryption soc for privacy-preserving cloud computing in intel 3 cmos," in *2026 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 69. IEEE, 2026, pp. 424–426. 5

[86] D. B. Cousins, Y. Polyakov, A. A. Badawi, M. French, A. Schmidt, A. Jacob, B. Reynwar, K. Canida, A. Jaiswal, C. Mathew *et al.*, "Trebuchet: Fully homomorphic encryption accelerator for deep computation," *arXiv preprint arXiv:2304.05237*, 2023. 5

[87] R. Geelen, M. Van Beirendonck, H. V. L. Pereira, B. Huffman, T. McAuley, B. Selfridge, D. Wagner, G. D. Dimou, I. Verbauwhede, F. Vercauteren, and D. W. Archer, "BASALISC: Programmable hardware accelerator for BGV fully homomorphic encryption," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2023, no. 4, pp. 32–57, 2023. 5

[88] J. Devlin, M.-W. Chang, K. Lee, and K. 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)*, J. Burstein, C. Doran, and T. Solorio, Eds. Minneapolis, Minnesota: Association for Computational Linguistics, Jun. 2019, pp. 4171–4186. [Online]. Available: https://aclanthology.org/N19-1423/ 1

[89] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," OpenAI, Tech. Rep., 2018, available at: https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf. 2

[90] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample, "Llama: Open and efficient foundation language models," 2023. [Online]. Available: https://arxiv.org/abs/2302.13971 3

[91] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. 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*, T. Linzen, G. Chrupała, and A. Alishahi, Eds. Brussels, Belgium: Association for Computational Linguistics, Nov. 2018, pp. 353–355. [Online]. Available: https://aclanthology.org/W18-5446/ A.3

[92] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Ng, and C. Potts, "Recursive deep models for semantic compositionality over a sentiment treebank," in *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, D. Yarowsky, T. Baldwin, A. Korhonen, K. Livescu, and S. Bethard, Eds. Seattle, Washington, USA: Association for Computational Linguistics, Oct. 2013, pp. 1631–1642. [Online]. Available: https://aclanthology.org/D13-1170/ A.3

[93] I. Dagan, O. Glickman, and B. Magnini, "The pascal recognising textual entailment challenge," in *Proceedings of the First International Conference on Machine Learning Challenges: Evaluating Predictive Uncertainty Visual Object Classification, and Recognizing Textual Entailment*, ser. MLCW'05. Berlin, Heidelberg: Springer-Verlag, 2005, p. 177–190. [Online]. Available: https://doi.org/10.1007/11736790_9 A.3

[94] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, "SQuAD: 100,000+ questions for machine comprehension of text," in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, J. Su, K. Duh, and X. Carreras, Eds. Austin, Texas: Association for Computational Linguistics, Nov. 2016, pp. 2383–2392. [Online]. Available: https://aclanthology.org/D16-1264/ A.3

[95] B. Dolan and C. Brockett, "Automatically constructing a corpus of sentential paraphrases," in *Third International Workshop on Paraphrasing (IWP2005)*. Asia Federation of Natural Language Processing, January 2005. [Online]. Available: https://www.microsoft.com/en-us/research/publication/automatically-constructing-a-corpus-of-sentential-paraphrases/ A.3

[96] S. Merity, C. Xiong, J. Bradbury, and R. Socher, "Pointer sentinel mixture models," 2016. A.3

[97] D. Paperno, G. Kruszewski, A. Lazaridou, N. Q. Pham, R. Bernardi, S. Pezzelle, M. Baroni, G. Boleda, and R. Fernández, "The LAMBADA dataset: Word prediction requiring a broad discourse context," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, K. Erk and N. A. Smith, Eds. Berlin, Germany: Association for Computational Linguistics, Aug. 2016, pp. 1525–1534. [Online]. Available: https://aclanthology.org/P16-1144/ A.3

[98] R. Zellers, A. Holtzman, Y. Bisk, A. Farhadi, and Y. Choi, "HellaSwag: Can a machine really finish your sentence?" in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, A. Korhonen, D. Traum, and L. Màrquez, Eds. Florence, Italy: Association for Computational Linguistics, Jul. 2019, pp. 4791–4800. [Online]. Available: https://aclanthology.org/P19-1472/ A.3

[99] P. Clark, I. Cowhey, O. Etzioni, T. Khot, A. Sabharwal, C. Schoenick, and O. Tafjord, "Think you have solved question answering? try ARC, the AI2 reasoning challenge," 2018. A.3

[100] A. Henzinger, M. M. Hong, H. Corrigan-Gibbs, S. Meiklejohn, and V. Vaikuntanathan, "One server for the price of two: Simple and fast single-server private information retrieval," in *USENIX Security 2023: 32nd USENIX Security Symposium*, J. A. Calandrino and C. Troncoso, Eds. Anaheim, CA, USA: USENIX Association, Aug. 9–11, 2023, pp. 3889–3905. [Online]. Available: https://www.usenix.org/conference/usenixsecurity23/presentation/henzinger B

[101] S. J. Menon and D. J. Wu, "SPIRAL: Fast, high-rate single-server PIR via FHE composition," in *2022 IEEE Symposium on Security and Privacy*. San Francisco, CA, USA: IEEE Computer Society Press, May 22–26, 2022, pp. 930–947. B

[102] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Stealing machine learning models via prediction APIs," in *USENIX Security 2016: 25th USENIX Security Symposium*, T. Holz and S. Savage, Eds. Austin, TX, USA: USENIX Association, Aug. 10–12, 2016, pp. 601–618. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/tramer B

[103] N. Carlini, S. Garg, S. Jha, S. Mahloujifar, M. Mahmoody, and F. Tramer, "Neuracrypt is not private," 2021. B

[104] Y. Yang, X. Huang, X. Liu, H. Cheng, J. Weng, X. Luo, and V. Chang, "A comprehensive survey on secure outsourced computation and its applications," *IEEE Access*, vol. 7, pp. 159 426–159 465, 2019. B.2

[105] B. Li and D. Micciancio, "On the security of homomorphic encryption on approximate numbers," in *Advances in Cryptology – EUROCRYPT 2021, Part I*, ser. Lecture Notes in Computer Science, A. Canteaut and F.-X. Standaert, Eds., vol. 12696. Zagreb, Croatia: Springer, Cham, Switzerland, Oct. 17–21, 2021, pp. 648–677. B.2

[106] J.-P. Bossuat, R. Cammarota, I. Chillotti, B. R. Curtis, W. Dai, H. Gong, E. Hales, D. Kim, B. Kumara, C. Lee, X. Lu, C. Maple, A. Pedrouzo-Ulloa, R. Player, Y. Polyakov, L. A. R. Lopez, Y. Song, and D. Yhee, "Security guidelines for implementing homomorphic encryption," *IACR Communications in Cryptology (CiC)*, vol. 1, no. 4, p. 26, 2024. B.2

Appendix A. Detailed LLM Architecture and Components

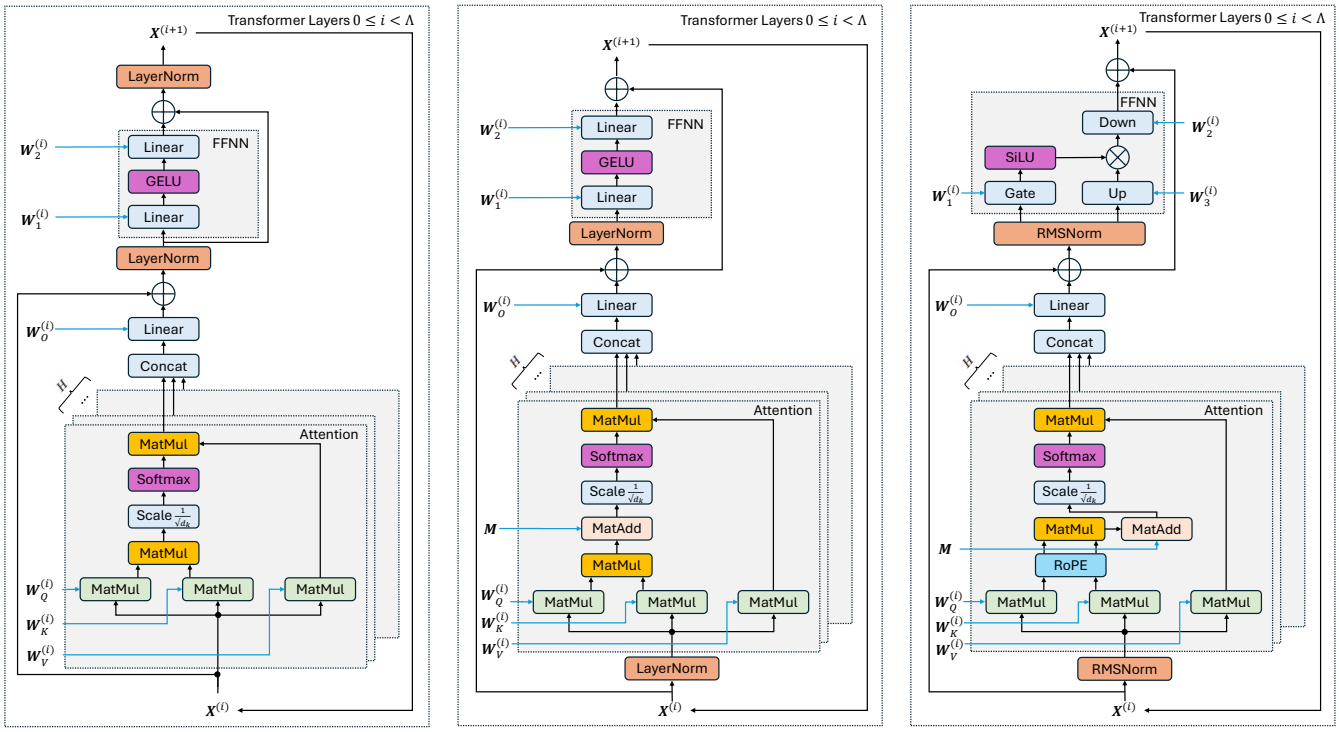
A.1. The Transformer Layer

Transformer-based LLMs are composed of an array of Λ identical transformer layers. The forward pass (inference) is iterative by nature: the input matrix ($\mathbf{X}_0 = [\mathbf{t}_0, \mathbf{t}_1, \ldots, \mathbf{t}_{n-1}]$) to the first layer is a concatenation of the vector embeddings of the n input tokens $\{\mathbf{t}_i\}_{i=0}^{n-1}$ (a.k.a. input sequence), whereas the input to each subsequent layer is the output of the preceding layer. Each layer includes two sub-layers: 1) multi-head attention (with H heads) and 2) FFNN. These sub-layers are connected by residual connections (the Direct Sum block ($\oplus$) in Figure 1), such that the output of a sub-layer is defined as $\mathbf{Y} = \text{Norm}(\mathbf{X} + \text{sub-layer}(\mathbf{X}))$, or $\mathbf{Y} = \mathbf{X} + \text{sub-layer}(\text{Norm}(\mathbf{X}))$, where the exact definition and positioning of the normalization block vary across the transformer architecture.

A.2. Model Families

Figure 1 illustrates the three main LLM architecture families studied in this work:

1) **The BERT Family (Encoder-Only)**: BERT [88] is designed to learn contextual bidirectional representations.



(a) BERT Architecture (b) GPT Architecture (c) Llama Architecture

Figure 1. Architectural comparison of Transformer-based models: (a) BERT, a bidirectional encoder; (b) GPT, a unidirectional decoder-only path; and (c) Llama, an optimized decoder with RoPE, RMSNorm, and SwiGLU FFNN.

It conditions each token on its entire global context (both left and right sequences simultaneously) within a given input, rather than processing the sequence in a unidirectional, autoregressive manner. A BERT attention sub-layer incorporates matrix multiplication blocks (MatMul) for transforming the input representations using the learned linear projections into *Query* $\mathbf{Q}_i = \mathbf{X}_i\mathbf{W}_Q$, *Key* $\mathbf{K}_i = \mathbf{X}_i\mathbf{W}_K$, and *Value* $\mathbf{V}_i = \mathbf{X}_i\mathbf{W}_V$ matrices. The attention scores are computed as the inner product of Query and Key, scaled by the square root of the key dimension: ($\mathbf{S}_i = \frac{\mathbf{Q}_i\mathbf{K}_i^T}{\sqrt{d_k}}$). The attention scores matrix $\mathbf{S}_i$ is then converted into a probabilities matrix ($\mathbf{P}_i$) using Equation (1) or the more numerically stable version in Equation (2). The probabilities matrix $\mathbf{P}_i$ is then used to compute a weighted sum of the associated $\mathbf{V}_i$ matrix. The outputs from all attention heads are concatenated and passed through a final linear projection to restore the original hidden dimension.

$$(\mathbf{P}_i)_{r,c} = \text{Softmax}(\mathbf{S}_i)_{r,c} = \frac{\exp((\mathbf{S}_i)_{r,c})}{\sum_k \exp((\mathbf{S}_i)_{r,k})}, \quad (1)$$

$$(\mathbf{P}_i)_{r,c} = \frac{\exp((\mathbf{S}_i)_{r,c} - \max_k(\mathbf{S}_i)_{r,k})}{\sum_k \exp((\mathbf{S}_i)_{r,k} - \max_k(\mathbf{S}_i)_{r,k})}. \quad (2)$$

BERT's FFNN sub-layer consists of a Gaussian Error Linear Unit (GELU) activation function (Equation 3) sandwiched between two linear layers (MatMul defined by the learned matrices $\mathbf{W}_1$ and $\mathbf{W}_2$)

$$\text{GELU}(x) = \frac{x}{2}\left[1 + \text{erf}\left(\frac{x}{\sqrt{2}}\right)\right]. \quad (3)$$

BERT typically uses Post-Layer Normalization (Post-LN), where the LayerNorm block (Equation 4) is applied after the residual addition for each row $\mathbf{h}$ of the hidden state matrix ($\mathbf{H}$)

$$\text{LayerNorm}(\mathbf{h}) = \gamma\left(\frac{\mathbf{h} - \mu_\mathbf{h}}{\sqrt{\sigma_\mathbf{h}^2 + \epsilon}}\right) + \beta. \quad (4)$$

2) **The GPT Family (Decoder-Only)**: GPT [89] is designed for autoregressive next-token prediction. Similar to BERT, GPT uses MatMul and scaled attention. However, it introduces causal masking, implemented as MatAdd with $\mathbf{M}$ defined in Equation (5). Causal masking is applied to the attention scores to ensure that each token r can only attend to current and preceding tokens $c \le r$. The FFNN sub-layer in GPT is identical to that in BERT.

$$M_{r,c} = \begin{cases} 0 & , r \ge c \\ -\infty & , r < c \end{cases}. \quad (5)$$

GPT models use Pre-Layer Normalization (Pre-LN), where the normalization occurs before the attention and FFNN sub-layers.

3) **The Llama Family (Optimized Decoder)**: Llama [90] introduces several optimizations to the GPT architecture. First, it uses Rotary Positional Embeddings (RoPE) by multiplying the Query and Key matrices by constant sinusoidal rotary matrices. Second, the FFNN sub-layer uses a Swish-Gated Linear Unit (SwiGLU) structure involving three learned weight matrices: Gate ($\mathbf{W}_1$), Up ($\mathbf{W}_3$), and Down ($\mathbf{W}_2$). The activation function used is SiLU, which is defined in Equation (6)

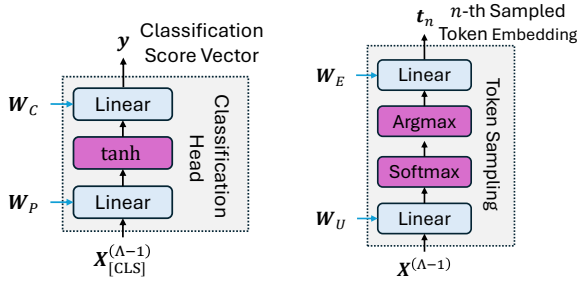
$$\text{SiLU}(x) = x \cdot \sigma(x) = \frac{x}{1 + e^{-x}}. \tag{6}$$

Third, Llama uses Root Mean Square Layer Normalization (RMSNorm) instead of standard LayerNorm. RMSNorm is more efficient because it omits the mean-centering step, as shown in Equation (7)

$$\text{RMSNorm}(\mathbf{h}) = \frac{\mathbf{h}}{\sqrt{\frac{1}{|\mathbf{h}|}\sum_{i=0}^{|\mathbf{h}|-1} h_i^2 + \epsilon}} \odot \gamma. \tag{7}$$

A.3. Downstream Tasks and Main Datasets

The frameworks addressed in this work primarily target two downstream tasks: sequence classification and autoregressive token generation, as shown in Figure 2.



(a) Sequence classification. (b) Autoregressive token generation.

Figure 2. Downstream task architectures: (a) sequence classification via pooled hidden states, and (b) autoregressive token generation via a four-step pipeline. $\mathbf{W}_P$ and $\mathbf{W}_C$ are pooler and classifier projection weight matrices. $\mathbf{W}_U$ and $\mathbf{W}_E$ are the vocabulary unembedding and embedding projection weight matrices.

Sequence Classification. For classification tasks, the intermediate representation of the first sequence token (known as the classification `[CLS]` token) is extracted from the final transformer encoder layer (e.g., the BERT family). This token is fed to a *Pooler* block, which comprises a linear layer and the tanh activation function to generate a fixed-size representation of the entire input sequence. Finally, a *Classifier* block uses another linear layer to project the pooled vector down to the specific number of target classes by producing the output *logits* (classification scores). The evaluation of these classification tasks is typically carried out using the General Language Understanding Evaluation (GLUE) benchmark [91] reporting performance on datasets such as SST-2 [92] (sentiment), Recognizing Textual Entailment (RTE) [93] (textual entailment), Question-answering Natural Language Inference (QNLI) [94], and Microsoft Research Paraphrase Corpus (MRPC) [95] (paraphrasing).

Autoregressive Token Generation. For most of the frameworks employing decoder-only architectures (e.g., the GPT and Llama families), the target downstream task is continuous token generation and logical reasoning. Here, the sequence output from the final transformer block goes through a four-step generation pipeline as follows:

1) *Final Projection*: the sequence output is fed to a linear projection layer that maps the output from the hidden model dimension to the full vocabulary space (e.g., 50,257 for GPT-2 or 128,256 for Llama-3).
2) Softmax: these raw scores are converted into a logit-based probability distribution using Softmax.
3) *Token Sampling*: the probability distribution from (2) is converted into a one-hot vector via $\arg\max$.
4) *Embedding Retrieval*: the one-hot vector is used to retrieve the sampled token's representation from the pre-trained embedding table via matrix multiplication or a lookup operation.

The generative frameworks employ various datasets such as *Wikitext*-103 and *Wikitext*-2 [96] (language modeling capabilities), *Lambada* [97] (contextual next-word prediction), *HellaSwag* [98] (commonsense reasoning), and *ARC Easy* [99] (multi-step Question Answering reasoning).

Appendix B. More on System and Threat Model

Our focus on this system model is primarily driven by computational efficiency. Most practical applications of FHE employ public weights or involve private queries over public datasets, as seen in SoA Private Information Retrieval (PIR) systems [100], [101]. A double-blinded privacy model, where both inputs and model weights are encrypted, introduces significant overhead, including a linear increase in encrypted-weight storage per client, higher noise growth due to ciphertext-ciphertext multiplications, and lower numerical precision. Furthermore, protecting model weights is also a major challenge on its own [102], as various attacks can potentially reconstruct the model from the inference results [103] and complicate key management, requiring multiparty FHE schemes.

B.1. System Model

We consider frameworks where the system architecture consists of two primary entities: the Client (Data Owner) and the Server (Model Host), as shown in Figure 3.

The Client ($\mathcal{C}$). The client holds private input data x. To initiate the protocol, the client first instantiates the CKKS scheme, generating a secret key sk and a corresponding set of public evaluation keys. The client further performs local processing, which may include text tokenization and embedding generation. We assume the model's embedding

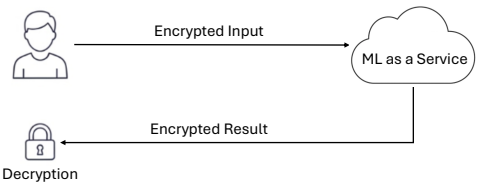


Figure 3. System model for the single-round MLaaS protocol.

look-up tables are available to the client. Once the embeddings are generated, the client encrypts them using sk under the CKKS scheme. The computed ciphertexts $\{ct_{\text{in}}\}$ along with the public evaluation keys are sent to the server for homomorphic evaluation. Once the server completes its computation, it transmits the output ciphertexts $\{ct_{\text{out}}\}$ back to the client, who uses sk to decrypt them and learn the final inference output.

The Server ($\mathcal{S}$). The server hosts an LLM defined by its architecture and pre-trained weight matrices $\{\mathcal{W}\}$, which remain in the clear. Upon receiving the client's ciphertexts $\{ct_{\text{in}}\}$, the server evaluates the model's forward pass directly on the encrypted embeddings using the client's evaluation keys. This involves the homomorphic evaluation of both linear and non-linear parts of the model architecture. The output encrypted logits or tokens $\{ct_{\text{out}}\}$ are returned to the client without the server ever accessing the underlying plaintext.

B.2. Threat Model

We adopt the *semi-honest* threat model typically employed in outsourced FHE computations [104]. In this threat model, both the client and server are assumed to follow the protocol steps as specified. However, the server may attempt to learn additional information from the received data. The security goal is to ensure that the server learns nothing about the client's input tokens or final inference result beyond what can be inferred from the public parameters of the chosen LLM, such as the embedding dimensions, number of layers, the input language, and the vocabulary size.

We also assume the client is the sole consumer of the computation results; that is, decrypted outputs are never shared with the server or any third party. This assumption is critical to maintaining the security guarantees of the Indistinguishability under Chosen Plaintext Attack (IND-CPA) model, as it prevents the server from exploiting decryption oracles to recover the secret key [105]. The reader is referred to Section 3 in [106] for further details on this threat model.

Appendix C. The CKKS Scheme

The CKKS scheme [5] is an approximate FHE scheme that enables computation on encrypted floating-point data, supporting both complex and real data types. In most practical instantiations, CKKS is constructed over the cyclotomic ring $R = \mathbb{Z}[x]/\Phi_M(x)$, where M is chosen as a power of two, resulting in the M-th cyclotomic polynomial $\Phi_M(x) = x^N + 1$ with $N = M/2$. To support leveled computations (up to a maximum level L), the scheme utilizes a sequence of quotient rings $R_{Q_l} = R/Q_l R = \mathbb{Z}_{Q_l}[x]/\Phi_M(x)$, where Q_l represents a decreasing chain of moduli $Q_L > Q_{L-1} > \cdots > Q_0$. Typically, these are implemented in Residue Number System (RNS) where $Q_l = \prod_{i=0}^{l} q_i$ for distinct primes q_i. Throughout the remainder of this paper, the RNS-CKKS scheme is referred to simply as CKKS.

CKKS operates on two primary data objects: 1) *plaintext* and 2) *ciphertext*. Each plaintext encodes a vector of real numbers (typically of size $N/2$) into a polynomial in R_{Q_l}. Encrypting a plaintext object results in a ciphertext object, which is a pair of elements in R_{Q_l}.

The following are the main operations in CKKS:

- **Setup** $(1^\lambda, L)$**:** given a security parameter λ and a maximum level L, fixes the global parameters: ring dimension N, scaling factor Δ, modulus chain $\{Q_0, \ldots, Q_L\}$, and distributions $\chi_{\text{err}}, \chi_{\text{key}}$. It outputs the public parameter set params.
- **Key Generation** ($\mathsf{KeyGen}(\mathsf{params})$): outputs the secret key sk, the public key pk, and the evaluation keys set[7] $\mathcal{EK} = (evk, \{rk^{(r_i)}\})$, comprising a relinearization key and a rotation key for each rotation index r_i.
- **Encoding and Decoding:** $\mathsf{Ecd}(z; \Delta)$ maps a complex vector $z \in \mathbb{C}^{N/2}$ to a plaintext $m \in R$, and $\mathsf{Dcd}(m; \Delta)$ recovers an approximation of z from m.
- **Encryption** ($\mathsf{Enc}(m, l, \cdot)$): produces a ciphertext $c \in R_{Q_l}^2$ at level l from plaintext m, using either pk or s.
- **Decryption** ($\mathsf{Dec}(c, sk)$): outputs a plaintext $m' \in R$ approximating the originally encrypted message.
- **Homomorphic Addition** (Add): adds two ciphertexts (Ct-Ct) or a ciphertext and a plaintext (Ct-Pt) element-wise, yielding a ciphertext encrypting the sum.
- **Homomorphic Multiplication** (Mult): multiplies two ciphertexts (Ct-Ct), followed by relinearization via evk and *key-switching* to restore the two-component form, or multiplies a ciphertext and a plaintext (Ct-Pt) element-wise.
- **Rescaling** ($\mathsf{RS}(c)$): consumes one level ($l \to l-1$) to restore the scaling factor Δ and reduce noise, typically applied after each multiplication.
- **Homomorphic Rotation** ($\mathsf{Rot}(c, r)$): cyclically shifts the encoded slots by r positions (left if $r > 0$, right if $r < 0$) via a rotation key and key-switching with $rk^{(r)}$.
- **Bootstrapping** ($\mathsf{BT}(c)$): refreshes an exhausted ciphertext by homomorphically evaluating an approximation of the decryption circuit, producing a ciphertext at level $l_{\text{BT}} = L - L_{\text{BT}}$, where L_{BT} is the *multiplicative depth* consumed during bootstrapping.

Appendix D. Private LLM Card System (PLCS) Example

Figure 4 illustrates the four PLCS reporting cards defined in §4.2, as instantiated for $\mathsf{SplitPrompt}$.

7. These keys are used to restore the ciphertext to its original form.

1. Model Card

Base Architecture: BERT-Mini, encoder-only (Λ=4 layers, d=256, H=4 heads, d_{ff}=1024, ≈11 M parameters)
Checkpoint: standard BERT-Mini fine-tuned on SST-2 in the clear; weights and architecture unchanged
Linear Blocks [Class L-ROW] (§3.2.1):

- Tiled token/row packing with 128×128 tiles: 2 ciphertexts per token

Non-Linear Blocks [Class N-PRE] (§3.2.2):

- Softmax$\rightarrow$ normalize-and-square [21], input shift -25, k=3 squarings; exp as Chebyshev deg. 6 on $[-6.59, -0.009]$
- LayerNorm $1/\sqrt{\cdot} \rightarrow$ Chebyshev deg. 59 on $[0.417, 242.3]$
- GELU$\rightarrow$ Chebyshev deg. 119 on $[-40.1, 62.0]$; tanh (pooler) $\rightarrow$ deg. 31 on $[-6.56, 6.55]$

Retraining: none. **Calibration data:** plaintext runs on SST-2 sentences.

(a)

2. CKKS Cryptographic Parameters Card

Protocol Paradigm: non-interactive, full server outsourcing (§2.2)
Security Level (λ): 128-bit
Ring Dimension (N): 2^{16} (2^{14} slots used)
Scaling Factor (Δ): 2^{50}; first modulus 53 bits
Secret Key Distribution: sparse ternary
Depth Budget & Bootstrapping (§3.3.1):

- Compute levels: 10; BT depth (L_{BT}): 16; total depth: 26
- Placement: static, manual; BT wraps every non-linear block

(b)

3. System Deployment Card

Compute Hardware (§3.3.2):

- GPU backend: 1$\times$ NVIDIA A100 (all latency figures)
- CPU backend: multi-core CPU (verification oracle; identical circuit)

Software Stack: OpenFHE (CPU) + FIDESlib (GPU); no FHE compiler
Client–Server Boundary: the client tokenizes, embeds, and encrypts; the server evaluates all encoder layers, the pooler, and the classifier under encryption; the client decrypts the logits

(c)

4. Downstream Application Card

Task Category: sequence classification, SST-2
Inference Paradigm: entire prompt encrypted; batch size 1
Sequence Length: 22.6 tokens on average over 10 validation sentences; latency normalized to 24 tokens
Utility Metrics:

- Plaintext accuracy 70.0% (F1 0.667) vs. encrypted accuracy 70.0% (F1 0.667) on the 10 sentences: loss 0%
- Fixed-degree plaintext circuit on 829 validation sentences: mild (<0.5%) loss

Runtime Performance:

- Unamortized latency: 83 s per 24-token sequence (3.47 s/token)

(d)

Figure 4. PLCS cards instantiated for POLARIS on BERT-Mini (the configuration of Table 3).

Appendix E. BERT Models

Table 4 summarizes the 5 BERT models [37] which can be evaluated under encryption in POLARIS.

TABLE 3. POLARIS PLCS CONFIGURATION FOR BERT-MINI ON SST-2. CHEBYSHEV ROWS GIVE THE APPROXIMATION INTERVAL AND POLYNOMIAL DEGREE. ACC. STANDS FOR ACCURACY, SEQ. FOR SEQUENCE (A SAMPLE SENTENCE), AND TOK. FOR TOKEN.

Model	Architecture	BERT-Mini
	Parameters	≈11M (Λ=4, d=256, H=4)
	Lin./non-lin. class	L-ROW / N-PRE
CKKS	N / slots	2^{16} / 2^{14}
	Scaling / first mod. (bits)	50 / 53
	Secret-key dist.	SPARSE_TERNARY
	Levels / L_{BT} / depth	10 / 16 / 26
	BT placement	Static, manual
Softmax	Algorithm	*normalize-and-square* [21]
	Input shift	-25
	Squaring iters (k)	3
Chebyshev approx.	$\exp(x)$	$[-6.59,\ -0.009]$, deg 6
	$1/\sqrt{x}$ (Softmax, iter 1)	$[0.00627,\ 1.442]$, deg 66
	$1/\sqrt{x}$ (Softmax, iter 2)	$[0.0184,\ 0.685]$, deg 24
	$1/\sqrt{x}$ (Softmax, iter 3)	$[0.0191,\ 0.987]$, deg 29
	$1/\sqrt{x}$ (LayerNorm)	$[0.417,\ 242.3]$, deg 59
	GELU(x)	$[-40.1,\ 62.0]$, deg 119
	$\tanh(x)$ (Pooler)	$[-6.56,\ 6.55]$, deg 31
System	Hardware	1× NVIDIA A100
	Libraries	OpenFHE + FIDESLIB
	Protocol	Non-interactive
Application	Task	Seq. classification – SST-2
	Plaintext acc.† (10 seq.)	70.0% (F1: 0.667)
	FHE acc. (10 seq.)	70.0% (F1: 0.667)
	Accuracy loss*	0%
	Avg. latency	3.47 s/tok. (83 s/seq., ≈24 tok.)

† Sequences are sampled randomly from the validation dataset. * Plaintext validation on 829 seq. shows mild ($< 0.5\%$) accuracy loss.

TABLE 4. BERT MODELS SUPPORTED IN POLARIS.

Model	d	Λ	H	Number of Params
BERT-Tiny	128	2	2	4.4 M
BERT-Mini	256	4	4	11.3 M
BERT-Small	512	4	8	29.1 M
BERT-Medium	512	8	8	41.7 M
BERT-Base	768	12	12	110 M