SoK of Private Deep Neural Network Inference with Approximate Fully Homomorphic Encryption

ZAIRA PINDADO, Barcelona Supercomputing Center (BSC), Spain
THOMAS SPENDLHOFER, Barcelona Supercomputing Center (BSC), Spain
MOHAMED ALLAM, Barcelona Supercomputing Center (BSC), Spain
PRIYAM MEHTA, Barcelona Supercomputing Center (BSC), Spain
LENA MARTENS, Barcelona Supercomputing Center (BSC), Spain
ANTONIO J. PEÑA, Barcelona Supercomputing Center (BSC), Spain

Deep neural networks (DNNs), a hot topic in this decade, are already solving many practical problems previously unsolved. There are clear use cases with strong privacy requirements for the input data in DNN inference. Fully Homomorphic Encryption (FHE) schemes provide privacy by enabling operations on encrypted data with post-quantum security, at the expense of a vast increase in data size. Overwhelming execution times and memory sizes currently limit DNN inference with FHE to severely reduced models and datasets. In this article, we thoroughly review the state of the art and the state of the practice around this topic, and identify the current challenges remaining to enable efficient DNN inference with FHE in production-sized use cases, along with the most promising trends to address them. Advancing upon previous review articles in the literature, our focus is specifically on deep learning inference on top of high-performance hardware. From our analysis, we define what we consider to be an ideal private inference system for DNNs, capturing notions already present in the literature and generalizing them.

CCS Concepts: • **General and reference** → **Surveys and overviews**; • **Computer systems organization** → *Neural networks*; • **Security and privacy** → *Public key encryption*; • **Software and its engineering** → Software performance.

Additional Key Words and Phrases: Fully Homomorphic Encryption, Privacy-Preserving Machine Learning, Private Inference

ACM Reference Format:
Zaira Pindado, Thomas Spendlhofer, Mohamed Allam, Priyam Mehta, Lena Martens, and Antonio J. Peña. 2025. SoK of Private Deep Neural Network Inference with Approximate Fully Homomorphic Encryption. *ACM Comput. Surv.* 8, 8, Article 8 (September 2025), 40 pages. https://doi.org/XXXXXXX.???????

1 Introduction

Deep learning (DL), implemented by deep neural networks (DNNs), has been a hot topic for more than a decade. DNNs are already solving practical problems previously unsolved, such as face recognition, while others, such as autonomous driving, are presenting clear signs of maturity. In order for a DNN to be able to learn functions that are more complex

Authors' Contact Information: Zaira Pindado, zaira.pindado@bsc.es, Barcelona Supercomputing Center (BSC), Barcelona, Spain; Thomas Spendlhofer, Barcelona Supercomputing Center (BSC), Barcelona, Spain, thomas.spendlhofer@bsc.es; Mohamed Allam, Barcelona Supercomputing Center (BSC), Barcelona, Spain, mohamed.allam@bsc.es; Priyam Mehta, Barcelona Supercomputing Center (BSC), Barcelona, Spain, priyam.mehta@bsc.es; Lena Martens, Barcelona Supercomputing Center (BSC), Barcelona, Spain, lena.martens@bsc.es; Antonio J. Peña, Barcelona Supercomputing Center (BSC), Barcelona, Spain, antonio.pena@bsc.es.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.
Manuscript submitted to ACM

and/or attain higher precision, DNNs are becoming increasingly large. Clear examples are large language models (LLMs): GPT-3 features 175 billion parameters, and GPT-4 is estimated to have broken the trillion-parameter boundary.

There are clear use cases for DNNs where the data provided to the network is sensitive or confidential, such as intellectual property or personal data. In these scenarios, privacy-preserving technologies, including cryptographic primitives such as fully homomorphic encryption (FHE) [56], oblivious transfer [130], or two-party computation [21, 157], are required. At first glance, cryptography, which aims to prevent information leakage, and DL, which aims to extract information from the data, seem to be in direct opposition. However, FHE enables performing computations directly on the encrypted data and obtaining the (encrypted) result of the operation without the need for decryption. The final result may be decrypted using the secret key and, once decrypted, the result is the same as if it had been produced by the traditional method (decryption, computation, and encryption). The owner of the data has control over it, and the DNN may be evaluated in a private fashion.

Research in FHE has seen rapid growth in recent years. Efforts in the field have focused on improving the efficiency of cryptographic operations by algorithmic design [28, 35] and by designing dedicated hardware accelerators [2, 73]. It is now also possible to easily test and build prototype implementations due to the appearance of many open-source FHE software libraries [1, 6, 160]. This rapid development and interest in FHE have made the technology attractive as a tool to potentially provide privacy in many production scenarios, in particular, in DNN inference.

A private DNN inference system constructed with FHE considers a client who owns the data and a server that owns the DNN model, already pre-trained and compatible with homomorphic inference. We assume that key generation and the transfer of evaluation keys from the client to the server are handled outside of the inference protocol. The client encrypts the data using FHE to maintain its privacy and sends it to the server. The server then executes the DNN homomorphically over the ciphertext and returns the result to the client. This solution is considered to be non-interactive, since only the first message with the data and the final message with the inference result are exchanged. An example of a use case in the medical sector is a DNN (held by the server) that is trained to distinguish between cancer and non-cancer samples of MRI scans, and a doctor (the client) who uses it without disclosing any sensitive information from their patients (including their MRI files).

We aim to define a private DNN inference system that captures all the properties that will make it practical and ready to use while maintaining the privacy of the data in a cloud-based execution. Ideally, for this purpose, it must be scalable, compatible with FHE, general, so that any type of data and model may be processed, and support the necessary backends for efficient execution over the available hardware. Constructing such systems is far from an easy task, since practical homomorphic encryption implementations emerged relatively recently and FHE is not exempt from drawbacks that render it currently impractical in many scenarios.

In general, FHE requires the selection of several parameters; ensuring a high security level and sufficient operational depth incurs large memory and runtime overheads. These overheads vary depending on the security parameters, but both computation and memory increase by about 100× with respect to non-encrypted computation [4]. Even when leveraging bootstrapping, a costly procedure that enables operating beyond the initial depth, there is a limit to the number of operations that may be performed. This is because, as the depth of operations increases, the polynomial degree also increases in order to maintain the same level of security. Additionally, there are challenges when we apply FHE in DNNs, where the data is composed of real numbers. Currently, no FHE scheme supports floating-point arithmetic and can efficiently perform computations across all types of layers commonly used in DNN models. Those that are compatible with all types of operations in the layers do not support real numbers and require quantization of the entire network. On the other hand, those that do not require quantization require the use of approximations or

functional bootstrapping, which are computationally expensive. In our work, we focus on the CKKS scheme, because of its suitability for working with real numbers by definition and its SIMD capacity (see Section 4 for a justification); however, we have to deal with the fact that in this scheme only additions and multiplications may be evaluated directly.

Currently, limited RAM sizes and very long inference times make homomorphically encrypted deep learning inference still impractical for many applications. The size of the data increases significantly when encrypted, leading to a considerably higher computation time when compared to the unencrypted execution. Fortunately, many efforts have advanced the field in this aspect since Gentry [56] introduced the first FHE scheme in 2009. These include improved FHE schemes [34, 39], but also software optimizations and hardware specializations that decrease (1) the data footprint and (2) the execution time. On top of that, several efforts have dealt specifically with DL-related issues, ranging from HE-friendly networks to HE inference frameworks/compilers/engines.

This work is a systematization of knowledge that sets a general definition of a *Private Inference Engine operating FHE* in Section 3, a notion advanced upon simpler concepts found in the literature such as FHE/DNN compilers or prediction systems. In contrast, we strive for a general (i.e., DNN-model-oblivious), scalable (i.e., able to execute today's production-ready DNNs), and agile system (i.e., able to execute inference tasks within a latency considered appropriate by the users of the specific use case). Additionally, we perform a thorough analysis of the suitability of FHE when leveraged for privacy in DNN inference, as well as a unique comparison between TFHE and CKKS schemes from the position set in this work in Section 4. Finally, we survey all the works that meet our criteria: being purely FHE-based (excluding combinations with other cryptography techniques, differential privacy, or any other method), using the CKKS scheme, and including bootstrapping, since we are focused on an inference engine that can process deep models. To perform our analysis, we have searched the databases *Google Scholar, arXiv, and IACR ePrint (Cryptology ePrint Archive),* using the keywords *private inference, FHE* and *CKKS*. The search period spans from the beginning of this topic (2016, CryptoNets [58]) to November 2025, when we finished writing this manuscript. We have analyzed all the works that resulted from this search, and the 43 works that passed our filter are those included in our analysis of the state-of-the-practice of CKKS implementations in Section 5 and performance in Section 6. We note that we have not distinguished between peer-reviewed works and preprints. The information on the implementations reported in this work is as reported in the original papers.

In summary, the main contribution of this article is our unique analysis of the techniques and approaches required to construct the described systems, by exploring current challenges along with the state-of-the-art solutions in the literature trying to solve or circumvent them. To this end, we first review techniques to integrate bootstrapping into an NN, function approximation methods, linear layer operations such as convolutions, and matrix multiplication optimization approaches—those already present in implementations, but also future directions guided by the most recently published theoretical works. In addition, we review parallelism techniques to deal, for instance, with data that is larger than the maximum ciphertext size (by splitting it into different ciphertexts, which are processed in parallel). Next, we review hardware acceleration techniques, including GPU, FPGA, and ASIC-based platforms. We end our analysis by comparing the performance of the different solutions that we review—noting a gap in the literature with respect to experimental evaluations focused on scalability and parallel inference—before discussing our conclusions.

2 Background

In this section, we first establish the notation we use in the rest of the manuscript. Next, we provide the necessary background on deep neural networks and cryptography to understand the rest of the work.

2.1 Notation

We express vectors ($\mathbf{v}$), matrices ($\mathbf{M}$) and tensors in bold. Vectors are denoted in lowercase, while for matrices and tensors we leverage capital letters. Given $\mathbf{v}$, $\mathbf{v}_i$ denotes either the i-th scalar component of the vector or the i-th element of an ordered finite set of vectors. For a positive integer q, we denote by $\mathbb{Z}$ the set of integers, by $\mathbb{Z}_q$ the set of integers modulo q, and by $\mathbb{Z}_q^*$ the multiplicative group of integers modulo q. We denote by $\mathbb{R}$ the set of real numbers.

2.2 Deep Neural Networks

A (deep) neural network or model is composed of many layers of interconnected neurons. Given an input tensor $\mathbf{x}$, each layer applies a parametrized function to produce an output that feeds the next layer. In the rest of the paper, we use the terminology employed in the NN literature, where the term linear layer refers to layers with linear operations and multiplications of two ciphertexts (ct-ct). We briefly describe the most common types of layers in the following:

Activation: A non-arithmetic function that decides whether the neuron is activated or not. Some examples are "Rectified linear unit" $\text{ReLU}(x) = \max(0, x)$, "Gaussian Error linear unit" $\text{GELU}(x) = x \cdot \Phi(x)$ where Φ is the cumulative distribution function of the standard normal distribution, and "Sigmoid linear unit" $\text{SiLU}(x) = x \cdot \sigma(x)$, where $\sigma(x)$ is the logistic sigmoid.

Attention: A mechanism to capture relationships and patterns in an input sequence. Self-attention is computed as $\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_K}}\right)\mathbf{V}$, where $\mathbf{Q}, \mathbf{K}$ and $\mathbf{V}$ are the Query, Key and Value matrices and d_K is the dimension of the key vectors.

Batch Normalization: Given a mini-batch of activations $\mathbf{x} \in \mathbb{R}^{b \times d}$, with batch size b and feature size d, this layer normalizes each feature as $\mathbf{y}_i = \gamma \cdot \frac{\mathbf{x}_i - \mu_b}{\sqrt{\sigma_b^2 + \varepsilon}} + \beta$, where μ_b and σ_b are the mean and standard deviation across the batch dimension for each feature, the affine parameters γ, β are learned during training, and ε is a constant for numerical stability.

Layer Normalization: Similar to batch normalization, but the mean μ_d and standard deviation σ_d are calculated for each sample of the batch and across all features d.

Fully Connected Layer: Performs the dot product between the value of the output in the previous layer and the weight of the neuron: $\mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{b}$, where $\mathbf{x}$ is the output of the previous layer, $\mathbf{W}$ the weight matrix, and $\mathbf{b}$ is a bias vector.

Convolutional Layer: Extracts features from the input by sliding a set of C_{out} filters over the input tensor $\mathbf{I} \in \mathbb{R}^{C_{in} \times W_{in} \times H_{in}}$, where C_{in} is the number of input channels and W_{in}, H_{in} refer to the width and height, respectively, and produces an output tensor $\mathbf{O} \in \mathbb{R}^{C_{out} \times W_{out} \times H_{out}}$. The convolution slides a filter bank $\mathbf{F} \in \mathbb{R}^{C_{out} \times C_{in} \times F_w \times F_h}$; at each spatial position, it computes: $\mathbf{O}[c, i, j] = \sum_{k,p,q} \mathbf{I}[k, s \cdot i + p, s \cdot j + q] \cdot \mathbf{F}[c, k, p, q] + \mathbf{b}[c]$, where $\mathbf{b}$ is a bias vector, s is the stride, (p, q) ranges over the filter window, and c, k over the output/input channels respectively.

Pooling: Reduces spatial dimensions by aggregating values over a local window of size $p \times p$ with stride s. Max-pooling computes $\mathbf{O}[c, i, j] = \max_{0 \le r,q < p} \mathbf{I}[c, s \cdot i + r, s \cdot j + q]$ the element-wise maximum, which is not compatible with FHE (see Section 5.2).

A *convolutional neural network* (CNN) is a class of DNNs that contains convolutional layers. A *transformer* is a DNN architecture based on the attention mechanism. Input data is represented as a sequence of *tokens* that are contextualized at each layer, amplifying the signal for tokens that are considered important. Transformers are widely adopted in LLMs for natural language processing (NLP).

2.3 Fully-Homomorphic Encryption

Fully homomorphic encryption (FHE) schemes enable arbitrary computations over encrypted inputs, i.e., any number of additions and multiplications. The security of FHE schemes is based on the hardness of the learning with errors (LWE) problem [108, 132] and its variants, such as ring LWE (RLWE) [143], all related to various well-known hard lattice problems (post-quantum secure [108]). In his thesis, Gentry provided a method called *bootstrapping* [56] for constructing general FHE schemes from those that support only a limited number of operations (leveled HE schemes). Many constructions have been proposed since, with TFHE and CKKS being among the most modern and widely leveraged in the literature related to the focus of this manuscript. TFHE [39], also referred to as CGGI for its authors, is an exact FHE scheme that enables computing any Boolean gate operation or non-linear function over bits homomorphically. Its security is based on a generalized version of the LWE problem: Torus LWE. We refer the interested reader to [40, 78] for further details on the inner workings of TFHE. In this work, we focus on CKKS and discuss this choice when analyzing the suitability for constructing a private inference engine in Section 4.

2.4 The CKKS FHE Scheme

The Cheon-Kim-Kim-Song (CKKS) scheme [36] is an approximate FHE scheme that was introduced in 2017, and it has become one of the most used in practical applications. It was defined as a leveled HE scheme and extended to an FHE scheme in 2018 [34]. The scheme is defined over a cyclotomic ring $\mathcal{R} = \mathbb{Z}[X]/(X^N + 1)$, for a positive integer N using RLWE-based encryption. To ensure security, the encryption procedure adds a random element (the noise or error), following a given error distribution. This noise increases every time a homomorphic operation is performed over a ciphertext. The key difference between CKKS and other schemes such as TFHE, is that the noise is not removed during decryption. In this fashion, the decryption yields $m + e \approx m$, where m is the encrypted message and e is the noise. If e is sufficiently small compared to m, it will only alter the least significant bits of m, enabling us to work with this approximation if we control the precision, i.e., the number of bits that are not affected by e. This design enables CKKS to be natively compatible with approximate homomorphic operations over complex and real numbers, which is an ideal scenario in machine-learning applications, as we discuss in Section 4.

The scheme works with $L + 1$ residue rings $\mathcal{R}_{Q_\ell}$, where $Q_0 = q_0$ is the zero-level modulus. The modulus at level ℓ is defined by $Q_\ell = \prod_{i=0}^{\ell} q_i$, typically $\approx q_0 \Delta^\ell$, where Δ is the scale factor and $q_i \approx \Delta$. The scale factor is used to scale the plaintext, i.e., the encoding of the message, which sets the highest precision possible (precision decreases as each operation increases the error). In the original scheme, all q_i are equal to a single q; in the RNS version, they are distinct primes of similar size. The residue number system (RNS) representation is the integer version of the Chinese Remainder Theorem (CRT) representation, where values in a ring $\mathcal{R}_{Q_L}$ are represented as vectors of values in the rings $\mathcal{R}_{q_0} \times \cdots \times \mathcal{R}_{q_L}$, where q_i are pairwise coprime. In RNS-CKKS, all polynomial multiplications are performed on ring elements in the polynomial CRT representation, where all integer components are represented in the integer CRT basis. Since each encoding includes the scale factor, after multiplying two ciphertexts, the scale factor is squared, and a rescaling operation is needed: to recover the original scale, we divide by q_ℓ, which is $\approx \Delta$. This rescaling changes the level of a ciphertext from ℓ to $\ell - 1$.

The number of levels L is tightly related to the concept of *multiplicative depth*, since rescaling is usually required after each multiplication in CKKS. While techniques like waterline [44] or double [42] rescaling are able to break this tight coupling, in most private DNN implementations, the multiplicative depth is linearly dependent on L. Once a ciphertext has reached level 0 (modulus Q_0), it needs to be refreshed to continue operating. This is achieved via the

bootstrapping procedure, which, in CKKS, extends the modulus back to Q_{L-b} (where b is the number of levels consumed by bootstrapping).

Plaintext encoding allows multiple messages to be packed into a single plaintext (and thus ciphertext), a technique known as *packing*, which enables parallel processing in a *single instruction, multiple data* (SIMD) manner, where a single operation is applied to multiple data elements simultaneously.

3 Private Inference Engines

In this section, we define our notion of an inference engine and explain the challenges when constructing such a system around FHE. Writing secure applications with FHE is unquestionably a challenging task for programmers without a cryptographic background. First, the homomorphic encryption scheme needs to be instantiated with parameters that are not only secure, but also (as much as possible) enable efficient calculations. Second, the cleartext data needs to be packed into ciphertexts that support only a limited number of operations, requiring the whole program to be expressed as a sequence of operations on ciphertexts (i.e., homomorphic primitive operations). On top of that, each ciphertext contains noise that grows with each operation and needs to be managed, creating non-intuitive performance trade-offs.

There have been several efforts aiming to make FHE more accessible by developing tools that provide a high level of abstraction, removing the need for programmers to interact with the ciphertexts directly. In their survey, Viand *et al.* [149] collect such tools under the concept of *compiler*, but they remark that the term is used loosely in the context of FHE and some of them function more as interpreters or libraries. Furthermore, even though many of these compilers include optimizations for packing data into ciphertexts (e.g., [24, 45]), specialized implementations for managing the layout of ciphertext tensors in the context of ML sometimes prefer to use the term *framework* (e.g., [5, 50]) instead, similar to unencrypted ML frameworks like PyTorch or JAX. These traditional (i.e., unencrypted) frameworks are supported by a rich software ecosystem that enables even ML novices to pick a pre-trained model and deploy it (for example, on a cloud service) with just a few lines of code.

In order for private inference to become practical in production applications, we consider it essential for the (encrypted) software stack to attain a similar level of maturity, enabling users to deploy a model without a background in ML or cryptography. In other words, an ideal black-box tool in this scenario would combine the aforementioned FHE functionality with the ease of use of current ML platforms. We use the term *private inference engine* to describe such a (software) tool that accepts an arbitrary (pre-trained) model as an input, converts it to an equivalent FHE version, and runs inference on that network for encrypted inputs (without ever having access to the unencrypted data), ideally as efficiently as possible and automatically applying state-of-the-art optimizations. With regard to FHE, we require this inference engine to support at least the following functionality:

Model Transformation: Convert a cleartext model into the equivalent FHE version, including data layout optimizations and data packing.

Cryptographic Optimization: Determine the optimal parameters and instantiate the underlying FHE schemes as efficiently as possible while maintaining the desired level of security.

Circuit Optimization: Handle noise growth, performing the bootstrapping procedure as appropriate.

Target Deployment: Deploy efficiently on the available hardware (CPU, GPU, FPGA, ASIC, etc.).

It is well known that combining NNs and FHE is challenging. Thus, for constructing a private inference engine that is user-friendly and suitable for use in production, we have to deal with several difficulties:

Non-linear Functions: NNs commonly employ activation functions such as ReLU or operators such as max-pooling that are non-linear, while FHE supports only the evaluation of arithmetic circuits (addition and multiplication). Different approaches may be used to address this problem: (1) client-aided design [107]; (2) approximation by a polynomial function; (3) replacement of non-linear functions with FHE-friendly functions; (4) combination with other cryptographic tools, the most commonly used being multi-party computation (MPC) primitives like secret-sharing, oblivious transfer, or garbled circuits; and (5) functional bootstrapping, a bootstrapping procedure that not only refreshes the ciphertext, but also evaluates a function over the plaintext.

Scalability: Operations on FHE ciphertexts are much more computationally intensive than those on clear data. Thus, scalability for large models and datasets is a primary concern, since encrypted inference may easily last several minutes, hours, or even days, depending on the size of the model. We identify several considerations:

- **Memory:** Ciphertexts, encoded plaintexts (CKKS), and necessary evaluation keys require large amounts of memory (easily on the order of 100 GB) that far exceed those of cleartext models.
- **Parallelism:** While parallel processing of ciphertexts is an option, parallelism might be inhibited by the memory requirements. Among open-source FHE libraries, only OpenFHE supports internal parallelism on CPU.
- **Accelerators:** GPU support for FHE has increasingly become available [42, 52, 80], but for many FHE operations, memory access is the main bottleneck [46]; hence, other hardware architectures such as ASICs [138], FPGAs [3, 134], or processing-in-memory (PiM) [77] have been explored.

Error Management: Ciphertexts in FHE inherently contain an error term, which increases with subsequent operations. Managing that error is important to obtain a valid result; however, there are trade-offs between error and performance:

- **Parameters:** To sustain accurate results over a sequence of further operations, larger parameters are required.
- **Bootstrapping:** A highly time-consuming but essential operation for large models to maintain practical parameters; thus, the number of bootstraps and their placement have significant performance implications.

Security: Selecting the proper parameters for an FHE scheme is non-trivial, with standards [27], tools [9], and formulas [91] providing indications on the parameter values that attain a desired security level. However, these parameters also impact performance; hence, finding a balance between security and computational load is extremely relevant. Another concern is the adversary model, with honest-but-curious (or semi-honest) being the most frequently used; it assumes that the parties behave honestly (i.e., follow the protocol), but they may try to gain information. This model may be improved by adding a proof along with the inference result that verifies the operations that have been performed to produce it (verifiable FHE (vFHE) [92]). While some vFHE solutions do not disclose information about the model, these are limited to theory and rarely adopted in practice. Accordingly, the scope of this review is limited to input data privacy; model privacy and verifiability are not considered.

We note that even though various solutions exist that attempt to tackle the above issues and enable automatic private inference based on NN models or tensor descriptions (e.g., Orion [50], ANT-ACE [101], HELayers [5]), current implementations are still limited to certain architectures or NN layer types. Section 5 provides an in-depth analysis of these existing frameworks/compilers alongside other solutions that have been proposed in the literature.

4 Suitability for Private NN Inference

In this section, we analyze the suitability of FHE for constructing an inference engine. First, we justify our choice of using stand-alone FHE solutions over hybrid solutions, which use FHE in combination with other privacy-preserving technologies. Next, we analyze the suitability of the two most frequently used FHE schemes in private NN inference: TFHE and CKKS. While TFHE features the fastest bootstrapping, CKKS is the most suitable for working with real numbers.

4.1 Purely FHE Solution for Private NN Inference

The main advantages of building private NN inference using FHE over other privacy techniques like two-party computation or oblivious transfer (OT)-based protocols are that FHE offers minimal communication requirements and full privacy over the data without any additional model leakage beyond cleartext inference. OT is typically used as a building block for interactive computation protocols in MPC, and it enables the receiver to obliviously select one of the sending party's inputs. The sender does not learn the receiver's choice, and the receiver does not learn the non-selected inputs. In DNN inference, it is used for the non-linear layers, but it implies additional communication between the server and the client (and non-negligible computations). Secret sharing, an approach for distributing a secret value via two or more shares, which, separately, do not reveal any information about the secret, enables evaluating any arithmetic function; however, when working with NNs, this protocol requires one round of interaction per non-linear layer, which not only adds communication cost but also reveals the number of non-linear layers in the model. Another MPC technique used in DNN inference is garbled circuits, a two-party protocol that aims to compute the value of a publicly known function by two parties over secret inputs. This protocol features a constant number of communication rounds, which is independent of the computed functionality; however, solutions based on garbled circuits leak the structure of the NN, i.e., the number and type of the layers, since these must be known by both parties.

With non-interactivity, we avoid additional communication costs beyond the input from the client and the final output response from the server [158], and also issues such as network latency and high bandwidth usage. Another argument for using an FHE-only solution is that its security relies only on the assumption of the hardness of the LWE problem (or its variants) and the selected parameters, whereas leveraging a hybrid approach implies relying on the assumptions of the other primitives or its requirements for security (for example, the existence of a simulator in garbled circuits) as well.

Still, we need to find the most suitable FHE scheme for our target. Since DNNs require intensive computations, the multiplicative depth of the circuit is considerably high. Although some works perform inference with BFV [30, 32], the highest-performing frameworks leverage either CKKS [50, 79] or TFHE [22, 137], which we analyze next.

4.2 TFHE vs CKKS

There have been previous comparisons between the two schemes for inference, considering relatively small NNs [116]. Others focused on the bootstrapping procedure of different schemes [7]. More recently, Krüger *et al.* [94] claimed that CKKS provides the best efficiency for most operations and applications that can be parallelized, given its high precision and throughput. We use the experimental results of the last two cited papers in our exposition. Additionally, we compare the schemes based on their features and their viability for the task of running large NN inference. We consider performance (time and memory), accuracy, whether the functionalities are available in a public library, and their ease of use (i.e., programmability and ability to use a pre-trained NN).

4.2.1 TFHE. Since TFHE is an exact scheme, there is no accuracy loss when performing FHE operations. A ciphertext represents a single integer, and thus, all the operations are on scalar operands. One of the most important advantages of TFHE is that it **supports non-linear layer operations** through programmable bootstrapping [41] by using an LUT containing the evaluation of an arbitrary function. This functionality maps well to the non-linear functions customarily found in DNNs. Programmable bootstrapping has relatively low latency for small integers [7]. In addition, it does not add overhead with respect to plain bootstrapping, since the LUT is encoded and used in the blind rotation operation (that is already part of the TFHE bootstrapping). All other schemes have to modify or add some step into the bootstrapping procedure to provide this functionality. However, a simple operation like integer multiplication may be costly in TFHE, especially if the multiplication result should not wrap around the current plaintext modulus [7]. Another advantage is the relatively **small modulus**. This leads to relatively small keys [109], and hence relatively low bandwidth required to communicate the evaluation keys from the client to the server, but also relatively low memory utilization during execution. There is **software available**, such as Concrete ML [159], which provides a ready-to-use NN API with the support of a large engineering team along with extensive tutorials and documentation.

On the other hand, one of the main disadvantages of TFHE is its **performance**. TFHE is much slower than CKKS for most operations [94], except for its fast bootstrapping and comparisons. As reported by Krastev *et al.* [93], "CKKS takes 31 µs on average per application-level scalar multiply while the TFHE-rs library takes 2.1 s per 32-bit multiply, over 60,000× slower"; this is almost entirely due to CKKS operations being SIMD. Thus, when the time is averaged over the number of elements in a vector, CKKS is considerably faster than TFHE. Another noteworthy disadvantage is that TFHE **only supports integers of small bit-width**; hence, only fully quantized neural networks are supported, which implies retraining for most NN applications and requires deep knowledge of ML training. In addition, quantization incurs a loss in accuracy compared to the original operation, and it is a time-consuming task that hinders development speed. A previous study concluded that the plaintext space may be efficiently extended to 3–8 bits, whereas extending it to more bits requires doubling the runtime for each additional bit, hence becoming impractical [7].

Recently, TFHE has been extended to be compatible with floating-point arithmetic [23]. In contrast to CKKS, which uses fixed-point arithmetic, floating-point enables higher precision for data within large ranges. However, the programmable bootstrapping is slower, the implementation is not publicly available to date[1], and the performance figures presented are markedly worse compared to those with CKKS or TFHE with integers. For example, in TFHE with floating-point arithmetic, addition and multiplication last 12 s and 82 s, respectively [23] (sequential 64-bit integers), while for integer representation and the same precision, 105 ms for addition and 401 ms for multiplication were reported[2] on similar hardware. Our own experiments using the CKKS Lattigo library range from less than 0.5 ms in a 2^{12}-dimensional ring to 6 ms in a 2^{16}-dimensional ring for the addition of 1,024 scalars, again leveraging a comparable CPU[3].

Recent works offer improvements over the LUTs used for functional bootstrapping. In Tetris [151], the authors estimate (since no CKKS implementation is available) that evaluating a 32-bit sign function with CKKS functional bootstrapping [10] would require 1538.2 s for 2^{16} slots (i.e., ≈23.5 ms amortized per slot), whereas their TFHE-based approach evaluates it in 527.54 ms for a single slot. This shows that TFHE LUT evaluation attains lower latency, while CKKS offers higher throughput.

[1](November 2025)
[2]https://docs.zama.ai/tfhe-rs/get-started/benchmarks
[3]Since we are comparing execution times involving single-threaded server-class CPU executions in terms of orders of magnitude, using exactly the same CPU is not required for this comparison.

4.2.2 CKKS. The CKKS scheme **supports real numbers** natively, since it has the capability of encoding floating-point values by defining a fixed-point arithmetic on them. This maps well to DNN inference, since most NN models use real numbers. Another advantage is performance, mainly due to the **SIMD** nature of the CKKS encoding. CKKS is able to encrypt an entire vector of floating-point numbers (with the precision being configurable via a scaling factor) into a single ciphertext, instead of encrypting each scalar into a separate ciphertext. Any homomorphic operation on this ciphertext will always operate on the full vector (e.g., a multiplication is an element-wise vector multiplication). If the program to evaluate features a large number of SIMD operations, as in NN inference, the cost of these is amortized over the size of the data dimension. Many inference applications process inputs in batches, which adds another SIMD dimension to the computation. The SIMD capability also applies to the bootstrapping procedure, enabling CKKS to bootstrap thousands of numbers at once. Hence, NN inference is well suited to leverage the performance benefits of CKKS. This argument is supported by Krüger *et al.*'s experiments [94] on different homomorphic operations using TFHE and CKKS over $N = 2^{16}$ integers, which showed that CKKS outperforms TFHE in most operations except bootstrapping. There exist several **software frameworks** based on CKKS that are implemented specifically for NN processing, such as Orion [50], which we analyze in Section 5.

One of the disadvantages of CKKS lies in its **programmability**. Given that all operations are defined on full vectors, there are no single-element operations, which makes designing efficient algorithms challenging. For example, to efficiently perform a matrix multiplication, one needs to carefully align the encrypted data through homomorphic rotations, which is a computationally expensive operation that should be used sparingly. Another important disadvantage is that CKKS only **supports arithmetic circuit operations**. Non-linear functions may be approximated using polynomials, but to obtain acceptable accuracy, these need to be of considerably high degree [55] and leverage many multiplications. This reduces the number of available levels, thus requiring bootstrapping, which is computationally demanding in CKKS. However, some recent results improve CKKS bootstrapping, such as SHIP [35], which decreases the number of levels consumed by bootstrapping to 6–7, according to their experiments with the HEaaN framework (which is not open-source). Other works enable arbitrary function evaluation via LUTs [10, 87]. However, only part of the former methodology has been recently released in the OpenFHE implementation (August 2025). Still, the bit-size of the LUTs is relatively small (20 bits maximum), which Dumezy *et al.* [49] increased to 32 bits. On the other hand, CKKS also requires **high memory use**. The ring where CKKS ciphertexts are defined features a large modulus, which results in large evaluation keys and might result in high bandwidth usage when communicating these from client to server. Evaluation keys are also used when performing FHE operations, requiring higher memory utilization than TFHE during execution.

As introduced in Section 2, CKKS is an approximate scheme, which leads to some loss of accuracy when performing FHE operations. Hence, we are not able to recover the exact message that was encrypted, but the bound of the error is controlled and constrained to the least-significant bits. Nevertheless, DNN inference is known to be tolerant to some precision loss. CKKS is able to encrypt all 52 mantissa bits of a double-precision floating-point value [7], which is also the limit in cleartext inference.

4.2.3 Discussion. We summarize the comparison of the RNS variant of CKKS with the TFHE scheme in Table 1, including general functional bootstrapping for CKKS (GFB CKKS) [10] and floating-point arithmetic capabilities in TFHE (Real TFHE) [23]. The clearest argument for working with CKKS instead of TFHE in the context of DNNs is its performance. The SIMD capability enables attaining the best throughput [7]. CKKS also favors scalability. The main weaknesses of CKKS are the limited accuracy due to its approximate nature and high computational complexity [137].

Scheme	SIMD	Real Numbers	Non-linear layers support	Software for NN	Software available
RNS-CKKS	Yes	Yes (fixed-point)	No	Yes	Yes
GFB CKKS	Yes	Yes (fixed-point)	Yes	No	Yes
TFHE	No	No (needs quantization)	Yes	Yes	Yes
Real TFHE	No	Yes (floating-point)	Yes	No*	No

Table 1. Comparison of CKKS and TFHE characteristics for NN inference. *The work provides ReLU and sigmoid only; no NN framework.

In RNS-CKKS, a fresh ciphertext has a precision of $\log_2(\Delta - e)$ bits, which decreases with each operation. The exact precision after bootstrapping depends on the bootstrapping procedure and cryptographic parameters, but can be refined to high precision with an iterative approach [16]. In contrast, the highest precision in GFB CKKS after bootstrapping is limited to 32 bits. In the case of TFHE, the maximum precision is fixed, with 32 bits being reported in [60] (Dec. 2025), while Real TFHE reports 8 bits.

However, NNs tend to be robust against inaccuracies; even if inference incurs some precision loss, we are likely to be able to attain comparable end-to-end accuracy, as shown, for example, by Orion [50], which attains comparable classification accuracy using the CKKS scheme for a variety of models and datasets (see Section 6).

Another important point is that NN training in the clear is done over floating-point values. All currently available FHE schemes except CKKS work with integers. Therefore, when training an NN to perform encrypted inference using the TFHE scheme, it is necessary to first quantize the network. Emerging techniques like *Real TFHE* [23], which enable TFHE to deal with real numbers directly, show performance that is not competitive with that of CKKS.

NNs contain non-linear layers that cannot be directly evaluated using CKKS[4]. So far, these functions have been approximated by a polynomial or replaced by an FHE-friendly operation. These approximations are non-trivial to find for sufficiently high accuracy and might require retraining. However, TFHE only supports small integers and requires quantization of the network. This is non-trivial and implies a loss of accuracy; it also requires retraining or extending the size, which implies higher latency.

In the literature, all papers describing NN inference of large networks (such as ResNet-50) use the CKKS scheme. For example, in Stoian *et al.* [144], a solution for arbitrary NNs with TFHE, the largest NN features 9 layers, while ResNet models used for CKKS may feature 150 layers [19]. As a matter of fact, the literature contains considerably more FHE-NN implementations using CKKS than TFHE, and these works justify this decision by arguing that CKKS is a more natural fit for NNs because of its inherent SIMD capability, supporting our arguments. This includes all the private inference solutions based on LLMs with FHE [136, 162].

It is also worth mentioning existing scheme-switching solutions for which multiple schemes are used [29, 106], where, e.g., TFHE is used for non-linear functions whereas CKKS is leveraged for linear functions [2]. However, scheme-switching requires on the order of N^2 ring operations [35], adding an overhead that is not easily recovered.

5 State of the Practice of CKKS Implementations

In this section, we present existing CKKS-based DNN inference implementations. Early CKKS-based solutions targeted small, shallow models evaluated on the MNIST dataset [74, 116] (28×28 grayscale images) and gradually shifted to more complex tasks such as CIFAR-10/100 (32×32 colored images) and beyond. In the following, we explore the latest

[4]At the time of writing, functional bootstrapping in OpenFHE does not provide the functionality for CKKS ciphertexts; a conversion to BFV is needed first.

CKKS-based implementations of DNNs targeting CIFAR-10 or larger datasets. We group the analysis by the challenges identified in Section 3 and discuss the techniques and strategies used to address them. A summary is provided in Tables [3–13].

5.1 Bootstrapping

Implementations based on leveled HE support only a pre-determined number of sequential multiplications, limiting their usefulness to relatively simple and shallow networks. To the best of our knowledge, the largest models (in terms of layer depth and number of parameters) implemented and run with this approach are SqueezeNet [45] and AlexNet [5], with a multiplicative depth of ~30 levels. Deeper networks and more complex architectures (e.g., transformers) require many more operations/levels and hence rely on the bootstrapping procedure to refresh the levels of a ciphertext. While this procedure is still costly in terms of computing time, the originally proposed method [34] has since been improved with regard to algorithmic time complexity [28, 63], attained accuracy [16, 99], and implementation [80], to enable actual production applications.

Lee *et al.* [97] were the first to make use of bootstrapping in CKKS to implement private inference for a DNN (ResNet-20). Their paper highlights three relevant trade-offs when integrating bootstrapping into an NN:

(1) **Security and Efficiency:** Both bootstrapping latency and failure probability depend directly on the sparsity of the secret key (Hamming weight h), and parameters should be chosen to minimize the risk of failure. However, while a lower h implies a lower failure probability and reduced latency, it also leads to lower security, since some attacks are able to exploit sparse keys [43].
(2) **Levels and Bootstraps:** The number of levels available after bootstrapping decides the number of multiplications that may be executed before another bootstrap is necessary. Because the complexity of all HE operations depends on the levels, having fewer levels means cheaper operations, but it also requires more frequent bootstrapping.
(3) **Complexity and Frequency:** Sparsely packed ciphertexts may be bootstrapped much more efficiently than fully packed ones. Thus, it is beneficial to bootstrap at places where the number of packed values is low, even if the maximum number of operations has not yet been reached. The same argument applies to bootstrapping one or a few ciphertexts versus many.

To resolve these trade-offs, private inference implementations either rely on **manual placement (M)** by domain experts or attempt to determine bootstrap locations **automatically (A)** by treating bootstrap placement as an optimization problem.

5.1.1 Manual Placement (M). The majority of implementations in the literature insert bootstraps manually, either when the multiplicative depth has been depleted, or at other points that are known (or have been experimentally verified) to be favorable in any of the above dimensions. For example, it has been demonstrated that placing bootstraps before activation functions is beneficial, since it enables subsequent operations to be performed at a lower level, leading to a reduction in overall latency [97]. However, identifying good bootstrap locations requires significant expertise in both FHE and NNs or exhaustive experiments.

5.1.2 Automatic Placement (A). As an alternative, several papers attempt to automate bootstrap placement, identifying locations based on a graph representation of the FHE tasks. Even though optimal placement has been shown to be an NP-hard problem [118], heuristics can be employed to find sufficiently good solutions, depending on the desired optimization

target. FHE-Booster [152] uses a score-based heuristic that minimizes either *1)* the total number of bootstraps or *2)* the latency of the program, considering the bootstrapping urgency (i.e., number of remaining operations), the number of unsatisfied bootstrapping segments (i.e., additional bootstraps needed), and the operation slack (i.e., interval between the operations' latest and earliest start times). Other approaches aim to determine points where the dataflow converges (i.e., the network is narrow), by considering the number of output ciphertexts (DaCapo [38]) or *depth boundaries* (Fhelipe [93]). Orion [50] introduces the concept of a *level digraph*, solving a shortest path problem based on a cost model that estimates the latency of each operation depending on the ciphertext level. While both DaCapo and Orion report 37 bootstrap operations for ResNet-20, evaluating just the effectiveness of the proposed placement methods is difficult due to different choices of FHE libraries, parameter settings, and further optimization strategies implemented within the same work.

5.2 Non-linear Operations

Since CKKS does not support non-linear function evaluation, activation functions have to be either approximated, replaced by some HE-friendly function, or evaluated with functional bootstrapping. Because functional bootstrapping for CKKS is a relatively new method, it has not yet been applied to neural networks, and the two prevailing approaches in the literature are *1)* accurate **approximation (APP)** of the non-linear function by a high-degree polynomial and *2)* **replacement (REP)** of the non-linear function by a low-degree polynomial (e.g., the square function [58]). While these two methods sound similar, a key difference between them is the requirement for retraining/fine-tuning. Since the low-degree polynomials in REP differ considerably from the original activation function, the original activations are either replaced before training or require extensive fine-tuning to maintain accuracy. On the other hand, APP mirrors the original function as closely as possible, employing the polynomial approximation only during inference on already pre-trained models.

5.2.1 Function Approximation (APP). While accurate approximation has the advantage of being able to use traditional, pre-trained models directly, the associated high-degree polynomials consume many multiplicative levels, introducing a large computational overhead. Hence, the goal is to find an approximation that is just accurate enough for the task at hand, while minimizing the resulting overhead and level consumption.

According to the Stone–Weierstrass theorem [145], any continuous function may be approximated with arbitrary precision using polynomials over a closed interval, and various approximation techniques exist. In the context of FHE, numerical analysis [31] and Taylor series [162] have been used to fit a function to a set of sampled points while minimizing the mean squared error. However, subsequent experiments revealed that Chebyshev polynomials outperform both of these approaches, since they determine the best fit on the entire interval instead of a small neighborhood around the sample points [66]. This characteristic is especially relevant for DNNs, where the exact values of the inputs are generally unknown. To reduce the number of costly ciphertext-ciphertext multiplications, the resulting polynomials are typically evaluated by the Paterson-Stockmeyer [33] or BSGS [63] algorithm, which makes inserting bootstraps during their evaluation challenging. To reduce the number of bootstraps for high-degree polynomials, Cheon *et al.* [37] propose using composite polynomials, which enable placing bootstraps among consecutive polynomial evaluations. Alternatively, iterative methods such as Newton–Raphson have also been employed (e.g., NEXUS [162]), using a fixed (pre-determined) number of iterations. The remainder of this section provides an overview of commonly employed approximations for specific layers/functions.

Activation Functions. By expressing the sign function as a composite polynomial, ReLU can be approximated with high accuracy using $\text{ReLU}(x) = \frac{1}{2}x\,(1 + \text{sign}(x))$ and three polynomials [96], which is sufficient to conduct private inference on ResNet-20 [97]. Subsequent papers either follow this composite blueprint or directly approximate ReLU as a single Chebyshev polynomial. Chebyshev polynomials are also the method of choice when approximating SiLU or GELU, where $\Phi(x)$ (see Section 2.2) in the latter is either computed directly or via $\tanh(\cdot)$ (e.g., PyTorch) for faster calculations. Approximation strategies then focus either on the $\tanh(x)$ function [105, 136, 163] or the entire GELU [113]. The $\text{softmax}(x) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$ (for each $i = 0, 1, \ldots, n$) function, on the other hand, requires an approximation of both the exponential function and the division operator. Chebyshev polynomials are commonly employed for e^x [97], while iterative methods such as Goldschmidt [59] or Newton-Raphson are used to resolve division. Other approaches employ Taylor series and stabilize the softmax evaluation by subtracting the maximum input value from all terms (i.e., $e^{x-x_{\max}}$) [76, 162], with slight modifications such as estimating $x_{\max}$ by statistical means [105] or refining the Goldschmidt division algorithm to achieve faster convergence [113]. Regardless of these advances, the softmax evaluation remains costly and, in non-transformer architectures, it left to the client in many implementations, since it is usually the last operation in the model.

LayerNorm. Transformer-based networks commonly do not normalize across a batch but employ layer normalization instead. The main challenge lies in the calculation of the reciprocal square root $1/\sqrt{\text{Var}(X)}$, which is either directly approximated by a Chebyshev polynomial [65, 105], the aforementioned iterative algorithms [112, 113, 165], or a combination of both [133]. When employing iterative methods, many implementations [47, 120, 162] use the algorithms proposed by Qu and Xu [129] to improve convergence by determining good initial values. Alternatively, some works [136] avoid the calculation of mean and standard deviation at inference time completely and instead use precomputed values from statistical estimations, thus simplifying computations significantly.

While the approximation techniques discussed so far are able to act as a drop-in replacement at inference time, they still feature a set of hyperparameters that need to be configured in order to attain high accuracy. In particular, the expected input range, which is unknown at inference time, is a major concern, and the majority of implementations use hand-crafted solutions based on observations during cleartext inference/training. As far as we are aware, only Orion [50] attempts to automate this process by adding a calibration step before the actual inference, which serves to estimate these data ranges for each layer in the network based on a set of representative inputs.

Following the described techniques, several DNN architectures (e.g., ResNet and BERT) have been successfully implemented using pre-trained weights [95, 162], attaining accuracy comparable to their non-encrypted counterparts.

5.2.2 Function Replacement (REP). The goal of this method is to design FHE-friendly NNs that replace non-linear functions with low-degree polynomials consuming fewer levels than the high-degree approximations from the previous section. This results in architectures that require less multiplicative depth, and thus the number of levels and/or bootstrap operations may be reduced considerably, significantly improving inference latency. However, when acting as a drop-in replacement, these functions generally yield decreased accuracy and thus require retraining/fine-tuning to attain results that are comparable to the original network.

Replacement strategies can be categorized into *1) constant polynomials*, aiming to determine low-degree polynomials that keep the expressiveness of the original function, or *2) trainable polynomials* that are adapted during training to determine good representations.

Constant Polynomials. A well-known low-degree polynomial replacement for ReLUs is the square function $f(x) = x^2$ [58]. However, especially if the number of layers is large, it suffers from the *escaping activation* problem [55], where

the activation values escape the well-behaved range, leading to either exploding values or poor approximations. Other studies use polynomial regression (for a constant degree D) to minimize the error between the approximation and the target function f (over the range $[-a, a]$). For the ReLU function, the optimal coefficients $w_0, w_1, \ldots, w_D$ may be found by minimizing:

$$\mathrm{E}(w) = \int_{-a}^{a} \left(\sum_{d=0}^{D} w_d x^d - \mathrm{ReLU}(x) \right)^2 dx.$$

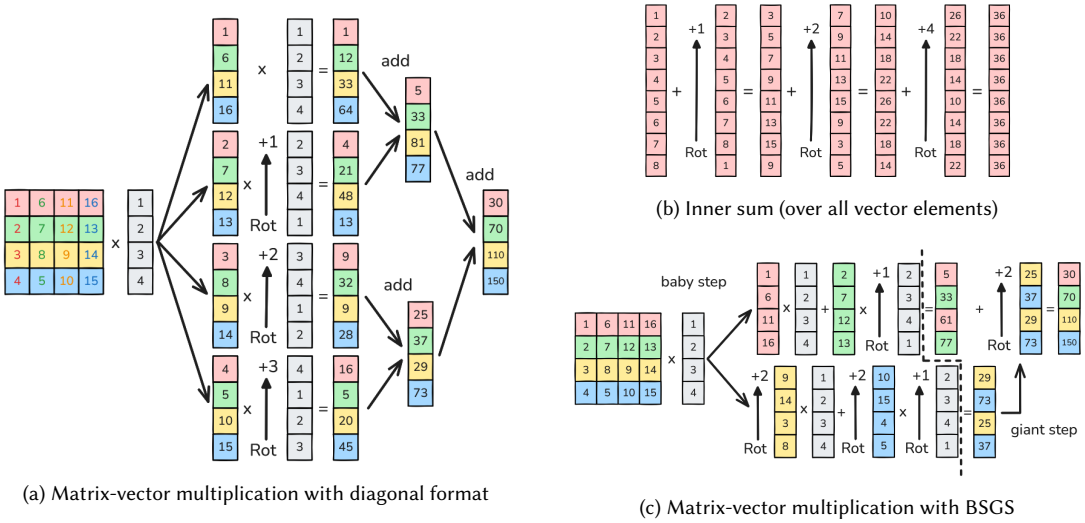
One example of such a ReLU replacement is the quadratic polynomial $f(x) = 0.00047x^2 + 0.5x$ [146]. While these functions may attain higher accuracy than the square approximation, they still suffer from escaping activations. Thus, to mitigate the potentially unbounded range (i.e., the escaping activation problem), there is a need to constrain the pre-activation values by enforcing an upper/lower bound. This may be attained by min–max normalization, where the minimum and maximum values are observed and adapted during training [55]. In a similar fashion, Park *et al.* [122] proposed an accuracy-preserving low-degree polynomial activation function (AESPA), replacing both ReLU and batch normalization layers with the composition of orthogonal basis polynomials and basis-wise batch normalization. The resulting approximation is a quadratic function computed as

$$f(x) = \alpha \sum_{i=0}^{d} \hat{f}_i \frac{h_i(x) - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta$$

where μ and σ are the mean and standard deviation of the batch, $\hat{f}_i$ is the i-th Hermite coefficient of ReLU and $h_i(x)$ is the i-th normalized Hermite polynomial. The reported performance results suggest that AESPA is able to retain the high accuracy of standard models across a number of different networks and datasets. With respect to GELU, common replacements include piecewise polynomials [162] and a modified sign function [120], while softmax simplifications usually aim to eliminate the division operation, removing the need to approximate the inverse and improving numerical stability. Powerformer [120] reduces the division to a constant term using a method similar to batch normalization and substitutes the exponential function with a power-based form. Other works [65, 133] propose employing sigmoid or Gaussian kernel attention instead, both of which require only the exponential function.

Trainable Polynomials. Even with extensive retraining, polynomial activation functions often result in lower accuracy when compared to the original network. To address this issue, Baruch *et al.* [20] propose a fine-tuning approach that uses different polynomials per layer, without necessarily approximating the original function, an approach also known as *trainable* or *parametric* [153] activation. At its core, this trainable activation uses a polynomial of the form $p(x) = ax^2 + bx$, where a and b are coefficients that are trained individually per layer. The provided methodology includes a *smooth-transition* policy that gradually modifies the NN architecture to transition from a non-linear activation to an HE-friendly variant, preventing the aforementioned loss in accuracy. AutoFHE [14] follows a similar approach but allows for polynomials of varying degrees that are adapted during training. Additionally, *knowledge distillation* [67] may be employed to increase the accuracy of HE-friendly models, essentially enabling a knowledge transfer from the stronger "teacher" model (with the original activation function) to a weaker (i.e., HE-friendly) "student" model.

It should also be mentioned that, apart from the activations, function replacement is commonly employed to address other non-linear functions such as max-pooling in CNNs. As noted in CryptoNets [58], the maximum may be approximated with $d = 1$ as a simple sum from the relation $\max(x_1, \ldots, x_n) = \lim_{d\to\infty} \left(\sum_i x_i^d\right)^{1/d}$. Consequently, most encrypted inference works replaced max-pooling with (scaled) average-pooling [126], and when considering only



(a) Matrix-vector multiplication with diagonal format

(b) Inner sum (over all vector elements)

(c) Matrix-vector multiplication with BSGS

Fig. 1. Linear Operations in CKKS.

CKKS and DNNs, this replacement was used by most papers in Tables [3–13] (where the original architecture contained max-pooling).

5.3 Linear Operations

In CKKS, each plain-/ciphertext corresponds to a vector of elements, and the scheme supports element-wise addition and multiplication among these, as well as rotate/shift operations on single vectors. As a consequence, operations that require access to individual elements are not straightforward and must be implemented via reduction/replication of vector elements. For example, the *dot product* between two vectors requires one (element-wise) multiplication followed by a summation of all the elements inside the resulting vector (inner sum). This summation may be computed in $O(\log n)$ operations as a sequence of rotations and additions, as illustrated in Figure 1b. However, since each rotation in CKKS is followed by an expensive key-switching procedure, a major goal of the algorithms for more complex operations discussed below is to keep the number of rotations low.

5.3.1 Matrix-Vector Multiplication. We consider the operation $\mathbf{Ax} = \mathbf{b}$, where $\mathbf{x}$ and $\mathbf{b}$ are encrypted column vectors, and the matrix $\mathbf{A}$ may be encrypted or not. If $\mathbf{A}$ is unencrypted, changes in the representation (e.g., column- or row-major) may be easily computed from the underlying data, while format changes for an encrypted matrix require costly data movement techniques, rendering layout selection a relevant consideration. For example, if $\mathbf{A}$ is represented in row-major format, naive computation of the matrix-vector product requires $O(n)$ multiplications, inner sums, and masked additions. On the other hand, if the matrix is stored in diagonal order (see Figure 1a), the expensive inner sums can be replaced by single rotations.

If the matrix is available in plaintext format, the number of ciphertext rotations may be reduced to $O(\sqrt{n})$ by using the baby-step giant-step (BSGS) algorithm [61]. As depicted in Figure 1c, the algorithm uses computationally inexpensive rotations of the input matrix $\mathbf{A}$ to reduce the number of ciphertext rotations. Additionally, all rotations during the baby step are used multiple times and may be precomputed from the same ciphertext, which enables an optimization

Packing	#Key-Switches
R [74]	$4n + 5\sqrt{n}$
C [120]	$n + 2\sqrt{2n} + 5\sqrt{n}$
D [113]	$\frac{n}{2} + 3 + \frac{n}{4}(\log(n) + 3)$
T [105]	$n + 2\sqrt{2n}$

(a) Multiplication of two ciphertext matrices $\mathbf{A}, \mathbf{B} \in \mathcal{R}_Q^{n \times n}$.

Encoding	#Key-Switches
SE [82]	$2k^2 + 2C - 4$
CE [83]	$C - 1$
CinS [79]	$2\sqrt{C} - 2$

(b) Convolution with C channels and a $k \times k$ filter.

Table 2. Number of key-switches for different packing/encoding techniques.

technique known as *hoisting* [61]. The key idea behind this concept is that certain computations are independent of the rotation offset and may be shared among multiple rotations of the same ciphertext. A similar concept is used by Bossuat *et al.* [28] to apply a second layer of hoisting, leading to a *double-hoisted* approach, which is currently the state of the art in many cryptographic libraries.

5.3.2 Matrix-Matrix Multiplication. The above algorithms may naturally be extended to matrix-matrix multiplication $\mathbf{AB} = \mathbf{C}$, where $\mathbf{A}$, $\mathbf{B}$, or both are encrypted. However, since changing the layout of an already encrypted matrix is non-trivial, algorithms aim to optimize the packing strategy to keep the number of key-switches (induced by rotations and ciphertext-ciphertext multiplications) low. The main strategies can be classified as row-major (R), column-major (C), and diagonal (D) orderings, as well as tricyclic (T) encoding. Table 2a provides an overview of the associated cost for these strategies when multiplying two square ciphertext matrices $\mathbf{A}, \mathbf{B} \in \mathcal{R}_Q^{n \times n}$, assuming they fit within a single ciphertext with n^2 slots.

We note that pure row-major packing results in under-utilization of the available ciphertext slots in typical transformer architectures and is thus usually employed with optimizations that increase ciphertext density [136] or in combination with other formats (e.g., outer product between columns $\mathbf{A}_i$ and rows $\mathbf{B}_i$, such that $\mathbf{AB} = \sum_{i=0}^{n-1} \mathbf{A}_i \otimes \mathbf{B}_i$ [119]). Other optimizations include interpreting each column of $\mathbf{C}$ as a linear combination of the columns of $\mathbf{A}$ multiplied by scaling factors from $\mathbf{B}$ when available as plaintext and using an extract-and-replicate mechanism if $\mathbf{B}$ is a ciphertext [65], or using low-rank adaptation (LoRA) [69] to reduce the size of the matrices. Alternatively, the cost of matrix multiplications can be amortized by processing multiple inputs at once (i.e., batching), combining different packing layouts to optimize throughput [162, 163].

Another line of work aims to benefit from fast linear algebra libraries (e.g., BLAS) to accelerate computations for both plaintext-ciphertext [17] and ciphertext-ciphertext [123] matrix multiplications, reducing these operations to plaintext-plaintext matrix multiplication, similar to the approach of Gentry and Lee [57]. While initial results are promising, these algorithms have not yet been applied in the context of NN inference.

5.3.3 Convolution. The convolution operation (see Figure 2) in CNNs commonly employs zero padding to preserve the spatial dimensions of the input (i.e., $W_{in} = W_{out}$ and $H_{in} = H_{out}$), which we will assume as the default setting in this section. Furthermore, some convolutional layers also specify an additional stride parameter s, which denotes the granularity at which the filters are applied. The algorithms themselves can be classified by their *encoding* strategy, and Table 2b provides an overview of their respective costs for processing a convolution with $C = C_{in} = C_{out}$ channels and a square filter ($k = F_H = F_W$).

Slot Encoding (SE). Using the standard CKKS encoding, Gazelle [82] proposed an algorithm for encrypted convolution that has served as a blueprint for many subsequent works. Their method flattens the input $\mathbf{I}$ into a single ciphertext that

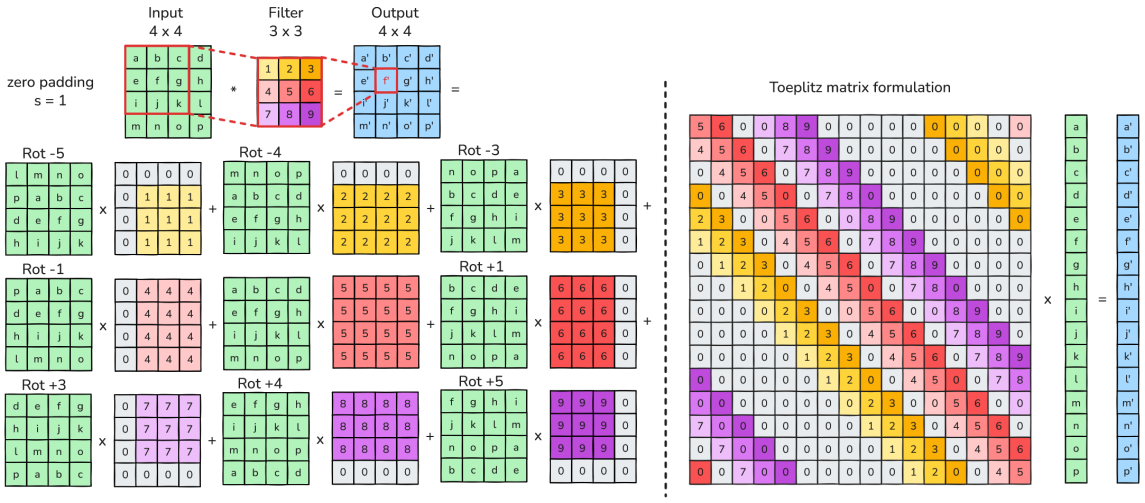


Fig. 2. Convolution with $C_{in} = C_{out} = 1$ (left: Gazelle [82], right: Toeplitz matrix [71]).

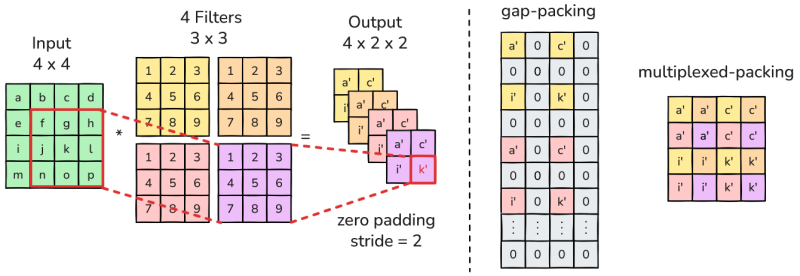


Fig. 3. Gap packing versus multiplexed packing for a convolution with $C_{out} = 4$ and $s = 2$.

is rotated $F_H \cdot F_W$ times, with each rotation being multiplied by a single filter weight encoded into a plaintext. Summing all partial products delivers the final result, as illustrated in Figure 2 for single-input single-output (SISO) convolution ($C_{in} = C_{out} = 1$), and multiple inputs can be convolved at once by concatenating them into a single ciphertext. Strided convolutions ($s > 1$) can be extracted from their non-strided counterparts by introducing an additional masking step [97], but this process introduces gaps (i.e., zero values) inside the ciphertext, occupying slots that could otherwise be leveraged for useful computations. This can be addressed by multiplexed packing [88], mapping a tensor of size $C^2 \times W \times H$ to a *multiplexed tensor* of size $CW \times CH$ (see Figure 3), in combination with masked rotations and additions to recombine several sparse ciphertexts into a single dense ciphertext, thus improving slot utilization.

However, this repacking procedure adds significant overhead, and several works have developed optimized packing strategies and/or algorithms to reduce the number of layout changes. HyPHEN [84] reduces the number of repacking rotations from $O(g^2)$ to $O(\log g^2)$ (for a gap of g slots) by alternating between two convolution operations that each produce a result matching the input format of the other. Other frameworks propose flexible representations, aiming to determine the packing strategy and necessary conversions automatically. Fhelipe [93] supports both row-/column-major layouts and aims to minimize the number of layout conversions, while HELayers [5] features a tile-based representation

and determines the ideal packing based on a specified optimization target (e.g., throughput, latency, or memory consumption).

As an alternative to Gazelle's algorithm, slytHErin [71] expressed convolution as matrix-vector multiplication, representing the filter as a *Toeplitz* matrix (depicted on the right side of Figure 2). Hence, this method is able to benefit directly from optimized matrix-vector algorithms (as discussed in the previous section) without requiring a specialized convolution implementation. Orion [50] extended this approach to multiplexed packing and strided convolutions, developing an algorithm that permutes the rows of the Toeplitz matrix so that it results in a densely packed ciphertext without additional repacking.

Coefficient Encoding (CE). By leveraging the *convolution theorem*, FALCON [103] expresses convolution as the product of two Fourier-transformed inputs, but it requires either interaction with the secret key holder or homomorphic evaluation of the (inverse) discrete Fourier transforms (DFT/IDFT), introducing additional overhead. Later works remove the need for evaluating the DFT with a novel packing method that directly encodes the input into the *coefficients* of a polynomial [70, 83]. In this case, homomorphic multiplication of two inputs in coefficient encoding results in a convolution of the underlying messages. Kim *et al.* [83] showed that if a single-channel input $\mathbf{I} \in \mathbb{Z}^{d\times d}$ and associated filter $\mathbf{F} \in \mathbb{Z}^{k\times k}$ are expressed as two polynomials $I(X)$ and $F(X)$ in $\mathcal{R} = \mathbb{Z}[X]/(X^N + 1)$ with $\max(d^2, k^2) \leq N$, then the $(id + j)$-th coefficient of $I(X) \cdot F(X)$ corresponds to $\mathrm{Conv}(I, F)_{i,j}$ if the coefficients are encoded as follows:

$$I(X) = \sum_{0\leq i,j<d} I_{i,j} \cdot X^{(i-k)d+j} \qquad F(X) = \sum_{0\leq i,j<k} F_{i,j} \cdot X^{dk-(id+j)} \tag{1}$$

where X^t means $-X^{N+t}$ when $t < 0$. Similar to SE, multiple channels can be processed at once, but fewer multiplications/rotations are required to obtain the final result. As a drawback, coefficient encoding does not support the element-wise multiplication needed for evaluating an activation function, and thus it cannot execute a full NN. This is resolved by modifying the bootstrapping procedure—which internally performs Slots-to-Coefficients (StC) and Coefficient-to-Slots (CtoS) conversions—to evaluate convolution in the coefficient domain, right after the StC conversion. Consequently, executing the bootstrap procedure between each convolution and activation function becomes mandatory, possibly leading to an increase in the bootstrapping frequency.

Coefficients-in-Slots (CinS). Even though CE does not require any rotation itself, when processing multiple channels C, the output still consists of C separate ciphertexts that need to be recompressed into a single ciphertext. This overhead can be reduced via CinS encoding, an intermediate representation between the slot and coefficient domains introduced in NeuJeans [79]. In this encoding, for each evenly partitioned slice of the vectors $\mathbf{a} = (a_0, a_1, \ldots, a_k)$ and $\mathbf{b} = (b_0, b_1, \ldots, b_k)$, with a_i and b_i in a subring, multiplication results in a partial local convolution $\langle \mathbf{a} \rangle_{CinS} \cdot \langle \mathbf{b} \rangle_{CinS} = \langle a *_{local} b \rangle_{CinS} = \langle a_0 *_{part} b_0, \cdots, a_k *_{part} b_k \rangle_{CinS}$, where $*_{part}$ denotes the convolution of two polynomials within a subring of the given polynomial ring. Since CinS encoding supports cyclic rotations of the sequence of slices, it can combine multiple ciphertexts more efficiently. Nonetheless, element-wise multiplication is not supported and bootstrapping is still required to enable evaluation of the activation function.

5.4 Parallelism and Hardware Acceleration

There are many forms of parallelism in private inference, depending on the granularity at which it is applied. At the coefficient level, multiple coefficients of the same polynomial may be processed concurrently, while the independent polynomials corresponding to each level of a ciphertext (often referred to as *limbs*) may also be scheduled in parallel. At the application level, one may attempt to pack multiple inputs into a single ciphertext to be processed in parallel or

execute operations on many concurrent ciphertexts. While the former is a popular strategy to increase throughput via batching for small inputs (e.g., MNIST), it is mostly obsolete for larger datasets that exceed the size of a single ciphertext (e.g., a single ImageNet input would already require a ciphertext with more than 2^{17} slots). In those cases, the input needs to be split across several ciphertexts, naturally increasing concurrency. However, even for single-ciphertext inputs, the matrix multiplication and convolution algorithms discussed in Section 5.3 typically create multiple temporary ciphertexts that may be processed in parallel.

A core component to reduce algorithmic complexity in CKKS is the Number Theoretic Transform (NTT) and its inverse, lowering the cost of polynomial multiplication from $O(N^2)$ to $O(N \log N)$. The NTT operation itself, however, is difficult to accelerate due to high computational complexity, extremely intensive memory access, and limited generality [161]. This is especially true for the ciphertext maintenance operations (bootstrapping and key-switching), which require a large number of these transformations; this is commonly addressed by exploiting limb- or coefficient-level parallelism. While both work reasonably well for operations such as polynomial addition and multiplication, the situation is more complex when considering the key-switch operation. Key-switching requires both basis conversion and NTT operations: while NTT generally needs all the coefficients of a single polynomial (and thus lends itself to limb-level parallelism), basis conversion may be parallelized over the coefficients, but requires access to data from multiple limbs. Consequently, many FHE accelerators employ either one or a combination of both of these strategies in order to speed up core FHE operations. This section focuses on discussing such issues on different hardware platforms. Tables [3–13] provide a summary of some relevant contributions.

5.4.1 CPU-Based Acceleration. Only a few CPU-based acceleration schemes have been proposed, and they are somewhat limited in their effect, since the CPU has the fewest parallel computing resources among all discussed platforms. HEXL[5] [26] implements polynomial modular multiplication and NTT using the Intel Advanced Vector Extensions 512 (AVX-512), through loop optimization and vectorization to process eight polynomial coefficients simultaneously. While it is able to attain high acceleration when running on a single thread, the overall performance degrades when many threads are used due to heat dissipation constraints [161].

As a consequence, many CPU-based implementations are restricted to single-threaded evaluation [83, 93]. Other papers only briefly mention the number of CPU threads [84, 110], without discussing efficiency/scalability, or limit their evaluation to shallow models without the need for bootstrapping [44, 164]. The only exception is Park *et al.* [121], who optimize private inference on ResNet-20 for a multi-threaded environment and report a 12× acceleration over the serial version when running on 64 threads.

5.4.2 GPU-Based Acceleration. While the main acceleration capabilities of GPUs are based on the large number of parallel computing resources available, they also feature a higher memory bandwidth than traditional CPUs. Hence, assuming appropriate data storage strategies, even memory-bound kernels may see acceleration due to faster data access. However, GPUs have limited customization flexibility, constraining the innovation of GPU-based implementations when compared to FPGAs or ASICs. Full-scheme implementations of CKKS on GPUs typically focus on accelerating specific, time-consuming operations such as multiplication [81, 140] or bootstrapping [80]. While the paper by Jung *et al.* [81] is based on exposing additional parallelism, optimizations in other works are mostly memory-centric and GPU-specific, due to the high data-access and low arithmetic density of many FHE functions, resulting in memory-bound algorithms. The reported speedups in these studies range from 2× to over 100×, depending on the homomorphic function and

[5] https://github.com/IntelLabs/hexl

optimization method, and a number of GPU libraries (e.g., HEonGPU [117] and Cheddar [42]) have been released, including further optimizations. In particular, both TensorFHE [53] and WarpDrive [52] include optimizations to leverage tensor cores on modern NVIDIA architectures. Nonetheless, many of these solutions limit their evaluation to the ResNet-20 architecture, and there is no extensive performance comparison across larger networks on GPUs.

Additionally, it is worth mentioning that the HEaaN[6] library, while not being open-source, is available as a binary package and offers support for both CPUs and GPUs. Although the extent of its optimizations is unknown, the library has been used in several studies evaluating private inference on GPUs [19, 79].

5.4.3 ASIC-Based Acceleration. In order to deal with compute- and memory-intensive FHE operations, recent ASIC accelerators for CKKS employ hardware–software co-design, wide vector custom functional units, and very large scratchpad or register files. While early ASICs targeted leveled HE, F1 [54] proposed the first *programmable* FHE accelerator capable of running complete FHE programs via wide-vector custom functional units designed for FHE kernels such as NTT and rotation, paired with a custom compiler that enables static scheduling of data movement across the memory hierarchy. However, for high polynomial degrees and large multiplicative depth, this design is bottlenecked by bandwidth and on-chip memory capacity. CraterLake [138] addresses these concerns through a novel key-switching algorithm, which reduces the amount of auxiliary data from GBs to tens of MBs. Subsequent ASICs, BTS [90] and ARK [86], both focus on accelerating the bootstrapping procedure by exploiting coefficient-level parallelism or using *minimum key-switching* and *on-the-fly* limb extension, to eliminate ~88% of off-chip memory reads.

SHARP [85] exploits the fact that in RNS-CKKS, large integer values are broken into smaller parts (usually 64 or 32 bits), implementing a short-word (36-bit) hierarchical microarchitecture that localizes NTT communication by introducing novel data-ordering and short-word functional units. Thus, it is able to reduce the required on-chip memory capacity and power consumption substantially while still retaining performance. On the other hand, Osiris [51] uses a systolic approach to exploit dataflow maps of the BSGS algorithm and double-hoisted rotations to achieve state-of-the-art results for both bootstrapping and encrypted inference.

Across all these designs, the main limitation stems from the required data movement (e.g., ciphertexts, evaluation keys, etc.). To mitigate this, ASICs generally propose large on-chip memory and a massive number of functional units, resulting in large monolithic chips. However, fabrication of such designs is slightly problematic due to the high production cost. To overcome this, Cinnamon [73] and CiFHER [89] propose cluster- and chiplet-based accelerators respectively, which provide better cost-efficiency and are highly scalable. In addition, Cinnamon combines its design with a compiler that is able to automatically extract coefficient- and limb-level parallelism from an application.

5.4.4 FPGA-Based Acceleration. While GPUs are widely available and offer massive parallelism, their architecture is primarily optimized for floating-point operations, whereas FHE computations require high-throughput modular integer arithmetic. Moreover, the irregular memory access patterns and fine-grained operations in FHE are a poor match for typical GPU execution models, resulting in under-utilization of available resources. ASICs, on the other hand, may provide high performance and energy efficiency through tightly customized pipelines and memory hierarchies; however, the development of ASICs incurs remarkably high engineering costs and long design cycles, hindering their suitability for rapid prototyping of evolving cryptographic standards.

FPGAs offer a compelling middle ground. Their reconfigurable logic fabric enables the implementation of customized data paths tailored to modular arithmetic, while offering significantly lower development costs and turnaround times

[6] https://www.cryptolab.co.kr/en/products-en/heaan-he/

compared to ASICs. Recent FPGA platforms equipped with high-bandwidth memory (HBM) and relatively large on-chip memory, such as the Xilinx Alveo U280 and V80, have addressed earlier bandwidth limitations, enabling FPGAs to effectively compete with both GPUs and CPUs in memory-bound workloads such as FHE.

HEAX [134] is one of the first works to target CKKS acceleration on FPGAs, implementing a high-throughput NTT engine and a homomorphic multiplication unit, but it does not support bootstrapping. FAB [3], on the other hand, supports the complete CKKS pipeline and evaluates a multi-FPGA setting for private inference. Poseidon [156] also supports the full scheme and implements several hardware and bandwidth optimizations to improve efficiency. On the other hand, HEAP [2] employs a hybrid scheme-switching approach to accelerate inference in both single- and multi-FPGA configurations, leveraging the TFHE [39] scheme for efficient bootstrapping while utilizing CKKS for the remaining operations. FAST [155] proposes datapath optimizations to reduce off-chip ciphertext access for two key operations in bootstrapping: homomorphic linear transformation and polynomial evaluation. Additionally, it employs a permutation network to efficiently reorder coefficients during NTT stages and rotation operations.

It is noteworthy that all these works propose implementations with significant resource utilization on multi-die FPGAs; however, placement and routing details, including floorplanning, inter-die crossings, and timing closure, are often not reported.

5.4.5 Memory Technologies. To alleviate memory bandwidth bottlenecks, several studies have proposed implementing NTT on near-memory architectures, mostly based on either Resistive RAM (ReRAM) or SRAM array architectures, not targeting commercially available hardware [100]. There are also works targeting the only commercially available processing-in-memory solution to date (UPMEM PIM) [77, 114]; these works, however, do not provide 128-bit security and/or end-to-end execution timings, which a more recent work addresses [18].

Considering that the large memory requirements of encrypted inference have been a common limitation among all of the discussed acceleration approaches, Lloret-Talavera *et al.* [107] proposed leveraging hybrid memory systems of DRAM and persistent memory for private inference on DNNs. They demonstrated that in such settings, data may be efficiently cached inside the DRAM, reducing the required size by 66%, while only incurring a small performance penalty.

5.5 Inference Engines and Compilers

Among the techniques discussed in the previous sections, automatic bootstrap placement (A) and accurate approximation of non-linear functions—via polynomials (APP) or functional bootstrapping—are essential for an inference engine. The former is necessary to support DNNs consisting of many layers, while the latter enables the usage of pre-trained models from repositories such as Hugging Face or PyTorch Hub. Despite the fact that retraining HE-friendly networks with REP is generally considered out of the scope of an inference engine, such an engine would still be able to run private inference on such models by supporting polynomial activation functions. We note that there are no requirements on specific packing/encoding strategies as long as common linear layers are supported. Even though an inference engine should strive to match the state-of-the-art set by handcrafted implementations, it distinguishes itself by being applicable across a wide range of architectures, even if it is not perfectly optimized for each individual model.

Another essential feature of an inference engine is the automatic conversion of model descriptions emitted by popular frameworks such as PyTorch or JAX into corresponding FHE variants. In the literature, methods that enable such transformations between unencrypted and equivalent FHE programs are generally known as FHE compilers. Research on such compilers can be roughly divided into two categories: *1) general compilers* that often focus on lower-level

optimizations and smaller workloads (e.g., ELASM [98], HECO [148]) and *2) domain-specific compilers* targeting specific applications such as NNs. Even though somewhat similar, the concepts of NN-specific compilers and inference engines are different, and only a subset of them qualifies as an inference engine. Many of these compilers (e.g., CHET [45], EVA [44], nGraph-HE2 [25]) support neither bootstrapping nor function approximation, and we refer to Viand *et al.* [149] for more details on these works. Compilers that have been applied to DNNs (Fhelipe [93], Orion [50], and ANT-ACE [101]) are included in our comparison in Tables [3–13]. These inference engines generally convert the NN graph into an FHE variant using the techniques described in this section and often omit lower-level optimizations such as waterline rescaling [44]. Even though these should (theoretically) suffice to support most standard models, experiments are usually restricted to a limited number of architectures that often belong to the same model family (e.g., ResNet).

6 Performance

Many dimensions can be used to evaluate the performance of private NN inference, including accuracy, latency, security, throughput, memory requirements, and energy efficiency. While latency (time-to-inference) is a primary concern in most papers, due to the additional overhead of homomorphic operations, it needs to be emphasized that there are trade-offs among accuracy, latency, and security. By definition, CKKS works on approximate numbers, and each operation introduces a (small) additional error that accumulates over many operations. Hence, a considerable amount of time in actual implementations is dedicated to error management techniques, restricting the error to a tolerable level. The definition of "tolerable" naturally depends on the application, and some homomorphic operations (e.g., bootstrapping) represent a compromise between speed and accuracy. A less accurate bootstrap operation may be computed faster, but also incurs larger error, and even though NNs are known to be somewhat robust to reduced precision, there is no guarantee that a larger error will not change the inference outcome. Therefore, we consider it essential that authors report the accuracy of their encrypted inference in addition to the latency, especially if the reported latency indicates that this may be attained within a reasonable amount of time; otherwise, it is difficult to assess the significance of reported achievements, since these may arrive at the cost of reduced accuracy.

A similar argument may be made with regard to the security parameters. More secure parameters are generally associated with a higher computational cost, and thus weakening the security requirements may lead to faster inference times. The specific parameters that may be considered sufficient are again application-dependent, but 128-bit security seems to be established as the gold standard that most works in private inference aim to attain. Even though there have been attempts to standardize parameters across security levels [27], these standards should not be considered as something static, since the recommended parameters will change when new attack methods are developed. Despite this, not all papers in Tables [3–13] that claim 128-bit security cite the corresponding source (i.e., the method to determine the security level), and some do not even disclose their encryption parameters. Thus, in some cases, it is impossible to determine whether the 128-bit security claim (in those cases in which this feature is mentioned) considers certain attacks (e.g., attacks benefiting from sparse keys [43]) or to re-evaluate their settings in light of new attacks.

Furthermore, when comparing inference times (i.e., latency/throughput), it needs to be considered that the experimental setups (i.e., available hardware) differ among studies and may well contribute to reported differences in performance. Regardless of these impediments, we attempt to evaluate the performance of current private inference applications in terms of four aspects that we consider critical in order for private inference to become practical.

6.1 Latency/Throughput

Despite the fact that sparse ciphertexts may be bootstrapped more efficiently than their dense counterparts, packing the ciphertexts as densely as possible (thus reducing their overall number) seems to be beneficial in terms of latency. This is demonstrated by multiplexed packing [95], attaining ~4.7× faster ResNet-20 inference on a single thread compared to previous work [97] using one thread and one (sparse) ciphertext per image channel (up to 64 threads in total). Subsequent works have built on this concept and reduced latency even further via novel packing strategies that minimize data layout rearrangements.

For ResNet-20 (the most commonly implemented network for the CIFAR-10 dataset), CPU inference time on a single thread has been reduced to less than 6 minutes for classical CKKS slot encoding [135]. When considering multi-threading on CPU, only HEaaN and OpenFHE support parallelization of their homomorphic operations, and HEaaN currently seems to have the advantage in both the number of published works and the reported latency. With respect to the effectiveness of library-supported parallelism, Park *et al.* [121] note that the ResNet-20 implementation of Lee *et al.* [95] may complete inference in three minutes when executed on 64 threads. Further improving upon this, by replacing the activation function with AESPA and designing specialized convolution operations to minimize data rearrangements, HyPHEN [84] attains the lowest ResNet-20 inference time reported on CPU of 37.6 s.

On GPUs, latency may be reduced even further, with reported inference times being as low as 0.82 s for highly optimized implementations [42]. Even though works on other hardware architectures rarely evaluate applications other than ResNet-20 on CIFAR-10, these seem to hold a significant advantage in terms of performance. HEAP [2] is able to conduct inference in less than 0.27 s on FPGAs, and using ASIC simulations, SHARP [85], Osiris [51], and Cinnamon [73] report latency below 0.1 s.

Considering other NNs for the CIFAR-10 dataset (e.g., VGG-16), we observe a similar scenario, with the lowest latency being reported for the combination of HEaaN and GPU. For more complex datasets such as ImageNet, the most commonly implemented architectures are ResNet-18 and ResNet-50, and reported latencies range from almost 9 hours (CPU, single-threaded) down to 5.35 s on GPU. HyPHEN [84] shows the best performance on CPU (multi-threading, AESPA), but coefficient encoding also seems to be advantageous (although Kim & Guyot [83] only encrypt the final eight layers of their network). However, the lowest GPU latencies of 5.35 s (with AESPA) and 14.11 s (with APP) are attained by the CinS encoding implemented in NeuJeans [79].

For transformer-based networks commonly employed in NLP, latencies range from less than two seconds to several hours, depending on the employed model and hardware architecture. Out of nine implementations, five employ GPUs to speed up inference, since pure CPU implementations seem to be far from practical. For example, Tricycle [105] reports a latency of over 5 min for BERT-tiny, and MOAI [163]—which is optimized for throughput—attains an amortized runtime of close to 10 min (with a batch size of 256) for BERT on CPU. Similarly, Encrypted-LLM [47] notes that, when switching to GPU, a speedup of ~200× is observed, reducing inference time to ~70 s for GPT-2. Despite their improvements, most works still report a latency of several minutes for transformer-based inference, even when using GPUs. Currently, specialized FHE accelerators seem to be the only means to reduce inference time even further, since Cinnamon [73] (the only ASIC tested for NLP) reports a latency of 1.67 s.

In addition to latency, a few works also consider the amortized runtime for multiple inputs in the case of batched inference. For instance, since the ResNet-20 implementation of Lee et al. [95] only uses a single thread, they suggest using multiple threads (one per input) for batched inference. Using 50 threads, they report a total latency of 3973 s, leading to an amortized latency of 79 s per image. Applying the same method to ResNet-32, they achieve an amortized latency

of 127 s for the CIFAR-100 dataset. HELayers [5] proposes optimizing the data layout considering different trade-offs, including batched versus single image processing. On SqueezeNet and CIFAR-10, they are able to achieve a speedup of almost 10× in amortized latency when comparing their batched optimizations to the single-input ones. Consequently, some implementations optimize their data layout and algorithms specifically for batched inference [53, 65, 162, 163] and thus Tables [3–13] report the amortized latency in those cases.

6.2 Accuracy

Besides execution efficiency, the ability of a model to solve a certain problem accurately is equally important. For classification tasks, this problem-solving capability is usually measured via the accuracy (i.e., the fraction of correct classifications) for a certain dataset. Here, private inference via HE faces two challenges: *1)* the cleartext model needs to be changed to facilitate homomorphic evaluation (e.g., activation functions, average pooling) and *2)* the additional error added during encryption, both of which might lead to a reduction in prediction accuracy. To address these issues and facilitate comparison, Podschwadt *et al.* [126] recommend that authors report the accuracy of the original model (i.e., without any changes) as well as the cleartext and encrypted accuracy on the adapted model. Unfortunately, most papers report only the last two, which makes comparison more difficult, since both model changes and encryption parameters may well vary among different studies. Nonetheless, this section will attempt to provide a summary of the state-of-the-art for private inference accuracy, ordered by datasets.

For the CIFAR-10 dataset, the original paper [64] reports a ResNet-20 accuracy of 91.25%, a number that is matched by most encrypted implementations. Orion [50] even reports a slightly higher number of 93.4%. Generally, most studies note that their encrypted inference results closely match the accuracy of the (adapted) cleartext model. Both APP and REP seem to work equally well, since there are no notable differences. In terms of other models, ResNet-9 [110], ResNet-44 [84], and VGG-16 [50] are reported to attain more than 94% accuracy, and the highest accuracy of 98.3% is attained by ResNet-50 as implemented by Maloney *et al.* [110]. To the best of our knowledge, the state-of-the-art for this dataset is around 99.5%, attained by a transformer architecture [48].

When leveraging the CIFAR-100 dataset, significantly fewer results are reported. Lee *et al.* [95] note an accuracy of 69.43% for ResNet-32, Kim & Guyot [83] list 64.08% for ResNet-20, and the top accuracy is again reported by Maloney *et al.* [110], attaining 75.3% with a ResNet-9 architecture. This seems to be in line with unmodified versions of these architectures; it is, however, quite far from the state of the art, which reaches more than 90% accuracy, even with convolution-based architectures [147]. One reason for this gap could be that, while CIFAR-100 increases the number of classes when compared to CIFAR-10, the dimensionality of the input remains unchanged, enabling private inference on both datasets with minimal changes to the network architecture. Thus, there seems to be less incentive to develop encrypted versions of networks specializing in CIFAR-100; many papers turn straight to ImageNet instead, which features both more classes and larger inputs.

The larger input size ($224 \times 224 \times 3$) and demand for deeper/larger networks increase the computational complexity, and single-threaded results for ImageNet are exceedingly rare. Kim & Guyot [83] report an inference time of less than 5 minutes, but they evaluate only the final eight layers of ResNet-18 homomorphically, while the previous layers are implemented in cleartext. Orion [50] executes encrypted inference on the whole network, but due to the increased latency (more than 3 h per image), no accuracy results are provided. Using a multi-threaded/GPU backend instead, HyPHEN [84] reports accuracy of 65.25% for the same network, while NeuJeans [79] manages to increase the accuracy up to 74.1% using ResNet-50. This number is matched by Maloney *et al.* [110], but when upscaling the images to $256 \times 256 \times 3$, they report a final accuracy of 80.2%. We note that the discussion of ImageNet accuracy is limited to top-1

accuracy, since it is the most commonly reported number, even though a few works (e.g., Kim & Guyot [83]) report top-5 accuracy instead.

To measure the accuracy of NLP tasks, most works employ the GLUE benchmark [150], which consists of several datasets/tasks including sentiment analysis, question answering, and textual entailment. Most studies report private inference accuracies that are relatively close to the accuracies of the baseline model (usually within ±1%).

6.3 Memory

Actual memory consumption during private inference is only reported in a few papers, and if no values are provided, Tables [3–13] list the available memory as an upper bound instead. Apart from the maximal ciphertext level (with a lower level needing less memory), there are two main components that may be tuned in order to decrease the memory demand: *1)* the evaluation keys and *2)* the plaintext weights. While the size of a single evaluation key is dictated by the homomorphic parameters (and largely depends on the maximal level), the number of required keys can be reduced at the cost of executing additional rotations. For example, any rotational offset could be attained by consecutively rotating the same ciphertext by a single slot (which would require one key only). As an indication, HyPHEN [84] notes that, when extending the implementation by Lee *et al.* [95] for ImageNet, a single evaluation key occupies more than 1 GB of memory. In contrast, with their choice of parameters (and a lower level due to REP), a single key only needs 168 MB. In total, this reduces the memory requirements for all evaluation keys for ResNet-18 from 69.7 GB to 11.1 GB. On the other hand, Rovida & Leporati [135] pursue a different strategy, reducing the number of rotation keys needed during the convolution to a constant number (five). This enables them to conduct private inference on ResNet-20 with CIFAR-10 using less than 16 GB of memory in total.

The size of the plaintext weights, on the other hand, inherently depends on the level at which the corresponding operation (i.e., the operation that applies the weights) is executed. The lower the level, the less memory is needed, which suggests that it may be possible to trade memory consumption for more frequent bootstrapping. Even though HELayers [5] considered optimizing data layouts according to memory requirements, to the best of our knowledge, none of the automated bootstrap placement strategies proposed so far has focused on optimizing memory. Especially for CNNs, the size of the plaintext weights is important, since common implementations (see Section 5.3.3) replicate each filter weight according to the size of the input. While this might be practical for small input data, it leads to an explosion of memory requirements for larger data such as ImageNet. During their attempt to enable private ImageNet inference on GPU, HyPHEN [84] reports that in common implementations, the filter plaintexts consume 285–365 GB of memory. Consequently, they propose an optimization called PRCR ("plaintext size reduction through the image to ciphertext rearrangement"), which essentially modifies the data layout to split an image into multiple segments. Hence, filter weights only need to match the segment size, reducing the plaintext memory footprint to 45.6 GB.

Transformer-based architectures do not suffer from this weight replication problem, and thus the evaluation keys remain the main memory constraint. Consequently, an A100 GPU with 80 GB of memory (which is commonly employed in these works) does not experience any problems fitting the whole CKKS circuit, and most papers do not mention the actual memory consumption. One exception is the study by Rovida & Leporati [136], who report a memory requirement of 18 GB for their implementation of the BERT-tiny model.

6.4 Datasets

Starting from early private inference solutions such as CryptoNets [58] (which classified handwritten digits from the MNIST dataset), considerable research has focused on improving encrypted image classification. In the context of DNNs,

the field has moved to larger and more difficult datasets, such as CIFAR and ImageNet. In the following, we discuss the unique challenges associated with each dataset and summarize the fastest implementation for each. For CIFAR-10, the fastest result on CPU (Table 3) is HyPHEN with 37.6 s, while on GPU (Table 8) Cheddar achieves 0.82 s and HyPHEN 1.4 s, both with ResNet-20 and similar accuracy. Using FPGA (Table 12), the fastest implementation is HEAP at 0.27 s, while with ASIC (Table 13) the fastest is Osiris at 0.069 s, both also with ResNet-20. For CIFAR-100, the lowest CPU latency is 368 s, reported by Kim & Guyot [83] for Plain-20 (Table 4); no GPU results have been reported for this dataset. For ImageNet, Kim & Guyot [83] also report the lowest CPU latency, 292 s for Plain-18 (Table 5), although only the final eight layers of the network are encrypted. On GPU, NeuJeans [79] attains the lowest latency for ResNet-18, with 14.11 s, followed by HyPHEN [84] and Park *et al.* [121] with 14.7 s (Table 9).

While the trend seems to evolve toward larger models that are more powerful in their capabilities, depending on the application, this might not be necessary. For example, CryptoFace [13] has shown that by adopting a tiling approach and developing a specialized network dedicated to face recognition, more efficient solutions may be found to conduct private inference on datasets such as LFW and AgeDB, among others (Table 6).

Another area that has gained popularity with the rise of LLMs is private inference in NLP. This field contains a variety of tasks such as text classification, information retrieval, spam detection, sentiment analysis, and question answering, among others. For text classification on encrypted data, recent approaches, based on word embeddings, usually apply two types of models: one model to obtain a vector representation of the text (i.e., the embedding) and another model that performs the required task (e.g., classification) on the vector data. In the context of private inference, the first type of model is generally processed on the client side, and only the second model works on the encrypted data. Early implementations included recurrent NNs [125] such as long short-term memory (LSTM) (Table 10), and gated recurrent unit (GRU) [72], and reviews of these methods are provided in several surveys [142, 158]. However, following the release and success of ChatGPT, transformer-based architectures have seen widespread adoption in NLP. Transformers present some unique challenges for private inference, such as the attention module, which requires three matrix multiplications and one softmax evaluation, both of which consume many multiplicative levels. For further discussion on the challenges of private transformer inference, we recommend the survey by Li *et al.* [104], whose (non-hybrid) results are extended in Tables 7 and 11. We observe that on CPU the best result is Tricycle with 322 s for SST and the BERT-tiny model, while for RTE it is MOAI using the full BERT model with 574.6 s; for SST-2, Rovida & Leporati achieve around 375 s with BERT-tiny and report an accuracy of 79%, while NEXUS attains higher accuracy (94.46%) with the Llama-3-8B model in 1088 s. The fastest implementation of BERT on GPU is reported by Rho *et al.* with a latency of 26.5 s.

7 Related Work

In this section, we provide an overview of previous surveys and review papers on privacy-preserving machine learning (PPML). We analyze papers that discuss PPML solutions using HE or contrast them with other privacy-preserving technologies with the goal of achieving private inference. In Section 4, we justify why we believe that using a purely FHE solution is the most suitable option for a practical inference engine, which motivates the focus of this work.

Combining HE and DNNs is not an easy task, and many surveys presenting DNN solutions with HE explore the challenges of constructing such approaches along with their limitations and solutions to address them. For example, Pulido-Gaytan *et al.* [127] identified challenges not only in the computational design of the homomorphic processing of inner network functions, but also in data and error manipulation. They concluded that the objectives for HE-based NN inference are threefold: high accuracy, high data security, and low computational complexity. Their work analyzed early

HE-based networks such as CryptoNets [58] and LoLa [30]. Along the same lines, Podschwadt *et al.* [126] reviewed the challenges of implementing DNN architectures with HE, but restricted their scope to well-known publications that are practical, and often tied to released software. They highlighted the difficulties of evaluating NNs under HE, such as computational, memory, and latency overheads, in addition to incompatible operations and limited network depth, while also including an overview of different FHE schemes and their advantages/disadvantages.

Hamza [62] highlighted open challenges and future directions in works using HE for AI-based applications together with a comparison between HE algorithms and libraries. They also discussed parallelization opportunities and assessed the quality of the solutions' documentation. Xu *et al.* [154] and Patel *et al.* [124] identified some future research directions in the field. To enhance scalability, the latter suggests parallelization and the exploration of lightweight privacy-preserving methods suitable for resource-constrained environments. To reduce communication overhead, they propose using model compression, sparsification, and aggregation methods, among others.

Other works discussed the **limitations of private inference**, along with potential solutions [8, 12]. The latter analyzed the major limitations identified in the literature, the most common approaches used to address those limitations, and identified the most used types of NNs and the most commonly used HE schemes. Hong [68] reviewed recent representative results of FHE-based PPML, discussing some results for inference on models such as CryptoNets, TAPAS, SHE, general pre-trained models, tree-based models, ResNets, and transformers. The survey by Yuan *et al.* [158] focused on training and inference solutions based exclusively on CKKS, covering different types of data inputs until 2024; however, it does not cover the considerable progress made since then. There are studies focused on **inference and FHE**, aligning with our own focus, such as Pulido-Gaytan *et al.* [128], which discussed machine-learning as a service (MLaaS) for inference, with FHE as the tool for preserving privacy. However, that work is now outdated, and it does not include CKKS implementations.

Further surveys covered solutions that use **other cryptographic primitives for privacy along with HE**. Aloufi *et al.* [11] discussed not only two-party HE, but also multi-party settings and other schemes, including threshold HE, multi-key HE, and attribute-based encryption. Rechberger and Walch [131] contrasted HE and MPC and the scenarios in which each of these may be used. They identified MLaaS as a suitable scenario for FHE, including training and inference tasks. They also discussed some of the limitations of the usage of FHE and MPC in machine-learning protocols, such as being able to evaluate only arithmetic circuits. We note that they did not consider any other solution, such as replacing non-linear functions with HE-friendly ones, which we discuss in Section 5.2.

Along the same lines, but focused on inference, there are surveys that **compare MPC-based, FHE-based, and hybrid solutions**. Azraoui *et al.* [15] compared and evaluated CKKS-based, MPC-based, and hybrid CKKS-MPC solutions on relatively small datasets. The work also discusses the drawbacks of not being able to evaluate non-linear layers. According to the paper, the CKKS scheme has the advantage of supporting batched computations. Similarly, Mann *et al.* [111] focused on secure NN inference solutions using techniques from HE and secure MPC, namely, secret sharing, garbled circuits, and oblivious transfer. The focus of the work is on the inference phase and, in their view, the requirements comprise low latency and high accuracy. Furthermore, Li *et al.* [102] compared approaches that use two-party computation (2PC) techniques and HE, including hardware acceleration. However, that survey only reviewed CryptoNets and PPCDNN [31], and it concluded that the best approach for secure NN prediction is a hybrid solution between 2PC and HE, an option we argued against in Section 4. Li *et al.* [104] published a survey on private transformer inference that considers not only FHE but also MPC and functional encryption.

We intentionally exclude from our review of related work that is further from our focus, such as surveys that discuss (F)HE schemes from a theoretical perspective, surveys focused on terminology definitions, and surveys on other

privacy-preserving technologies such as differential privacy, trusted execution environments, federated learning, or functional encryption.

Some previous surveys considered both HE and MPC, and many did not consider bootstrapping, an essential ingredient for supporting the computational depth of DNN architectures. In contrast, our survey focuses specifically on FHE-based inference of large DNN models, including recently emerged solutions.

8 Conclusion

In this work, we describe what we would consider to be a practical private inference engine and summarize the state of the art in private inference and its applications. Although considerable improvements have been made in recent years, we conclude that many challenges remain. While bootstrapping has enabled private inference on deeper models, scalability remains a major concern. Practical applications are currently either restricted to models with a small number of parameters or need to be able to tolerate high latencies. We have concluded that CKKS is the most promising FHE scheme for encrypted inference on larger models; however, the cost of bootstrapping (in terms of latency) and accurate activation functions (in terms of level consumption) still limit many applications.

The fact that deeper NNs require more bootstrapping calls exacerbates the latency problem, and implementations aim to find a good balance between bootstrapping frequency/placement and FHE parameters. Currently, many studies employ hand-crafted solutions to attain high performance, but this is only feasible for a small number of networks. For a general private inference engine, automated solutions need to be found to address this problem; otherwise, encrypted inference will remain restricted to certain architectures (e.g., ResNet) and thus be limited in scope. Similarly, polynomial approximation of the activation functions currently either requires a very high degree to attain sufficient accuracy or implies retraining the network. Ideally, we would like to find some solution with high accuracy and low latency produced with minimal fine-tuning. We believe that the emerging functional bootstrapping technique has the potential to solve both problems, since the bootstrapping would also evaluate the activation function at the same time. However, the implications of using it apart from LUT precomputations remain to be studied.

Another observation that has to be made is that currently a considerable body of literature focuses on privacy-preserving CNNs, while the state of the art in ML has transitioned to transformer-based architectures that outperform CNNs in many applications. Even though CNNs may provide sufficient accuracy for certain privacy-sensitive applications, widespread adoption of private inference will probably only be possible if it is able to mimic the solution quality of current ML applications.

We have also observed a gap in hardware acceleration techniques and open-source parallelism support (only OpenFHE among open-source libraries supports parallelism for CKKS). The existing research around ASIC accelerators has primarily used small ML models for benchmark/evaluation studies, which calls into question their performance on large ML workloads like LLMs. Due to the memory explosion in the FHE domain, cluster-based approaches, as proposed by Cinnamon [73], should be prioritized. We believe that the ideal inference engine has to combine techniques from software scalability and hardware acceleration to produce an entire system that is practical in production environments.

Acknowledgments

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council Executive Agency. Neither the European Union nor the granting authority can be held responsible for them. This work is supported by ERC grant

(HomE, 101043467). Antonio J. Peña was in addition partially supported by the Ramón y Cajal fellowship RYC2020-030054-I funded by MCIN/AEI/ 10.13039/501100011033 and by "ESF Investing in your future".

References

[1] 2022. Lattigo v4. Online: https://github.com/tuneinsight/lattigo. EPFL-LDS, Tune Insight SA.

[2] Rashmi Agrawal, Anantha Chandrakasan, and Ajay Joshi. 2024. HEAP: A Fully Homomorphic Encryption Accelerator with Parallelized Bootstrapping. In *ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. 756–769.

[3] Rashmi Agrawal, Leo de Castro, Guowei Yang, Chiraag Juvekar, Rabia Yazicigil, Anantha Chandrakasan, Vinod Vaikuntanathan, and Ajay Joshi. 2023. FAB: An FPGA-based Accelerator for Bootstrappable Fully Homomorphic Encryption. In *IEEE Int. Symp. on High-Performance Comp. Arch.*

[4] Rashmi Agrawal and Ajay Joshi. 2023. *On architecting fully homomorphic encryption-based computing systems*. Springer.

[5] 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. 2023. HeLayers: A Tile Tensors Framework for Large Neural Networks on Encrypted Data. *Privacy Enhancing Technologies* 1, 325–342.

[6] Ahmad Al Badawi, Andreaa Alexandru, Jack Bates, Flavio Bergamaschi, David Bruce Cousins, Saroja Erabelli, Nicholas Genise, Shai Halevi, Hamish Hunt, et al. 2022. OpenFHE: Open-Source Fully Homomorphic Encryption Library. In *10th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*.

[7] Ahmad Al Badawi and Yuriy Polyakov. 2023. Demystifying Bootstrapping in Fully Homomorphic Encryption. *Cryptology Archive, 2023/149* (2023).

[8] Anmar A Al-Janabi, Sufyan T Faraj Al-Janabi, and Belal Al-Khateeb. 2023. Secure Data Computation Using Deep Learning and Homomorphic Encryption: A Survey. *International Journal of Online & Biomedical Engineering* 19, 11 (2023).

[9] Martin R Albrecht. 2021. Lattice estimator documentation. https://lattice-estimator.readthedocs.io/en/latest/.

[10] Andreea Alexandru, Andrey Kim, and Yuriy Polyakov. 2025. General Functional Bootstrapping Using CKKS. In *Advances in Cryptology (CRYPTO)*.

[11] Asma Aloufi, Peizhao Hu, Yongsoo Song, and Kristin Lauter. 2020. Computing Blindfolded on Data Homomorphically Encrypted under Multiple Keys: An Extended Survey. *arXiv:2007.09270* (2020).

[12] Ivone Amorim, Eva Maia, Pedro Barbosa, and Isabel Praça. 2023. Data Privacy with Homomorphic Encryption in Neural Networks Training and Inference. In *International Symposium on Distributed Computing and Artificial Intelligence*. Springer, 365–374.

[13] Wei Ao and Vishnu Boddeti. 2025. CryptoFace: End-To-End Encrypted Face Recognition. In *Computer Vision and Pattern Recognition Conference*.

[14] Wei Ao and Vishnu Naresh Boddeti. 2024. AutoFHE: Automated Adaption of CNNs for Efficient Evaluation over FHE. In *USENIX Security Symp.*

[15] Monir Azraoui, Muhammad Bahram, Beyza Bozdemir, Sébastien Canard, Eleonora Ciceri, Orhan Ermis, Ramy Masalha, Marco Mosconi, Melek Önen, Marie Paindavoine, Boris Rozenberg, Bastien Vialla, and Sauro Vicini. 2020. *SoK: Cryptography for Neural Networks*. Springer, 63–81.

[16] Youngjin Bae, Jung Hee Cheon, Wonhee Cho, Jaehyung Kim, and Taekyung Kim. 2022. META-BTS: Bootstrapping Precision Beyond the Limit. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS '22)*. 223–234.

[17] Youngjin Bae, Jung Hee Cheon, Guillaume Hanrot, Jai Hyun Park, and Damien Stehlé. 2024. Plaintext-Ciphertext Matrix Multiplication and FHE Bootstrapping: Fast and Fused. In *Advances in Cryptology – CRYPTO 2024, Part III (Lecture Notes in Computer Science, Vol. 14922, 387–421)*.

[18] Tathagata Barik, Priyam Mehta, Zaira Pindado, Harshita Gupta, Mayank Kabra, Mohammad Sadrosadati, Onur Mutlu, and Antonio J. Peña. 2026. Long integer NTT execution on UPMEM-PIM for 128-bit secure fully homomorphic encryption. *Future Generation Computer Systems* 180 (2026), 108386. doi:10.1016/j.future.2026.108386

[19] Moran Baruch, Nir Drucker, Gilad Ezov, Yoav Goldberg, Eyal Kushnir, Jenny Lerner, Omri Soceanu, and Itamar Zimerman. 2024. Polynomial Adaptation of Large-Scale CNNs for Homomorphic Encryption-Based Secure Inference. In *Cyber Security, Cryptology, and Machine Learning*.

[20] Moran Baruch, Nir Drucker, Lev Greenberg, and Guy Moshkowich. 2022. A Methodology for Training Homomorphic Encryption Friendly Neural Networks. In *Applied Cryptography and Network Security Workshops*. Springer International Publishing, 536–553.

[21] Donald Beaver. 1992. Efficient Multiparty Protocols Using Circuit Randomization. In *Advances in Cryptology – CRYPTO'91 (Lecture Notes in Computer Science, Vol. 576)*, Joan Feigenbaum (Ed.). Springer Berlin Heidelberg, Germany, Santa Barbara, CA, USA, 420–432. doi:10.1007/3-540-46766-1_34

[22] Adrien Benamira, Tristan Guérand, Thomas Peyrin, and Sayandeep Saha. 2023. TT-TFHE: A Torus Fully Homomorphic Encryption-Friendly Neural Network Architecture. *arXiv:2302.01584* (2023).

[23] Loris Bergerat, Ilaria Chillotti, Damien Ligier, Jean-Baptiste Orfila, and Samuel Tap. 2025. TFHE Gets Real: an Efficient and Flexible Homomorphic Floating-Point Arithmetic. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2025, 2 (2025), 126–162.

[24] Fabian Boemer, Anamaria Costache, Rosario Cammarota, and Casimir Wierzynski. 2019. nGraph-HE2: A High-Throughput Framework for Neural Network Inference on Encrypted Data. In *7th ACM Workshop on Encrypted Computing & Applied Homomorphic Cryptography (WAHC'19)*. 45–56.

[25] Fabian Boemer, Anamaria Costache, Rosario Cammarota, and Casimir Wierzynski. 2019. nGraph-HE2: A High-Throughput Framework for Neural Network Inference on Encrypted Data. In *Proceedings of the 7th ACM Workshop on Encrypted Computing & Applied Homomorphic Cryptography* (London, United Kingdom) *(WAHC'19)*. Association for Computing Machinery, New York, NY, USA, 45–56. doi:10.1145/3338469.3358944

[26] Fabian Boemer, Sejun Kim, Gelila Seifu, Fillipe DM de Souza, and Vinodh Gopal. 2021. Intel HEXL: Accelerating Homomorphic Encryption with Intel AVX512-IFMA52. In *Proceedings of the 9th on Workshop on Encrypted Computing & Applied Homomorphic Cryptography*. 57–62.

[27] Jean-Philippe Bossuat, Rosario Cammarota, Ilaria Chillotti, Benjamin R Curtis, Wei Dai, Huijing Gong, Erin Hales, Duhyeong Kim, Bryan Kumara, Changmin Lee, et al. 2025. Security Guidelines for Implementing Homomorphic Encryption. *IACR Communications in Cryptology* 1, 4 (2025).

[28] Jean-Philippe Bossuat, Christian Mouchet, Juan Ramón Troncoso-Pastoriza, and Jean-Pierre Hubaux. 2021. Efficient Bootstrapping for Approximate Homomorphic Encryption with Non-sparse Keys. In *Advances in Cryptology (EUROCRYPT), Part I*. Springer, 587–617.

[29] Christina Boura, Nicolas Gama, Mariya Georgieva, and Dimitar Jetchev. 2020. Chimera: Combining Ring-LWE-based Fully Homomorphic Encryption schemes. *Journal of Mathematical Cryptology* 14, 1 (2020), 316–338.

[30] Alon Brutzkus, Oren Elisha, and Ran Gilad-Bachrach. 2019. Low Latency Privacy Preserving Inference. In *Int. Conference on Machine Learning*.

[31] Hervé Chabanne, Amaury de Wargny, Jonathan Milgram, Constance Morel, and Emmanuel Prouff. 2017. Privacy-Preserving Classification on Deep Neural Network. Cryptology ePrint Archive, Report 2017/035.

[32] J. Chao, A. Al Badawi, B. Unnikrishnan, J. Lin, C. F. Mun, J. M. Brown, J. P. Campbell, M. Chiang, J. Kalpathy-Kramer, V. R. Chandrasekhar, P. Krishnaswamy, and K. M. M. Aung. 2019. CaRENets: Compact and Resource-Efficient CNN for Homomorphic Inference on Encrypted Medical Images.

[33] Hao Chen, Ilaria Chillotti, and Yongsoo Song. 2019. Improved Bootstrapping for Approximate Homomorphic Encryption. In *Advances in Cryptology*. Springer.

[34] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. 2018. Bootstrapping for Approximate Homomorphic Encryption. In *Advances in Cryptology – EUROCRYPT, Part I (Lecture Notes in Computer Science, Vol. 10820, 4—54) (LNCS)*. Springer, 360–384.

[35] Jung Hee Cheon, Guillaume Hanrot, Jongmin Kim, and Damien Stehlé. 2025. SHIP: A Shallow and Highly Parallelizable CKKS Bootstrapping Algorithm. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 398–428.

[36] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. 2017. Homomorphic Encryption for Arithmetic of Approximate Numbers. In *Advances in Cryptology – ASIACRYPT 2017*, Tsuyoshi Takagi and Thomas Peyrin (Eds.). Springer International Publishing, Cham, 409–437.

[37] Jung Hee Cheon, Dongwoo Kim, and Duhyeong Kim. 2020. Efficient Homomorphic Comparison Methods with Optimal Complexity. In *Advances in Cryptology – ASIACRYPT 2020 (Lecture Notes in Computer Science, Vol. 12492)*. Springer, 221–256.

[38] Seonyoung Cheon, Yongwoo Lee, Dongkwan Kim, Ju Min Lee, Sunchul Jung, Taekyung Kim, Dongyoon Lee, and Hanjun Kim. 2024. DaCapo: Automatic Bootstrapping Management for Efficient Fully Homomorphic Encryption. In *USENIX Security 2024: 33rd USENIX Security Symposium*.

[39] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène. 2020. TFHE: Fast Fully Homomorphic Encryption Over the Torus. *J. Cryptol.* 33, 1 (2020).

[40] Ilaria Chillotti, Marc Joye, Damien Ligier, Jean-Baptiste Orfila, and Samuel Tap. 2020. CONCRETE: Concrete Operates on Ciphertexts Rapidly by Extending TFHE. In *WAHC 2020–8th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, Vol. 15.

[41] Ilaria Chillotti, Marc Joye, and Pascal Paillier. 2021. Programmable Bootstrapping Enables Efficient Homomorphic Inference of Deep Neural Networks. Cryptology ePrint Archive, Report 2021/091. https://eprint.iacr.org/2021/091

[42] Wonseok Choi, Jongmin Kim, and Jung Ho Ahn. 2025. Cheddar: A Swift Fully Homomorphic Encryption Library Designed for GPU Architectures *(ASPLOS '26)*. Association for Computing Machinery, New York, NY, USA, 35–49. doi:10.1145/3760250.3762223

[43] Benjamin R Curtis and Rachel Player. 2019. On the Feasibility and Impact of Standardising Sparse-secret LWE Parameter Sets for Homomorphic Encryption. In *Proceedings of the 7th ACM Workshop on Encrypted Computing & Applied Homomorphic Cryptography*. 1–10.

[44] Roshan Dathathri, Blagovesta Kostova, Olli Saarikivi, Wei Dai, Kim Laine, and Madan Musuvathi. 2020. EVA: An Encrypted Vector Arithmetic Language and Compiler for Efficient Homomorphic Computation. In *ACM SIGPLAN Conf. on Programming Language Design and Implementation*.

[45] Roshan Dathathri, Olli Saarikivi, Hao Chen, Kim Laine, Kristin Lauter, Saeed Maleki, Madanlal Musuvathi, and Todd Mytkowicz. 2019. CHET: An Optimizing Compiler for Fully-Homomorphic Neural-Network Inferencing. In *ACM SIGPLAN Conf. Programming Language Design Implementation*.

[46] Leo de Castro, Rashmi Agrawal, Rabia Yazicigil, Anantha Chandrakasan, Vinod Vaikuntanathan, Chiraag Juvekar, and Ajay Joshi. 2021. Does Fully Homomorphic Encryption Need Compute Acceleration? Cryptology ePrint Archive, Report 2021/1636.

[47] Leo de Castro, Daniel Escudero, Adya Agrawal, Antigoni Polychroniadou, and Manuela Veloso. 2025. EncryptedLLM: Privacy-Preserving Large Language Model Inference via GPU-Accelerated Fully Homomorphic Encryption. In *Forty-second International Conference on Machine Learning*.

[48] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. 2020. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. *arXiv:2010.11929* (2020).

[49] Jules Dumezy, Andreea Alexandru, Yuriy Polyakov, Pierre-Emmanuel Clet, Olive Chakraborty, and Aymen Boudguiga. 2025. Evaluating Larger Lookup Tables using CKKS. Cryptology ePrint Archive, Paper 2025/1301.

[50] Austin Ebel, Karthik Garimella, and Brandon Reagen. 2025. Orion: A Fully Homomorphic Encryption Framework for Deep Learning. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. 734–749.

[51] Austin Ebel and Brandon Reagen. 2024. Osiris: A Systolic Approach to Accelerating Fully Homomorphic Encryption. *arXiv:2408.09593* (2024).

[52] G. Fan, M. Zhang, F. Zheng, S. Fan, T. Zhou, X. Deng, W. Tang, L. Kong, Y. Song, and S. Yan. 2025. WarpDrive: GPU-Based Fully Homomorphic Encryption Acceleration Leveraging Tensor and CUDA Cores. In *IEEE Int. Symp. on High Performance Computer Architecture (HPCA)*. 1187–1200.

[53] Shengyu Fan, Zhiwei Wang, Weizhi Xu, Rui Hou, Dan Meng, and Mingzhe Zhang. 2023. TensorFHE: Achieving Practical Computation on Encrypted Data Using GPGPU. In *IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 922–934.

[54] Axel Feldmann, Nikola Samardzic, Aleksandar Krastev, Srinivas Devadas, Ronald Dreslinski, Christopher Peikert, and Daniel Sanchez. 2021. F1: A Fast and Programmable Accelerator for Fully Homomorphic Encryption. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*.

[55] K. Garimella, N. K. Jha, and B. Reagen. 2021. Sisyphus: A Cautionary Tale of Using Low-Degree Polynomial Activations in Privacy-Preserving Deep Learning. *arXiv:2107.12342* (2021).

[56] Craig Gentry. 2009. *A Fully Homomorphic Encryption Scheme*. Ph. D. Dissertation. Stanford University.

[57] Craig Gentry and Yongwoo Lee. 2025. Fully Homomorphic Encryption for Matrix Arithmetic. Cryptology ePrint Archive, Report 2025/1935.

[58] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. 2016. CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy. In *Proceedings of The 33rd International Conference on Machine Learning*. PMLR, 201–210.

[59] Robert E. Goldschmidt. 1964. Applications of division by convergence. In *Thesis (M.S.)–Massachusetts Institute of Technology, Dept. of Electrical Eng.*

[60] Antonio Guimarães, Edson Borin, and Diego F. Aranha. 2021. Revisiting the functional bootstrap in TFHE. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2021, 2 (2021), 229–253. doi:10.46586/tches.v2021.i2.229-253

[61] Shai Halevi and Victor Shoup. 2018. Faster Homomorphic Linear Transformations in HElib. In *Advances in Cryptology (CRYPTO)*. 93–120.

[62] Rafik Hamza. 2023. Homomorphic Encryption for AI-based Applications: Challenges and Opportunities. In *Int. Conf. on Knowledge & Systems Eng.*

[63] Kyoohyung Han and Dohyeong Ki. 2020. Better Bootstrapping for Approximate Homomorphic Encryption. In *Topics in Cryptology (CT-RSA)*.

[64] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[65] Zhiyu He, Maojiang Wang, Xinwen Gao, Yuchuan Luo, Lin Liu, and Shaojing Fu. 2025. ENSI: Efficient Non-Interactive Secure Inference for Large Language Models. (2025), 83–94.

[66] Ehsan Hesamifard, Hassan Takabi, and Mehdi Ghasemi. 2017. CryptoDL: Deep Neural Networks over Encrypted Data. *arXiv:1711.05189* (2017).

[67] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. 2015. Distilling the Knowledge in a Neural Network. *arXiv:1503.02531* (2015).

[68] Cheng Hong. 2025. Recent Advances of Privacy-Preserving Machine Learning Based on (Fully) Homomorphic Encryption. *Security Safety* 4 (2025).

[69] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-Rank Adaptation of Large Language Models. In *International Conference on Learning Representations (ICLR)*, Vol. 1. 3.

[70] Zhicong Huang, Wen-jie Lu, Cheng Hong, and Jiansheng Ding. 2022. Cheetah: Lean and Fast Secure Two-Party Deep Neural Network Inference. In *31st USENIX Security Symposium*. 809–826.

[71] Francesco Intoci, Sinem Sav, Apostolos Pyrgelis, Jean-Philippe Bossuat, Juan Ramón Troncoso-Pastoriza, and Jean-Pierre Hubaux. 2023. slytHErin: An Agile Framework for Encrypted Deep Neural Network Inference. In *Applied Cryptography and Network Security Workshops*. 359–377.

[72] Jaehee Jang, Younho Lee, Andrey Kim, Byunggook Na, Donggeon Yhee, Byounghan Lee, Jung Hee Cheon, and Sungroh Yoon. 2022. Privacy-Preserving Deep Sequential Model with Matrix Homomorphic Encryption. In *ACM Asia Conf. on Computer & Communications Security*. 377–391.

[73] Siddharth Jayashankar, Edward Chen, Tom Tang, Wenting Zheng, and Dimitrios Skarlatos. 2025. Cinnamon: A Framework for Scale-Out Encrypted AI. In *30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1 (ASPLOS)*. 133–150.

[74] Xiaoqian Jiang, Miran Kim, Kristin Lauter, and Yongsoo Song. 2018. Secure Outsourced Matrix Computation and Application to Neural Networks. In *ACM SIGSAC Conference on Computer and Communications Security (CCS'18)*. Association Computing Machinery, 1209–1222.

[75] Dian Jiao, Xianglong Deng, Zhiwei Wang, Shengyu Fan, Yi Chen, Dan Meng, Rui Hou, and Mingzhe Zhang. 2025. Neo: Towards Efficient Fully Homomorphic Encryption Acceleration using Tensor Core. In *52nd Annual International Symposium on Computer Architecture*. 107–121.

[76] Wen jie Lu, Zhicong Huang, Zhen Gu, Jingyu Li, Jian Liu, Cheng Hong, Kui Ren, Tao Wei, and Wenguang Chen. 2025. BumbleBee: Secure Two-party Inference Framework for Large Transformers. In *ISOC Network and Distributed System Security Symposium – NDSS 2025*. The Internet Society.

[77] Gilbert Jonatan, Haeyoon Cho, Hyojun Son, Xiangyu Wu, Neal Livesay, Evelio Mora, Kaustubh Shivdikar, José L Abellán, Ajay Joshi, David Kaeli, et al. 2024. Scalability limitations of processing-in-memory using real system evaluations. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 8, 1 (2024), 1–28.

[78] Marc Joye. 2022. SoK: Fully Homomorphic Encryption over the [Discretized] Torus. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2022, 4 (2022), 661–692. doi:10.46586/tches.v2022.i4.661-692

[79] Jae Hyung Ju, Jaiyoung Park, Jongmin Kim, Minsik Kang, Donghwan Kim, Jung Hee Cheon, and Jung Ho Ahn. 2024. NeuJeans: Private Neural Network Inference with Joint Optimization of Convolution and FHE Bootstrapping. In *Conference on Computer and Communications Security*.

[80] Wonkyung Jung, Sangpyo Kim, Jung Ho Ahn, Jung Hee Cheon, and Younho Lee. 2021. Over 100x Faster Bootstrapping in Fully Homomorphic Encryption through Memory-centric Optimization with GPUs. *IACR Transactions on Cryptographic Hardware and Embedded Systems 2021* (2021).

[81] Wonkyung Jung, Eojin Lee, Sangpyo Kim, Jongmin Kim, Namhoon Kim, Keewoo Lee, Chohong Min, Jung Hee Cheon, and Jung Ho Ahn. 2021. Accelerating Fully Homomorphic Encryption Through Architecture-Centric Analysis and Optimization. *IEEE Access* 9 (2021), 98772–98789.

[82] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. 2018. GAZELLE: A Low Latency Framework for Secure Neural Network Inference. In *USENIX Security 2018: 27th USENIX Security Symposium*. USENIX Association.

[83] Dongwoo Kim and Cyril Guyot. 2023. Optimized Privacy-Preserving CNN Inference with Fully Homomorphic Encryption. *IEEE Transactions on Information Forensics and Security* 18 (2023), 2175–2187.

[84] Donghwan Kim, Jaiyoung Park, Jongmin Kim, Sangpyo Kim, and Jung Ho Ahn. 2024. HyPHEN: A Hybrid Packing Method and Its Optimizations for Homomorphic Encryption-Based Neural Networks. *IEEE Access* 12 (2024), 3024–3038.

[85] Jongmin Kim, Sangpyo Kim, Jaewan Choi, Jaiyoung Park, Donghwan Kim, and Jung Ho Ahn. 2023. SHARP: A Short-Word Hierarchical Accelerator for Robust and Practical Fully Homomorphic Encryption. In *50th Annual International Symposium on Computer Architecture (ISCA)*.

[86] Jongmin Kim, Gwangho Lee, Sangpyo Kim, Gina Sohn, Minsoo Rhu, John Kim, and Jung Ho Ahn. 2022. ARK: Fully Homomorphic Encryption Accelerator with Runtime Data Generation and Inter-Operation Key Reuse. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*.

[87] Jaehyung Kim and Taeyeong Noh. 2024. Modular Reduction in CKKS. Cryptology ePrint Archive, Paper 2024/1638.

[88] Miran Kim, Xiaoqian Jiang, Kristin Lauter, Elkhan Ismayilzada, and Shayan Shams. 2022. Secure Human Action Recognition by Encrypted Neural Network Inference. *Nature Communications* 13 (2022).

[89] Sangpyo Kim, Jongmin Kim, Jaeyoung Choi, and Jung Ho Ahn. 2024. CiFHER: A Chiplet-Based FHE Accelerator with a Resizable Structure. In *Proceedings of the International Symposium on Secure and Private Execution Environment Design (SEED)*. 119–130.

[90] Sangpyo Kim, Jongmin Kim, Michael Jaemin Kim, Wonkyung Jung, John Kim, Minsoo Rhu, and Jung Ho Ahn. 2022. BTS: An Accelerator for Bootstrappable Fully Homomorphic Encryption. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*. 711–725.

[91] Elena Kirshanova, Chiara Marcolla, and Sergi Rovira. 2024. Guidance for Efficient Selection of Secure Parameters for Fully Homomorphic Encryption. In *AFRICACRYPT 24: 15th International Conference on Cryptology in Africa, Lecture Notes in Computer Science, 14861, 376–400*. Springer.

[92] Christian Knabenhans, Alexander Viand, Antonio Merino-Gallardo, and Anwar Hithnawi. 2024. vFHE: Verifiable Fully Homomorphic Encryption. In *Proceedings of the 12th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*. 11–22.

[93] Aleksandar Krastev, Nikola Samardzic, Simon Langowski, Srinivas Devadas, and Daniel Sanchez. 2024. A Tensor Compiler with Automatic Data Packing for Simple and Efficient Fully Homomorphic Encryption. *Proc. ACM Program. Lang.* 8, PLDI, Article 152 (June 2024), 25 pages.

[94] Clemens Krüger, Bhavinkumar Moriya, and Dominik Schoop. 2025. A performance comparison of the homomorphic encryption schemes ckks and tfhe. In *Computation and Communication for Smart Systems Symposium*. Springer, 92–108.

[95] Eunsang Lee, Joon-Woo Lee, Junghyun Lee, Young-Sik Kim, Yongjune Kim, Jong-Seon No, and Woosuk Choi. 2022. Low-Complexity Deep Convolutional Neural Networks on Fully Homomorphic Encryption using Multiplexed Parallel Convolutions. In *Int. Conf. on Machine Learning*.

[96] Junghyun Lee, Eunsang Lee, Joon-Woo Lee, Yongjune Kim, Young-Sik Kim, and Jong-Seon No. 2023. Precise Approximation of Convolutional Neural Networks for Homomorphically Encrypted Data. *IEEE Access* 11 (2023), 62062–62076.

[97] Joon-Woo Lee, HyungChul Kang, Yongwoo Lee, Woosuk Choi, Jieun Eom, Maxim Deryabin, Eunsang Lee, Junghyun Lee, Donghoon Yoo, Young-Sik Kim, et al. 2022. Privacy-Preserving Machine Learning with Fully Homomorphic Encryption for Deep Neural Network. *IEEE Access* (2022).

[98] Yongwoo Lee, Seonyoung Cheon, Dongkwan Kim, Dongyoon Lee, and Hanjun Kim. 2023. ELASM: error-latency-aware scale management for fully homomorphic encryption. In *Proceedings of the 32nd USENIX Conference on Security Symposium* (Anaheim, CA, USA) *(SEC '23)*. USENIX Association, USA, Article 263, 18 pages.

[99] Yongwoo Lee, Joon-Woo Lee, Young-Sik Kim, Yongjune Kim, Jong-Seon No, and HyungChul Kang. 2022. High-Precision Bootstrapping for Approximate Homomorphic Encryption by Error Variance Minimization. In *Int. Conf. on the Theory and Applications of Cryptographic Techniques*.

[100] Dai Li, Akhil Pakala, and Kaiyuan Yang. 2022. MeNTT: A Compact and Efficient Processing-in-Memory Number Theoretic Transform (NTT) Accelerator. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 30, 5 (2022), 579–588.

[101] Long Li, Jianxin Lai, Peng Yuan, Tianxiang Sui, Yan Liu, Qing Zhu, Xiaojing Zhang, Linjie Xiao, Wenguang Chen, and Jingling Xue. 2025. ANT-ACE: An FHE Compiler Framework for Automating Neural Network Inference. In *ACM/IEEE Int. Symp. on Code Generation and Optimization*.

[102] M. Li, Y. Yan, Q. Wang, M. Du, Z. Qin, and C. Wang. 2021. Secure Prediction of Neural Network in the Cloud. *IEEE Network* 35 (2021), 251–257.

[103] Shaohua Li, Kaiping Xue, Bin Zhu, Chenkai Ding, Xindi Gao, David Wei, and Tao Wan. 2020. FALCON: A Fourier Transform Based Approach for Fast and Secure Convolutional Neural Network Predictions. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

[104] Yang Li, Xinyu Zhou, Yitong Wang, Liangxin Qian, and Jun Zhao. 2024. A Survey on Private Transformer Inference. *arXiv:2412.08145* (2024).

[105] Lawrence Lim, Vikas Kalagi, Divyakant Agrawal, and Amr El Abbadi. 2025. Tricycle: Private transformer inference with tricyclic encodings. *Cryptology ePrint Archive* (2025).

[106] Tzu-Li Liu, Yu-Te Ku, Ming-Chien Ho, Feng-Hao Liu, Ming-Ching Chang, Chih-Fan Hsu, Wei-Chao Chen, and Shih-Hao Hung. 2023. An Efficient CKKS-FHEW/TFHE Hybrid Encrypted Inference Framework. In *Computer Security. ESORICS International Workshops*. 535–551.

[107] Guillermo Lloret-Talavera, Marc Jorda, Harald Servat, Fabian Boemer, Chetan Chauhan, Shigeki Tomishima, Nilesh N. Shah, and Antonio J. Peña. 2021. Enabling Homomorphically Encrypted Inference for Large DNN Models. *IEEE Trans. Comput.* 71, 5 (2021), 1145–1155.

[108] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. 2010. On Ideal Lattices and Learning with Errors over Rings. In *Advances in Cryptology*.

[109] Jiaao Ma, Ceyu Xu, and Lisa Wu Wills. 2023. PyTFHE: An End-to-End Compilation and Execution Framework for Fully Homomorphic Encryption Applications. In *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 24–34.

[110] Vivian Maloney, Richard Obrecht, Vikram Saraph, Prathibha Rama, and Kate Tallaksen. 2024. High-Resolution Convolutional Neural Networks on Homomorphically Encrypted Data via Sharding Ciphertexts. *arXiv:2306.09189* (2024).

[111] Zoltán Ádám Mann, Christian Weinert, Daphnee Chabal, and Joppe W. Bos. 2023. Towards Practical Secure Neural Network Inference: The Journey So Far and the Road Ahead. *ACM Comput. Surv.* 56, 5 (2023).

[112] Jungho Moon, Zhanibek Omarov, Donghoon Yoo, Yongdae An, and Heewon Chung. 2024. Adaptive Successive Over-Relaxation Method for a Faster Iterative Approximation of Homomorphic Operations. *Cryptology ePrint Archive, Paper 2024/1366* (2024).

[113] Jungho Moon, Dongwoo Yoo, Xiaoqian Jiang, and Miran Kim. 2025. THOR: Secure Transformer Inference with Homomorphic Encryption. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. 3765–3779.

[114] Mpoki Mwaisela, Joel Hari, Peterson Yuhala, Jämes Ménétrey, Pascal Felber, and Valerio Schiavoni. 2024. Evaluating the Potential of in-Memory Processing to Accelerate Homomorphic Encryption: Practical Experience Report. In *43rd Int. Symp. on Reliable Distributed Systems (SRDS)*. 92–103.

[115] Nges Brian Njungle, Eric Jahns, and Michel A. Kinsy. 2025. FHEON: A Configurable Framework for Developing Privacy-Preserving Neural Networks Using Homomorphic Encryption. *arXiv:2510.03996* (2025).

[116] Martin Nocker, David Drexel, Michael Rader, Alessio Montuoro, and Pascal Schöttle. 2023. He-man–homomorphically encrypted machine learning with onnx models. (2023), 35–45.

[117] Ali Sah Özcan and Erkay Savas. 2024. HEonGPU: a GPU-based Fully Homomorphic Encryption Library 1.0. Cryptology ePrint Archive, 2024/1543.

[118] Marie Paindavoine and Bastien Vialla. 2016. Minimizing the Number of Bootstrappings in Fully Homomorphic Encryption. In *22nd Annual International Workshop on Selected Areas in Cryptography (SAC)*. 25–43.

[119] Qi Pang, Jinhao Zhu, Helen Möllering, Wenting Zheng, and Thomas Schneider. 2024. BOLT: Privacy-Preserving, Accurate and Efficient Inference for Transformers. In *IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 4753–4771.

[120] Dongjin Park, Eunsang Lee, and Joon-Woo Lee. 2025. 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, Vol. 1*. 11090–11111.

[121] Jaiyoung Park, Donghwan Kim, Jongmin Kim, Sangpyo Kim, Wonkyung Jung, Jung Hee Cheon, and Jung Ho Ahn. 2023. Toward Practical Privacy-Preserving Convolutional Neural Networks Exploiting Fully Homomorphic Encryption. *arXiv:2310.16530* (2023).

[122] Jaiyoung Park, Michael Jaemin Kim, Wonkyung Jung, and Jung Ho Ahn. 2022. AESPA: Accuracy Preserving Low-degree Polynomial Activation for Fast Private Inference. *arXiv:2201.06699* (2022).

[123] Jai Hyun Park. 2025. Ciphertext-Ciphertext Matrix Multiplication: Fast for Large Matrices. In *Advances in Cryptology (EUROCRYPT)*. 153–180.

[124] Helly Patel, Ajay Patel, and Asha Patel. 2024. A Comprehensive Analysis of Privacy-Preserving Techniques in Machine Learning. In *1st International Conference on Advances in Computing, Communication and Networking (ICAC2N)*. IEEE, 1836–1841.

[125] Robert Podschwadt and Daniel Takabi. 2021. Non-interactive Privacy Preserving Recurrent Neural Network Prediction with Homomorphic Encryption. In *IEEE 14th International Conference on Cloud Computing (CLOUD)*. IEEE, 65–70.

[126] Robert Podschwadt, Daniel Takabi, Peizhao Hu, Mohammad H. Rafiei, and Zhipeng Cai. 2022. A Survey of Deep Learning Architectures for Privacy-Preserving Machine Learning with Fully Homomorphic Encryption. *IEEE Access* 10 (2022), 117477–117500.

[127] Bernardo Pulido-Gaytan, Andrei Tchernykh, Jorge M Cortés-Mendoza, Mikhail Babenko, Gleb Radchenko, Arutyun Avetisyan, and Alexander Yu Drozdov. 2021. Privacy-Preserving Neural Networks with Homomorphic Encryption: Challenges and opportunities. *Peer-to-Peer Netw. Appl.* 14, 3 (2021), 1666–1691.

[128] Luis Bernardo Pulido-Gaytan, Andrei Tchernykh, Jorge M Cortés-Mendoza, Mikhail Babenko, and Gleb Radchenko. 2020. A Survey on Privacy-Preserving Machine Learning with Fully Homomorphic Encryption. In *Latin American High Performance Computing Conference*. Springer, 115–129.

[129] Hongyuan Qu and Guangwu Xu. 2023. Improvements of Homomorphic Secure Evaluation of Inverse Square Root. In *ICICS*. 110–127.

[130] Michael O. Rabin. 2005. How To Exchange Secrets with Oblivious Transfer. Cryptology ePrint Archive, Paper 2005/187.

[131] Christian Rechberger and Roman Walch. 2022. *Privacy-Preserving Machine Learning Using Cryptography*. Springer International, 109–129.

[132] Oded Regev. 2009. On Lattices, Learning with Errors, Random Linear Codes, and Cryptography. *Journal of the ACM (JACM)* 56, 6 (2009), 1–40.

[133] D. Rho, T. Kim, M. Park, J. W. Kim, H. Chae, E. K. Ryu, and J. H. Cheon. 2025. Encryption-Friendly LLM Architecture. In *Int. Conf. Learning Rep.*

[134] M. Sadegh Riazi, Kim Laine, Blake Pelton, and Wei Dai. 2020. HEAX: An Architecture for Computing on Encrypted Data. In *Proceedings of the 25th International Conference on Architectural Support for Programming Languages and Operating Systems*.

[135] Lorenzo Rovida and Alberto Leporati. 2024. Encrypted Image Classification with Low Memory Footprint using Fully Homomorphic Encryption. Cryptology ePrint Archive, Report 2024/460. (2024). https://eprint.iacr.org/2024/460

[136] L. Rovida and A. Leporati. 2024. Transformer-based Language Models and Homomorphic Encryption: An Intersection with BERT-Tiny. In *IWSPA'24: Proceedings of the 10th ACM International Workshop on Security and Privacy Analytics*. 3–13.

[137] Arjun Roy and Kaushik Roy. 2025. DCT-CryptoNets: Scaling private inference in the frequency domain. In *International Conference on Learning Representations*, Vol. 2025. 14690–14707.

[138] Nikola Samardzic, Axel Feldmann, Aleksandar Krastev, Nathan Manohar, Nicholas Genise, Srinivas Devadas, Karim Eldefrawy, Chris Peikert, and Daniel Sanchez. 2022. CraterLake: A Hardware Accelerator for Efficient Unbounded Computation on Encrypted Data. In *Int. Symp. Comp. Arch.*

[139] Guilherme T. S. Sato and Wakaha Ogata. 2025. Privacy-Preserving Inference of Machine Learning Models without Retraining. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* advpub (2025), 2025CIP0011.

[140] Shiyu Shen, Hao Yang, Yu Liu, Zhe Liu, and Yunlei Zhao. 2023. CARM: CUDA-Accelerated RNS Multiplication in Word-Wise Homomorphic Encryption Schemes for Internet of Things. 72, 07 (2023), 1999–2010.

[141] Menatallah Fadoua Slama, Hiba Guerrouache, Yacine Challal, Karima Benatchba, and Riyadh Baghdadi. 2025. Lightening Encrypted Convolutions: A GPU-Optimized Approach to Private Inference. In *IEEE Conference on Communications and Network Security (CNS)*. 1–9.

[142] Samuel Sousa and Roman Kern. 2023. How to Keep Text Private? A Systematic Review of Deep Learning Methods for Privacy-Preserving Natural Language Processing. *Artificial Intelligence Review* 56, 2 (2023), 1427–1492.

[143] Damien Stehlé, Ron Steinfeld, Keisuke Tanaka, and Keita Xagawa. 2009. Efficient Public Key Encryption Based on Ideal Lattices. In *Adv. in Crypt.*

[144] Andrei Stoian, Jordan Frery, Roman Bredehoft, Luis Montero, Celia Kherfallah, and Benoit Chevallier-Mames. 2023. Deep Neural Networks for Encrypted Inference with TFHE. In *Cyber Security, Cryptology, and Machine Learning*. Springer Nature Switzerland, Cham, 493–500.

[145] M. H. Stone. 1948. The Generalized Weierstrass Approximation Theorem. *Mathematics Magazine* 21, 4 (1948), 167–184.

[146] Daniel Takabi, Robert Podschwadt, Jeff Druce, Curt Wu, and Kevin Procopio. 2019. Privacy Preserving Neural Network Inference on Encrypted Data with GPUs. *arXiv:1911.11377* (2019).

[147] Mingxing Tan and Quoc Le. 2021. EfficientNetV2: Smaller Models and Faster Training. In *Int. Conference on Machine Learning*. 10096–10106.

[148] Alexander Viand, Patrick Jattke, Miro Haller, and Anwar Hithnawi. 2023. HECO: fully homomorphic encryption compiler. In *Proceedings of the 32nd USENIX Conference on Security Symposium* (Anaheim, CA, USA) *(SEC '23)*. USENIX Association, USA, Article 264, 18 pages.

[149] Alexander Viand, Patrick Jattke, and Anwar Hithnawi. 2021. SoK: Fully Homomorphic Encryption Compilers. In *IEEE Symp. on Security & Privacy*.

[150] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2018. GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding. In *EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*. 353–355.

[151] Ruida Wang, Jikang Bai, Xuan Shen, Xianhui Lu, Zhihao Li, Binwu Xiang, Zhiwei Wang, Hongyu Wang, Lutan Zhao, Kunpeng Wang, and Rui Hou. 2025. Tetris: Versatile TFHE LUT and Its Application to FHE Instruction Set Architecture. Cryptology ePrint Archive, Report 2025/1623. https://eprint.iacr.org/2025/1623

[152] Tommy White, Charles Gouert, Chengmo Yang, and Nektarios Georgios Tsoutsos. 2023. FHE-Booster: Accelerating Fully Homomorphic Execution with Fine-tuned Bootstrapping Scheduling. In *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. 293–303.

[153] Wei Wu, Jian Liu, Huimei Wang, Fengyi Tang, and Ming Xian. 2018. PPolyNets: Achieving High Prediction Accuracy and Efficiency With Parametric Polynomial Activations. *IEEE Access* 6 (2018), 72814–72823.

[154] R. Xu, N. Baracaldo, and J. Joshi. 2021. Privacy-Preserving Machine Learning: Methods, Challenges and Directions. *arXiv:2108.04417* (2021).

[155] Zhihan Xu, Tian Ye, Rajgopal Kannan, and Viktor K. Prasanna. 2025. FAST: FPGA Acceleration of Fully Homomorphic Encryption with Efficient Bootstrapping. In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA '25)*. 115–126.

[156] Yinghao Yang, Huaizhi Zhang, Shengyu Fan, Hang Lu, Mingzhe Zhang, and Xiaowei Li. 2023. Poseidon: Practical Homomorphic Encryption Accelerator. In *IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 870–881.

[157] Andrew Chi-Chih Yao. 1982. Protocols for Secure Computations. In *Annual Symposium on Foundations of Computer Science (FOCS)*. 160–164.

[158] Jiangjun Yuan, Weinan Liu, Jiawen Shi, and Qingqing Li. 2025. Approximate Homomorphic Encryption Based Privacy-Preserving Machine Learning: a Survey. *Artificial Intelligence Review* 58, 3 (2025), 82.

[159] Zama. 2022. Concrete ML: a Privacy-Preserving Machine Learning Library using Fully Homomorphic Encryption for Data Scientists. https://github.com/zama-ai/concrete-ml.

[160] Zama. 2022. Concrete: TFHE Compiler that Converts Python Programs into FHE Equivalent. https://github.com/zama-ai/concrete.

[161] Junxue Zhang, Xiaodian Cheng, Liu Yang, Jinbin Hu, Ximeng Liu, and Kai Chen. 2024. SoK: Fully Homomorphic Encryption Accelerators. In *ACM Computing Surveys*, Vol. 56. Association for Computing Machinery, New York, NY, USA.

[162] Jiawen Zhang, Xinpeng Yang, Lipeng He, Kejia Chen, Wen-jie Lu, Yinghao Wang, Xiaoyang Hou, Jian Liu, Kui Ren, and Xiaohu Yang. 2025. Secure Transformer Inference Made Non-interactive. In *The Network and Distributed System Security (NDSS) Symposium*.

[163] Linru Zhang, Xiangning Wang, Jun Jie Sim, Zhicong Huang, Jiahao Zhong, Huaxiong Wang, Pu Duan, and Kwok Yan Lam. 2025. MOAI: Module-Optimizing Architecture for Non-Interactive Secure Transformer Inference. Cryptology ePrint Archive, Paper 2025/991.

[164] Haoyun Zhu, Takuya Suzuki, and Hayato Yamana. 2023. Performance Comparison of Homomorphic Encrypted Convolutional Neural Network Inference Among HElib, Microsoft SEAL and OpenFHE. In *IEEE Asia-Pacific Conference on Computer Science and Data Engineering (CSDE)*.

[165] Itamar Zimerman, Moran Baruch, Nir Drucker, Gilad Ezov, Omri Soceanu, and Lior Wolf. 2024. Converting Transformers to Polynomial form for Secure Inference over Homomorphic Encryption. In *Forty-first International Conference on Machine Learning*.

Supplementary Material

Legend for the tables. BTS: bootstrap placement (M: manual, A: automatic); Encoding: convolution encoding (SE: slot encoding, CE: coefficient encoding, CinS: coefficients-in-slots); Packing: matrix packing (R: row, C: column, D: diagonal, T: tricyclic); Act.: activation handling (APP: polynomial approximation, REP: replacement, SSw.: scheme switching); MT: multithreading; ?: not disclosed; —: not reported.

Arch.	Work	Encoding	BTS	Act.	MT	Library	Latency (s)	Mem. (GB)	Acc. (%)	128-bit
AlexNet	Orion [50]	SE	A	APP	N	Lattigo	337.2	≤512	92.80	Y
DenseNet	Sato&Ogata [139]	SE	M	APP	Y	OpenFHE	599*	?	95.52	N
ResNet-9	Maloney *et al.* [110]	SE	M	REP	Y	OpenFHE	924	≤128	94.50	Y
Plain-20	Kim&Guyot [83]	CE	M	APP	N	Lattigo	368	≤100	90.32	Y
ResNet-20	ANT-ACE [101]	SE	A	APP	N	—	1454	≤512	91.20	Y
	AutoFHE [14]	SE	M	REP	Y	SEAL	45*	≤768	91.96	Y
	Fhelipe [93]	SE	A	APP	N	Lattigo	1555	?	91.00	Y
	FHEON [115]	SE	M	APP	?	OpenFHE	403	20.4	92.20	N
	HyPHEN [84]	SE	M	REP	Y	HEaaN	37.6	≤480	92.17	Y
	Lee *et al.* [97]	SE	M	APP	Y	SEAL	10602	172	92.43	N
	Lee *et al.* [95]	SE	M	APP	N	SEAL	2271	≤512	91.31	Y
	Orion [50]	SE	A	APP	N	Lattigo	618.2	≤512	93.40	Y
	Rov.&Lep. [135]	SE	M	APP	N	OpenFHE	336	~15.2	91.67	Y
ResNet-44	ANT-ACE [101]	SE	A	APP	N	—	3045	≤512	90.70	Y
	AutoFHE [14]	SE	M	REP	Y	SEAL	84*	≤768	92.55	Y
	HyPHEN [84]	SE	M	REP	Y	HEaaN	78.0	≤480	94.08	Y
ResNet-50	Maloney *et al.* [110]	SE	M	REP	Y	OpenFHE	4321	≤800	98.3	Y
ResNet-56	ANT-ACE [101]	SE	A	APP	N	—	3908	≤512	93.80	Y
	Lee *et al.* [95]	SE	M	APP	N	SEAL	6852	≤512	93.07	Y
VGG-11	AutoFHE [14]	SE	M	REP	Y	SEAL	16*	≤768	86.78	Y
	FHEON [115]	SE	M	APP	?	OpenFHE	1647	39.3	86.10	N
VGG-16	FHEON [115]	SE	M	APP	?	OpenFHE	2232	42.3	85.80	N
	Orion [50]	SE	A	APP	N	Lattigo	588.6	≤512	94.50	Y

Table 3. CKKS implementations on CPU for CIFAR-10. *: batched (amortized) inference.

Arch.	Work	Encoding	BTS	Act.	MT	Library	Latency (s)	Mem. (GB)	Acc. (%)	128-bit
ResNet-9	Maloney *et al.* [110]	SE	M	REP	Y	OpenFHE	942	≤128	75.3	Y
Plain-20	Kim&Guyot [83]	CE	M	APP	N	Lattigo	368	≤100	64.08	Y
ResNet-32	AutoFHE [14]	SE	M	REP	Y	SEAL	114*	≤768	71.34	Y
	Lee *et al.* [95]	SE	M	APP	N	SEAL	3942	≤512	69.43	Y
ResNet-34	FHEON [115]	SE	M	APP	?	OpenFHE	1594	≤64	74.4	N
VGG-11	AutoFHE [14]	SE	M	REP	Y	SEAL	48*	≤768	64.31	Y

Table 4. CKKS implementations on CPU for CIFAR-100. *: batched (amortized) inference.

Arch.	Work	Encoding	BTS	Act.	MT	Library	Latency (s)	Mem. (GB)	Acc. (%)	128-bit
Plain-18[a]	Kim&Guyot [83]	CE	M	APP	N	Lattigo	292	≤100	87.65[b]	Y
ResNet-18	HyPHEN [84]	SE	M	REP	Y	HEaaN	373.4	56.7	65.25	Y
ResNet-34	Orion [50]	SE	A	APP	N	Lattigo	14338	≤512	73.66	Y
ResNet-50	Maloney *et al.* [110]	SE	M	REP	Y	OpenFHE	13009	≤800	80.2	Y
	Orion [50]	SE	A	APP	N	Lattigo	32324	≤512	76.22	Y
	Baruch *et al.* [19]	SE	M	REP	Y	HEaaN	1752	35	78.60[c]	Y
DenseNet	Sato&Ogata [139]	SE	M	APP	Y	OpenFHE	32169	?	74.50	Y

Table 5. CKKS implementations on CPU for ImageNet. [a]Only the final 8 layers are encrypted. [b]Top-5 accuracy. [c]Accuracy of the HE-friendly unencrypted model.

Dataset	Arch.	Work	Encoding	BTS	Act.	MT	Library	Latency (s)	Mem. (GB)	Acc. (%)	128-bit
IMDB	RNN	Fhelipe [93]	SE	A	APP	N	Lattigo	4502	?	78	Y
LFW/AgeDB/CALFW CPLFW/CFP-FP	CryptoFaceNet16	CryptoFace [13]	SE	M	REP	Y	SEAL	1446	277	91.46	Y

Table 6. CKKS implementations on CPU for other datasets (text classification and face recognition).

Dataset	Arch.	Work	Packing	BTS	Act.	MT	Library	Tokens	Latency (s)	Mem. (GB)	Acc. (%)	128-bit
RTE	BERT-base	NEXUS [162]	R/C/D	M	APP	Y	SEAL	128	857*	≤128	69.88	Y
	BERT	MOAI [163]	C/D	M	APP	Y	SEAL	128	574.6*	?	65.5	Y
	Llama-3-8B	NEXUS [162]	R/C/D	M	APP	Y	SEAL	8	1088*	≤128	81.24	Y
SST	BERT-tiny	Tricycle [105]	T	M	APP	Y	OpenFHE	26.6[d]	322	≤4000	?	Y
SST-2	BERT-tiny[e]	Rov.&Lep. [136]	R	M	APP	?	OpenFHE	20	~375	18	79	Y
								40	~600	18	79	
	BERT-base	NEXUS [162]	R/C/D	M	APP	Y	SEAL	128	857*	≤128	92.11	Y
	BERT	MOAI [163]	C/D	M	APP	Y	SEAL	128	574.6*	?	93.8	Y
	Llama-3-8B	NEXUS [162]	R/C/D	M	APP	Y	SEAL	8	1088*	≤128	94.46	Y
QNLI	BERT-base	NEXUS [162]	R/C/D	M	APP	Y	SEAL	128	857*	≤128	89.90	Y
	BERT	MOAI [163]	C/D	M	APP	Y	SEAL	128	574.6*	?	90.5	Y
	Llama-3-8B	NEXUS [162]	R/C/D	M	APP	Y	SEAL	8	1088*	≤128	90.20	Y

Table 7. CKKS implementations on CPU for language understanding benchmarks (GLUE). *: batched (amortized) inference. [d]Average padded token count during evaluation. [e]Embedding dimension 20 (first row) and 40 (second row).

Arch.	Work	Encoding	BTS	Act.	Library	Latency (s)	Mem. (GB)	Acc. (%)	128-bit
AlexNet	DaCapo [38]	SE	A	APP	HEaaN	~55	24	89	Y
MobileNet	DaCapo [38]	SE	A	APP	HEaaN	~60	24	91.2	Y
ResNet-20	Cheddar [42]	SE	M	APP	—	0.82	80	91.88	Y
	DaCapo [38]	SE	A	APP	HEaaN	~20	24	90.7	Y
	HyPHEN [84]	SE	M	REP	HEaaN	1.4	≤80	92.17	Y
	NEO [75]	SE	M	APP	—	9.11	≤80	?	Y
	Park *et al.* [121]	SE	M	REP	?	1.4	≤80	92.18	?
	Slama *et al.* [141]	SE	M	APP	HEonGPU	29.8	≤160	91.97	?
	TensorFHE [53]	SE	M	APP	—	4.94*	40	?	Y
	WarpDrive [52]	SE	M	APP	—	5.88	≤80	?	Y
ResNet-44	HyPHEN [84]	SE	M	REP	HEaaN	3.0	≤80	94.08	Y
ResNet-56	NEO [75]	SE	M	APP	—	26.48	≤80	?	Y
SqueezeNet	DaCapo [38]	SE	A	APP	HEaaN	~45	24	89.2	Y
VGG-16	DaCapo [38]	SE	A	APP	HEaaN	~50	?	93.8	Y

Table 8. CKKS implementations on GPU for CIFAR-10. *: batched (amortized) inference.

Arch.	Work	Encoding	BTS	Act.	Library	Latency (s)	Mem. (GB)	Acc. (%)	128-bit
ResNet-18	HyPHEN [84]	SE	M	REP	HEaaN	14.7	56.7	65.25	Y
	NeuJeans [79]	CinS	M	APP/REP[f]	HEaaN	14.11/5.35[f]	≤80	64.8/66.4[f]	Y
	Park *et al.* [121]	SE	M	REP	?	14.7	≤80	69.78	?
ResNet-50	Baruch *et al.* [19]	SE	M	REP	HEaaN	193	35	78.60[g]	Y
	NeuJeans [79]	CinS	M	APP	HEaaN	56.1	≤80	74.1	Y

Table 9. CKKS implementations on GPU for ImageNet. [f] Obtained with AESPA activations. [g] Accuracy of the HE-friendly unencrypted model.

Dataset	Arch.	Work	Encoding	BTS	Act.	Library	Latency (s)	Mem. (GB)	Acc. (%)	128-bit
IMDB	LSTM	TensorFHE [53]	SE	M	APP	—	3.85*	40	?	Y
WikiText-103	BERT (6 layers)	Zimerman *et al.*† [165]	?	M	REP	HEaaN	305	≤80	?	Y

Table 10. CKKS implementations on GPU for text datasets. * batched (amortized) inference. †backend reported as hybrid CPU/GPU.

Dataset	Arch.	Work	Packing	BTS	Act.	Library	Tokens	Latency (s)	Mem. (GB)	Acc. (%)	128-bit
RTE	BERT-base	NEXUS [162]	R/C/D	M	APP	Phantom	128	37.34*	≤160	69.88	Y
	BERT-base	THOR [113]	D	M	APP	Liberate	128	602.26	≤80	71.12	Y
	BERT-base	Powerformer [120]	C	M	REP	Liberate	128	344.52	≤ 80	73.29	Y
	BERT	Rho *et al.* [133]	R	M	REP	HEaaN	128	26.5	24	99[i]	Y
	Llama-3-8B	NEXUS [162]	R/C/D	M	APP	Phantom	8	51.84*	≤160	81.24	Y
SST	BitNet	ENSI† [65]	C	M	APP	OpenFHE	2048	958.12*	≤80	55.08	Y
SST-2	BERT-base	NEXUS [162]	R/C/D	M	APP	Phantom	128	37.34*	≤160	92.11	Y
	BERT-base	THOR [113]	D	M	APP	Liberate	128	602.26	≤80	90.71	Y
	BERT	Rho *et al.* [133]	R	M	REP	HEaaN	128	26.5	24	100[i]	Y
	Llama-3-8B	NEXUS [162]	R/C/D	M	APP	Phantom	8	51.84*	≤160	94.46	Y
	GPT-2 small	EncryptedLLM [47]	?	M	APP	OpenFHE	128	~102	≤80	?	Y
QNLI	BERT-base	NEXUS [162]	R/C/D	M	APP	Phantom	128	37.34*	≤160	89.90	Y
	BERT	Rho *et al.* [133]	R	M	REP	HEaaN	128	26.5	24	99[i]	Y
	Llama-3-8B	NEXUS [162]	R/C/D	M	APP	Phantom	8	51.84*	≤160	90.20	Y
MRPC	BERT	Rho *et al.* [133]	R	M	REP	HEaaN	128	26.5	24	100[i]	Y
	BERT-base	THOR [113]	D	M	APP	Liberate	128	602.26	≤80	84.8/89.49[h]	Y
CoLA	BERT (2 layers)	Rho *et al.* [133]	R	M	REP	HEaaN	128	26.5	24	99[i]	Y
STS-B	BERT	Rho *et al.* [133]	R	M	REP	HEaaN	128	26.5	24	?	Y
HellaSwag / ARC / MNLI / ANLI / WiC / Social IQa	GPT-2 small	EncryptedLLM [47]	?	M	APP	OpenFHE	128	~102	≤80	?	Y
PIQA / COPA	BitNet	ENSI† [65]	C	M	APP	OpenFHE	2048	958.12*	≤80	65.89/71.26	Y

Table 11. CKKS implementations on GPU for language understanding and commonsense benchmarks. Accuracy is measured on the GLUE benchmark unless stated otherwise. *: batched (amortized) inference. [h] F1 metric. [i] Accuracy relative to the cleartext model. †: backend reported as hybrid CPU/GPU.

Work	Main focus	Act.	Word size	Freq. (GHz)	Parallelism	Latency (s)	Mem. (MB)	Acc. (%)	128-bit
FAST [155]	Bootstrap	APP	32	0.3	512 PEs	1.859	43	?	Y
HEAP [2]	HSSw	SSw.	36	0.3	512 MMs	0.267	43[j]	94	Y
Poseidon [156]	Bootstrap	APP	32	0.45	512 PEs	2.661	8.6	?	N

Table 12. FPGA-based hardware acceleration solutions for ResNet-20 on CIFAR-10. Mem. refers to on-chip memory. [j] Per FPGA; 8 FPGAs in total. HSSw stands for hybrid scheme switching. Parallelism reports the number of processing elements (PEs) or modular multipliers (MMs).

Dataset	Arch.	Work	Main focus	Act.	Word size	Freq. (GHz)	Power (W)	Area (mm^2)	Compiler/ DSL	Latency (s)	Mem. (MB)	Acc. (%)	128-bit
CIFAR-10	ResNet-20	ARK [86]	Bootstrap	—	64	1	281.3	418.3	N	0.125	512	?	Y
CIFAR-10	ResNet-20	BTS [90]	Bootstrap	—	64	1.2	163.2	373.6	N	1.91	512	?	Y
CIFAR-10	ResNet-20	CiFHER [89]	Chiplet	—	32	1	~120	310.59	N	0.222	256	?	Y
CIFAR-10	ResNet-20	Cinnamon [73]	Cluster	—	28	1	190^k	223.18^k	Y	0.070	56	?	Y
?	BERT-base			—						1.67		?	Y
CIFAR-10	ResNet-20	CraterLake [138]	Key switching	—	28	1	320	472	Y	0.321	256	?	Y
CIFAR-10	ResNet-20 [93]			APP						0.2		91.00	Y
IMDB	LSTM			—						0.224		?	Y
IMDB	RNN [93]			APP						0.4		78	Y
CIFAR-10	LoLA	F1 [54]	Flexible	—	32	1	180.4	151.4	Y	0.241	64	?	?
CIFAR-10	ResNet-20	Osiris [51]	Dataflow	—	40	1	116	246.9	N	0.069	210	?	Y
CIFAR-10	ResNet-20	SHARP [85]	Bit width	—	36	1	98	178.8	N	0.099	180	?	Y

Table 13. ASIC-based hardware acceleration solutions and their reported private inference results (simulated). Mem. refers to on-chip memory. Rows citing Fhelipe [93] correspond to the Fhelipe compiler executed on CraterLake. ^{k}Per chip.