From LUT to Layer: Neural Network Inference with CKKS Functional Bootstrapping

Matheus de O. Saldanha[1], Guilherme A. Horn[1], Mauricio L. Ferreira[1], Andreis G. M. Purim[1], Alex L. D. Cassinelli[1], Hilder V. L. Pereira[1]

[1]Institute of Computing, University of Campinas (UNICAMP)

matheus@saldanhash.xyz, guilherme.horn@students.ic.unicamp.br
{mauricioleite.fe, andreis.purim, cassinelli.alex2}@dac.unicamp.br
hilder@unicamp.br

Abstract. *Neural networks implemented with homomorphic cryptography usually rely on costly polynomial approximations for non-linear activation functions. This work presents, to the best of our knowledge, the first implementation of neural-network inference using the recently proposed CKKS functional bootstrapping framework for arbitrary Look-Up Tables (LUTs). We implement fully connected neural networks for MNIST classification using LUT-based functional bootstrapping of sign, Heaviside, and ReLU activations, are compare against the traditional Chebyshev polynomial approximation. Across all evaluated configurations, the functional pipeline reduces inference latency by approximately 2 to 3.5 times while preserving accuracy more effectively for discontinuous activations at larger hidden widths, at the cost of an average memory overhead of approximately 1.6 times.*

1. Introduction

Homomorphic Encryption (HE) refers to a family of cryptographic schemes that allow computation directly over encrypted data, producing ciphertexts that decrypt to the same result as the corresponding plaintext computation. This property makes HE schemes useful for outsourcing computation to external servers without exposing the underlying data, including neural-network inference over encrypted user inputs. In the homomorphic setting, the linear layers of a neural network reduce primarily to ciphertext-plaintext operations because the weights and biases remain known at inference time. The non-linear activations instead become the dominant computational bottleneck, since they require a lot of ciphertext-ciphertext multiplications. Many HE schemes support only computations of bounded depth because ciphertexts accumulate noise during homomorphic operations, especially multiplications. As this noise increases, ciphertexts eventually become undecryptable. Fully Homomorphic Encryption (FHE) schemes address this limitation through a *bootstrapping* procedure that refreshes the ciphertext noise and enables further computation [Gentry 2009].

Early works on neural-network inference over encrypted data, such as CryptoNets [Gilad-Bachrach et al. 2016] and CryptoDL [Hesamifard et al. 2016], replaced

The authors gratefully acknowledge the financial support provided by the Technology Innovation Institute (TII) to some of the authors during the development of this work. This study was also financed, in part, by the São Paulo Research Foundation (FAPESP), Brazil, under Grant Nos. 2026/08423-8, 2024/23608-9, and 2023/12755-8.

standard non-linear activations with low-degree polynomials — either by training directly with polynomial activations or by approximating existing non-linearities such as ReLU — to limit ciphertext noise growth during homomorphic evaluation. Certain FHE schemes, notably TFHE [Chillotti et al. 2020], instead support *functional bootstrapping*, in which non-linear functions of small domain are evaluated during the noise-refreshing step itself. The FHE-DiNN framework [Bourse et al. 2018] applies a functional bootstrap at every neuron in order to evaluate discretized neural networks over encrypted inputs with homomorphic complexity that grows linearly with network depth.

These TFHE-based systems achieve efficiency partly through discretized arithmetic, i.e., operating over integers, whereas many machine-learning models rely heavily on real-valued computations. Unlike integer-only schemes, the CKKS scheme [Cheon et al. 2017] supports approximate arithmetic over real and complex numbers, making it well suited for homomorphic machine-learning inference. Historically, CKKS relied on polynomial approximations such as Chebyshev interpolation for non-linear activations, which perform function evaluation separately from the bootstrapping step. Recently, general functional bootstrapping techniques for CKKS were introduced [Alexandru et al. 2025], enabling the evaluation of small domain up to 14 bits arbitrary Look-Up Tables (LUTs) during bootstrapping. This enables activations such as sign and ReLU to be evaluated during bootstrapping, while simultaneously reducing the ciphertext noise and enabling further homomorphic computation.

Previous work such as RBOOT [Yang et al. 2025] introduced specialized CKKS bootstrapping procedures for evaluating ReLU activations with lower multiplicative depth than polynomial approximations. In contrast, this work implements and evaluates, to the best of our knowledge, the first homomorphic neural-network inference pipeline based on the general CKKS functional bootstrapping framework [Alexandru et al. 2025]. More specifically, we implement fully connected neural networks inspired by the FHE–DiNN approach [Bourse et al. 2018], replacing polynomial activation approximations with LUT-based CKKS functional bootstrapping. Our preliminary contributions include: (i) a set of neural networks for MNIST classification using CKKS functional bootstrapping for activation evaluation; (ii) a systematic evaluation of accuracy, latency, and memory usage compared to polynomial activation approximations; and (iii) an open-source implementation to support reproducibility and further research.

2. Methodology

CKKS ciphertexts are organized around a chain of *multiplicative levels*: each homomorphic multiplication consumes one level, and once the chain is exhausted, bootstrapping is required to restore the available levels and enable further computation. CKKS also supports *Single Instruction, Multiple Data* (SIMD) packing, where multiple values are encoded into the slots of a single ciphertext and every homomorphic operation is applied to all slots simultaneously. For instance, the addition of two encrypted vectors, $\mathsf{Add}(\mathsf{Enc}(\mathbf{u}), \mathsf{Enc}(\mathbf{v})) = \mathsf{Enc}(\mathbf{w})$, yields $w_i = u_i + v_i$ for all i upon decryption, with no additional cost over adding a single pair of values. This batching capability amortizes the overhead of homomorphic operations across the entire vector.

The functional bootstrapping method of [Alexandru et al. 2025] requires the input plaintexts to be represented as integers in $\mathbb{Z}_p^N$, although the network weights and biases may

remain real-valued. The BFV scheme [Fan and Vercauteren 2012], which supports exact integer arithmetic, is therefore used for the initial integer-domain representation before switching into CKKS for approximate real-valued computation as proposed by the authors. The remainder of this section describes the three pipeline stages: *data preparation* (Phase I), *encrypted inference* (Phase II), and *output extraction* (Phase III).

2.1. Phase I: Data Preparation and Domain Switching

The CKKS functional bootstrapping procedure requires inputs to be represented as integers in the plaintext space $\mathbb{Z}_p$. Consequently, raw input features $\mathbf{x} \in \mathbb{R}^n$ undergo a discretization step: $\bar{x}_i = \lfloor x_i \rfloor \pmod{p}$. These discretized values are first encrypted using the BFV scheme, which provides the necessary algebraic structure for the initial domain transitions.

To leverage CKKS for real-valued matrix multiplications, we perform a homomorphic *scheme switch* from BFV to CKKS. This transition allows the encrypted integers to be treated as complex numbers in the CKKS slot domain. Once in the CKKS domain, we adopt an *intra-sample packing* strategy where the n features of a single input occupy n consecutive slots — e.g., the n pixels of an image. This packing is essential because the functional bootstrap evaluates the LUT independently on each slot, preventing cross-talk between different features or batch instances.

2.2. Phase II: Homomorphic Inference and Functional Bootstrapping

The core of the inference pipeline consists of alternating linear transformations and functional bootstrapping stages.

Linear Transformation. A linear layer computes, for each output neuron $j \in \{1, \ldots, m\}$, the pre-activation value $s_j = \sum_{i=1}^{n} w_{ji} \cdot x_i + b_j$. Since the weight matrix $W \in \mathbb{R}^{m \times n}$ and bias vector $\mathbf{b} \in \mathbb{R}^m$ are known to the server, they are encoded as plaintexts and never encrypted. The input vector $\mathbf{x}$, however, remains encrypted throughout in ciphertext C_{in}.

For each neuron j, C_{in} is multiplied by the plaintext row $\mathbf{w}_j$ of W and added to the bias b_j, placing the scalar $s_j = \mathbf{w}_j \cdot \mathbf{x} + b_j$ in a dedicated slot $c_j = \mathsf{Enc}(0, \ldots, s_j, \ldots, 0)$. Accumulating over all neurons then yields the output ciphertext $C_{\text{out}} = \sum_{j=1}^{m} c_j = \mathsf{Enc}(s_1, s_2, \ldots, s_m)$, whose j-th slot holds s_j. In practice, this is carried out efficiently via a rotation-based summation [Halevi and Shoup 2014].

Functional Bootstrapping (LUT Evaluation). The activation function f is applied homomorphically via functional bootstrapping: $C_{\text{act}} = \mathsf{FuncBoot}(C_{\text{out}}, \text{LUT}_f)$, where LUT_f is precomputed using trigonometric Hermite interpolation [Alexandru et al. 2025], which encodes the activation function over the discrete domain $\mathbb{Z}_p$. This step is the most computationally intensive part of the pipeline, but serves two purposes simultaneously: it evaluates the non-linear activation $f(\cdot)$ and refreshes the ciphertext noise and multiplicative level, enabling further computations on neural networks. The resulting ciphertext C_{act} is then passed to the next layer.

2.3. Phase III: Decryption and Output Extraction

The final layer of the network is a fully connected layer without an activation function, producing a vector of class scores $\mathbf{s} \in \mathbb{R}^K$. After evaluation, the server performs a scheme-switch from CKKS back to BFV and returns the resulting ciphertext to the client. Since

only the client possesses the secret key, the ciphertext is decrypted locally to recover the plaintext score vector. The predicted label is then obtained by a plaintext $\arg\max$ operation: $\hat{y} = \arg\max_{k \in [K]} s_k$, where s_k denotes the score associated with class k.

Notably, performing the $\arg\max$ after decryption does not weaken the privacy guarantees of the protocol under the standard FHE client–server threat model [Pulido-Gaytan et al. 2020]. The server operates exclusively on encrypted data and never accesses plaintext inputs, intermediate activations, output scores, or predicted labels. Since the client is the authorized recipient of the inference result [Liu et al. 2021], decrypting before the final comparison introduces no additional information leakage to the server. Homomorphically evaluating the $\arg\max$ would only reduce the information disclosed to the client while increasing computational cost. We therefore perform the comparison after decryption to minimize latency while preserving the protocol's privacy guarantees.

3. Preliminary Results and Discussion

We evaluate the inference pipeline on MNIST using a two-layer fully-connected network trained in plaintext and subsequently deployed in the homomorphic neural network. The input layer has $28 \times 28 = 784$ neurons, matching the MNIST image dimensionality. Two hidden widths ($d_h \in \{30, 100\}$) and three activation functions (sign, Heaviside, ReLU) are considered, yielding six configurations. Each is compared against a baseline (*Regular*) that replaces functional bootstrapping with standard CKKS bootstrapping followed by a Chebyshev polynomial approximation of the activation. The computational setup is described in Table 1.

Table 1. Experimental environment.

Component	Specification
CPU	Snapdragon(R) X Elite - X1E78100
Memory	16 GB allocated to WSL2 (host: 32 GB)
OS	Ubuntu 24.04.4 LTS (6.6.114.1-microsoft-standard-WSL2)
Compiler	g++ 13.3.0 (aarch64-linux-gnu)
Build	cmake version 3.28.3
Libraries	OpenFHE 1.5.0

The evaluation subset consists of 50 images drawn from the MNIST test split via stratified sampling, with 5 instances per digit class, ensuring balanced class representation and avoiding distortions. Inputs undergo binarization as a pre-processing step to further reduce computational cost. Since each forward pass corresponds to an independent encrypted computation whose cost is effectively unaffected by batch size, this compact subset is sufficient to obtain representative measurements of accuracy, latency, and memory usage without introducing unnecessary computational overhead. Average latency and memory consumption are reported for each inference stage; for clarity, memory usage is expressed as a delta relative to the preceding stage. Accuracy is assessed against a 50-image baseline from the plaintext model, allowing us to isolate the error introduced by the cryptographic pipeline from the inherent limitations of the trained architecture. Table 2 reports these results.

Table 2. Results for functional bootstrapping and polynomial approximation across different activation functions and hidden-layer widths. (T: time in seconds, Mem: memory in megabytes, expressed as deltas).

(a) Sign Activation (30 Neurons)

	Functional		Regular	
Metric	**T (s)**	**Mem (MB)**	**T (s)**	**Mem (MB)**
Input-encode	–	12318.12	–	8926.19
Layer 1	13.25	-897.26	86.20	+119.01
Boot+Act	49.14	+906.05	20.79	+0.70
Layer 2	2.48	-830.75	23.15	+21.23
Total	**64.87**	**15079.16**	**130.14**	**9322.30**
Accuracy	92.0%		90.0%	
Plain Acc.	96.0%		96.0%	

(b) Sign Activation (100 Neurons)

	Functional		Regular	
Metric	**T (s)**	**Mem (MB)**	**T (s)**	**Mem (MB)**
Input-encode	–	12291.4	–	8949.07
Layer 1	44.24	-800.96	287.30	+93.84
Boot+Act	47.89	+764.31	20.75	+1.69
Layer 2	3.3	-721.16	23.15	+15.40
Total	**95,43**	**15207.48**	**331,2**	**9318.70**
Accuracy	86%		58%	
Plain Acc.	82%		82%	

(c) Heaviside Activation (30 Neurons)

	Functional		Regular	
Metric	**T (s)**	**Mem (MB)**	**T (s)**	**Mem (MB)**
Input-encode	–	12292.84	–	10131.48
Layer 1	15.36	-555.33	93.63	+160.13
Boot+Act	53.94	+472.18	25.67	+2.86
Layer 2	2.78	-457.08	25.58	-0.13
Total	**72.08**	**15219.39**	**144.88**	**10594.23**
Accuracy	92.0%		66.0%	
Plain Acc.	98.0%		98.0%	

(d) Heaviside Activation (100 Neurons)

	Functional		Regular	
Metric	**T (s)**	**Mem (MB)**	**T (s)**	**Mem (MB)**
Input-encode	–	11831.35	–	10129.39
Layer 1	47.46	-790.05	305.92	+131.66
Boot+Act	49.44	+723.99	24.50	+4.62
Layer 2	3.49	-806.76	25.23	+1.15
Total	**100,39**	**15244.85**	**355.65**	**10604.70**
Accuracy	100%		48.0%	
Plain Acc.	100%		100.0%	

(e) ReLU Activation (30 Neurons)

	Functional		Regular	
Metric	**T (s)**	**Mem (MB)**	**T (s)**	**Mem (MB)**
Input-encode	–	12352.54	–	10129.94
Layer 1	14.61	-667.44	95.61	+135.51
Boot+Act	49.42	+764.84	28.48	+4.66
Layer 2	2.68	-707.31	26.16	+3.42
Total	**66.71**	**15241.26**	**150.25**	**10604.35**
Accuracy	78.0%		96.0%	
Plain Acc.	96.0%		96.0%	

(f) ReLU Activation (100 Neurons)

	Functional		Regular	
Metric	**T (s)**	**Mem (MB)**	**T (s)**	**Mem (MB)**
Input-encode	–	12422.39	–	10169.50
Layer 1	47.56	-758.27	301.35	+110.75
Boot+Act	48.18	+678.96	25.10	+10.26
Layer 2	3.54	-729.01	24.54	+3.74
Total	**99,28**	**15242.91**	**350.99**	**10593.04**
Accuracy	84.0%		86.0%	
Plain Acc.	94.0%		94.0%	

The codebase[1] provides a modular CKKS framework for constructing and evaluating homomorphic neural networks. It supports sequences of fully connected layers using functional bootstrapping for activations (enabling a variety of functions within the same network), with configurable input encoding schemes for varied discretization needs. Designed for extensibility, the architecture allows the integration of new layers and functions. To ensure reproducibility, an included evaluation script automates the collection of the accuracy, latency, and memory metrics reported in this work.

[1] `https://github.com/zinho02/func-ckks-neural-networks`

Security Parameters. The CKKS scheme is configured with ring dimension $N = 2^{16}$ and a ternary secret key distribution. The ciphertext modulus Q has $\log_2 Q \approx 1270$ bits, corresponding to approximately 128 bits of security according to OpenFHE [Al Badawi et al. 2022].

Activation Functions and Evaluation Dataset. We evaluate discontinuous, non-polynomial operators commonly encountered in neural network inference. The sign function was adopted in the original FHE-DiNN architecture [Bourse et al. 2018] and remains a standard reference in homomorphic neural network literature. Heaviside serves as a thresholding primitive in binarized networks, while ReLU is the dominant activation in deep learning architectures.

From a homomorphic evaluation perspective, these functions stress the inference pipeline differently: their Chebyshev approximations require higher-degree polynomials (7 versus 3 for sign), increasing modulus consumption and amplifying differences between approximation-based and functional bootstrapping pipelines. Evaluating all three activations allows us to analyze how each backend behaves as approximation complexity increases.

Latency. Comparing across every configuration, the functional pipeline outperforms polynomial approximation by a factor that grows from $2\times$ with $d_h = 30$ to $3.5\times$ at $d_h = 100$. This asymmetry is explained by the cost distribution within a hidden layer: in the regular pipeline, the linear layer dominates because it must operate at a high modulus level ($L = 32$ for ReLU/Heaviside and $L = 29$ for Sign). In the functional pipeline, the linear layer is evaluated at a lower level with depth $L = 28$, folding the activation into the bootstrapping itself.

Accuracy. The functional pipeline matches plaintext accuracy closely for both sign and Heaviside activations, whereas the regular pipeline degrades sharply with hidden width: at $d_h = 100$, the accuracy gap reaches 28% for Sign and 52% for Heaviside. This degradation is likely attributable to the poor suitability of Chebyshev polynomial approximation for discontinuous functions, whose approximation error is further amplified as network width increases and pre-activation values drift outside the polynomial's interpolation domain.

Memory. The latency and accuracy gains of the functional pipeline come with a memory overhead of $1.6\times$ on average. Most of this difference is established at the input-encoding stage, where the BFV-to-CKKS scheme switch, precomputation of Hermite LUT coefficients, and rotation keys required inflate the working set. Another distinction is that in each linear layer, the functional pipeline releases memory as ciphertexts at intermediate levels are consumed, while Boot+Act allocates heavily when refreshing the chain. By contrast, the regular pipeline allocates monotonically across both stages, yielding a smaller per-layer footprint that grows with depth.

Next Steps. Preliminary results suggest several directions for future work. The most natural extension is incorporating convolutional layers to enable more expressive architectures and extend the evaluation to more challenging datasets such as CIFAR-10, allowing a broader comparison between functional bootstrapping and approximation-based pipelines under deeper and more realistic network topologies.

References

Al Badawi, A., Bates, J., Bergamaschi, F., Cousins, D. B., Erabelli, S., Genise, N., Halevi, S., Hunt, H., Kim, A., Lee, Y., Liu, Z., Micciancio, D., Quah, I., Polyakov, Y., R. V., S., Rohloff, K., Saylor, J., Suponitsky, D., Triplett, M., Vaikuntanathan, V., and Zucca, V. (2022). OpenFHE: Open-source fully homomorphic encryption library. *Cryptology ePrint Archive*. Report 2022/915, `https://eprint.iacr.org/2022/915`.

Alexandru, A., Kim, A., and Polyakov, Y. (2025). General functional bootstrapping using ckks. In *Advances in Cryptology – CRYPTO 2025*, volume 16002 of *Lecture Notes in Computer Science*, pages 304–337. Springer.

Bourse, F., Minelli, M., Minihold, M., and Paillier, P. (2018). Fast homomorphic evaluation of deep discretized neural networks. In *Advances in Cryptology – CRYPTO 2018*, volume 10993 of *Lecture Notes in Computer Science*, pages 483–512. Springer.

Cheon, J. H., Kim, A., Kim, M., and Song, Y. (2017). Homomorphic encryption for arithmetic of approximate numbers. In *Advances in Cryptology – ASIACRYPT 2017*, volume 10624 of *Lecture Notes in Computer Science*, pages 409–437. Springer.

Chillotti, I., Gama, N., Georgieva, M., and Izabachène, M. (2020). Tfhe: Fast fully homomorphic encryption over the torus: I. chillotti et al. *Journal of Cryptology*, 33(1):34–91.

Fan, J. and Vercauteren, F. (2012). Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144.

Gentry, C. (2009). *A Fully Homomorphic Encryption Scheme*. PhD thesis, Stanford University.

Gilad-Bachrach, R., Dowlin, N., Laine, K., Lauter, K., Naehrig, M., and Wernsing, J. (2016). Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In Balcan, M. F. and Weinberger, K. Q., editors, *Proceedings of the 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 201–210, New York, NY, USA. PMLR.

Halevi, S. and Shoup, V. (2014). Algorithms in HElib. In *Advances in Cryptology – CRYPTO 2014*, pages 554–571. Springer.

Hesamifard, E., Takabi, H., and Ghasemi, M. (2016). Cryptodl: Towards deep learning over encrypted data. In *Proceedings of the Annual Computer Security Applications Conference (ACSAC)*, Los Angeles, CA, USA.

Liu, B., Ding, M., Shaham, S., Rahayu, W., Farokhi, F., and Lin, Z. (2021). When machine learning meets privacy: A survey and outlook. *ACM Computing Surveys (CSUR)*, 54(2):1–36.

Pulido-Gaytan, L. B., Tchernykh, A., Cortés-Mendoza, J. M., Babenko, M., and Radchenko, G. (2020). A survey on privacy-preserving machine learning with fully homomorphic encryption. In *Latin American High Performance Computing Conference*, pages 115–129. Springer.

Yang, Z., Niu, C., Wei, B., Huang, Z., Hong, C., and Wei, T. (2025). Rboot: Accelerating homomorphic neural network inference by fusing relu within bootstrapping. Cryptology ePrint Archive, Report 2025/1534.