Privacy-Preserving Machine Learning with Homomorphic Encryption: A Complete Benchmark for Medical Image Analysis

Lucas Castro Truppel Machado[1], **Thaís Bardini Idalino**[1]

[1]Universidade Federal de Santa Catarina (UFSC)
Florianópolis – SC – Brazil

lucas.ctm@posgrad.ufsc.br, thais.bardini@ufsc.br

Abstract. *Homomorphic encryption (HE) enables a remote server to perform Convolutional Neural Network (CNN) inference directly on encrypted medical images, but existing benchmarks often underreport deployment costs such as client-side cryptographic overhead and communication volume. This work presents an end-to-end benchmark of CKKS-encrypted CNN inference for binary pneumonia detection on PneumoniaMNIST, evaluated according to the seven-point reporting checklist proposed by Yanez and Yadav (2026). The evaluation covers plaintext-versus-encrypted classification quality, client-side key generation, encryption, and decryption on both a laptop and a Raspberry Pi 3 Model B+, server-side inference latency by operation, communication volume, edge-device energy consumption, and the assumed threat model under a CKKS parameter set providing at least* 128 *bits of classical security. The results show the main deployment barrier is not inference itself, but the one-time provisioning of evaluation keys, whose memory footprint exceeds the resources of typical edge devices and must therefore be delegated to a more capable host.*

1. Introduction

Artificial intelligence has significantly advanced automated medical diagnostics, with CNNs demonstrating remarkable success in detecting pathologies in medical imaging [Litjens et al. 2017]. However, deploying these models as cloud-based services introduces significant privacy risks, as it typically requires hospitals and patients to transmit potentially highly sensitive medical images to untrusted remote servers. Privacy-preserving methods are already established practice over medical data, such as privacy-preserving record linkage (PPRL), which links records referring to the same patient across institutions without revealing identifiers [Gkoulalas-Divanis et al. 2021], but such methods protect identifiers rather than the content under analysis. HE offers a solution to this challenge by allowing the remote server to perform computation, such as CNN inference, directly on encrypted data, returning an encrypted prediction that only the client can decrypt [Gilad-Bachrach et al. 2016].

Despite the theoretical capability of HE-based inference, a gap remains in the privacy-preserving machine learning (PPML) literature: most published benchmarks for encrypted CNNs are incomplete and non-transparent [Yanez and Yadav 2026]. Studies frequently report inference latency and accuracy but critically omit the hidden, yet substantial, costs of real-world deployment. These omitted metrics often include client-side cryptographic overhead (such as key generation, encryption, and decryption times),

client–server communication volume, the energy impact on edge devices, and a full disclosure of the threat model and HE parameter sets. Yanez and Yadav (2026) proposed a seven-point reporting checklist to promote comparable evaluation standards.

To address this gap, our paper presents a complete end-to-end benchmark of CKKS-based homomorphically encrypted CNN inference for binary pneumonia detection on the PneumoniaMNIST dataset [Yang et al. 2023]. By applying this checklist, this work provides a transparent evaluation of client and server costs. Specifically, we evaluate distinct deployment scenarios, contrasting a resource-rich client (a standard modern laptop) against a constrained IoT edge device (a Raspberry Pi), to determine whether encrypted medical-image inference can be practically deployed across two representative real-world settings.

The remainder of this article is organized as follows: Section 2 provides the theoretical background on convolutional neural networks and homomorphic encryption. Section 3 covers related work in encrypted image classification. Section 4 details the experimental methodology, encompassing the network architecture, training protocols, target hardware, and measurement metrics. Section 5 breaks down the specific implementation choices, including single-instruction multiple-data (SIMD) packing and polynomial activation design. Section 6 presents the results and discussion, reporting plaintext-versus-encrypted accuracy alongside comprehensive metrics for client-side overhead, server inference time, communication volume, edge energy consumption, and the assumed threat model. Finally, Section 7 concludes the paper.

2. Background

This section introduces the foundational concepts underlying this work: convolutional neural networks, the class of models targeted by our approach, and homomorphic encryption, which enables computation over encrypted data.

2.1. Convolutional neural networks

Convolutional neural networks represent a category of deep neural networks, frequently applied in the analysis of visual data [Yamashita et al. 2018]. The fundamental building block of a CNN is the convolutional layer, which transforms an input tensor into an output tensor by applying a set of learnable weights, known as filters. A single filter is composed of multiple 2D kernels, one for each input channel. Given an input tensor X with C_{in} channels and a weight tensor K containing C_{out} filters (one per output channel), and a kernel size L, the k-th filter has dimensions $L \times L \times C_{in}$. The output feature map Y for the k-th output channel at spatial position (i, j) is computed as:

$$Y_{i,j,k} = \sum_{c=0}^{C_{in}-1} \sum_{m=0}^{L-1} \sum_{n=0}^{L-1} X_{i+m,j+n,c} \cdot K_{m,n,c,k} + b_k$$

where b_k is the learnable bias for the k-th filter [Krichen 2023].

By sliding these 3D filters across the input tensor, the network aggregates spatial information, capturing simple patterns like edges in early layers and complex structures in deeper layers. Following the convolution, each element of the tensor is passed through a non-linear function, known as an activation function. This enables the network

to learn complex patterns, rather than being simply a linear combination of the inputs [Li et al. 2022].

Furthermore, the neural network incorporates pooling layers, which reduce the spatial dimensions of the feature maps by applying a filter that extracts the dominant features from each region, typically via maximum or average operations, thereby decreasing the number of learnable parameters [Yamashita et al. 2018]. Finally, after a sequence of these blocks, the CNN terminates in a fully connected layer, which connects all neurons from the preceding layer to generate the final output [Krichen 2023].

2.2. Homomorphic Encryption

Homomorphic encryption denotes a class of cryptographic schemes that allow operations to be performed directly on ciphertexts, such that decrypting the result yields the same value as applying the corresponding operations to the underlying plaintexts [Acar et al. 2018]. HE schemes are commonly classified according to the operations they support. Partially homomorphic encryption supports an unbounded number of operations of a single type, either additions or multiplications, but not both. Somewhat homomorphic encryption supports both additions and multiplications, but with a limited number of operations before noise growth prevents correct decryption. Fully homomorphic encryption (FHE) removes this limitation through a ciphertext-refreshing procedure known as bootstrapping, enabling the evaluation of arbitrarily deep circuits [Acar et al. 2018].

Gentry's construction, based on ideal lattices, was the first fully homomorphic encryption scheme [Gentry 2009]. Later schemes improved the practicality of FHE by targeting different computational settings. BGV and BFV are schemes based on the Ring Learning With Errors (RLWE) problem, designed for exact arithmetic over encrypted integers and are commonly used with SIMD batching, which allows multiple plaintext slots to be processed within a single ciphertext [Marcolla et al. 2022]. In contrast, the Fast Fully Homomorphic Encryption over the Torus (TFHE) scheme represents messages over the torus and relies on fast programmable bootstrapping, making it particularly suitable for boolean circuits, bit-level operations, and nonlinear functions [Marcolla et al. 2022].

The Cheon-Kim-Kim-Song (CKKS) scheme differs from integer HE schemes such as BGV and BFV by supporting approximate arithmetic on encoded real or complex values. This makes CKKS particularly suitable for floating-point computations and, consequently, for many machine learning applications. CKKS is defined over the cyclotomic ring $R = \mathbb{Z}[X]/(X^N + 1)$, with N a power of two, and a single ciphertext packs $N/2$ complex slots. Homomorphic additions and multiplications act slot-wise in a SIMD manner, while rotations cyclically permute the slots [Liu et al. 2025]. This packing structure is central to efficient neural-network evaluation, since it enables feature maps to be processed in parallel within a single ciphertext.

3. Related Work

Practical homomorphic CNN inference is commonly traced back to CryptoNets, which performed leveled HE evaluation of a small network on MNIST and established server-side latency and message sizes as central cost metrics [Gilad-Bachrach et al. 2016]. Much of the subsequent literature has continued to focus primarily on server-side performance. Rovida and Leporati (2024) introduced a low-memory packing strategy for evaluating

ResNet-20, reporting inference latency and peak memory footprint, but not client-side cryptographic operations or communication cost. Choi et al. (2024) and Kim et al. (2024), with UniHENN and HyPHEN respectively, report per-image latency, but no client-side timings, no energy consumption, and no explicit threat-model discussion. Recent packing and implementation optimizations include tile-tensor packing [Aharoni et al. 2023], throughput-oriented multi-image batch packing [Xu and Wang 2025, Ye et al. 2026], and GPU-accelerated encrypted convolutions [Slama et al. 2025], yet none measures the client workload on a constrained edge device.

Wang et al. (2026), with PipFHE, breaks down server-side latency into convolution and bootstrapping components and provides a complete CKKS parameter set, but its measurements remain restricted to server-side costs. Kim et al. (2022) reports client-side encryption and decryption timings together with input and result ciphertext sizes for an HE-based action-recognition CNN, but does not measure energy consumption or evaluate the client workload on a constrained edge device. The size of the evaluation keys that a client must generate and store is itself a recognized limitation for constrained devices [Cheon et al. 2026].

These omissions reflect a difference in purpose, since the referenced works mainly focus on optimizations, whereas this work measures what a full deployment costs when the client and server are real devices. Therefore, taken together, these studies leave no single benchmark that covers all seven items specified by Yanez and Yadav (2026): client-side cryptographic overhead; communication cost; (server) cryptographic operation costs; ciphertext expansion and packing strategy; energy and battery impact (edge deployments); security level and parameterization; threat model specification. In particular, prior CNN-inference benchmarks do not report edge-device energy. The remainder of this article addresses this gap for CKKS-based binary pneumonia detection on PneumoniaMNIST, reporting all seven items, together with standard classification metrics, using a workstation server and two client tiers: a modern laptop and a Raspberry Pi 3 Model B+.

4. Methodology

This section specifies the experimental methodology employed to benchmark CKKS-encrypted CNN inference for pneumonia detection from chest radiographs, covering the dataset, the network architecture, the training and fine-tuning procedures, the hardware platforms and software libraries, and the measurement protocol and metrics.

4.1. Dataset

We use PneumoniaMNIST, the chest X-ray dataset of the MedMNIST v2 collection [Yang et al. 2023]. Each sample is a 28×28 grayscale image labeled as either normal or pneumonia (binary classification). The official split provided by the MedMNIST distribution is preserved without modification: 4708 images for training, 524 for validation, and 624 for testing, totaling 5856 chest radiographs.

4.2. CNN Architecture

We use a compact convolutional neural network with three main blocks, as shown in Figure 1. The input is a grayscale $1\times28\times28$ image, which is padded to 32×32 before being passed through the network. The first block contains two 3×3 convolutional layers with

32 output channels, each followed by batch normalization and a ReLU activation (the rectified linear unit, defined as $\mathrm{ReLU}(x) = \max(0, x)$). A 2×2 average pooling layer then reduces the spatial resolution to 16×16. The second block follows the same design but increases the number of channels to 64 and reduces the resolution to 8×8. The third block uses a single 3×3 convolution with 128 channels, followed again by batch normalization and ReLU. The resulting 128×8×8 feature map is processed by global average pooling, dropout with probability 0.3, and a final linear layer that produces two logits. The predicted class is obtained by taking the argmax over these logits, that is, the index of the largest entry in the logit vector.

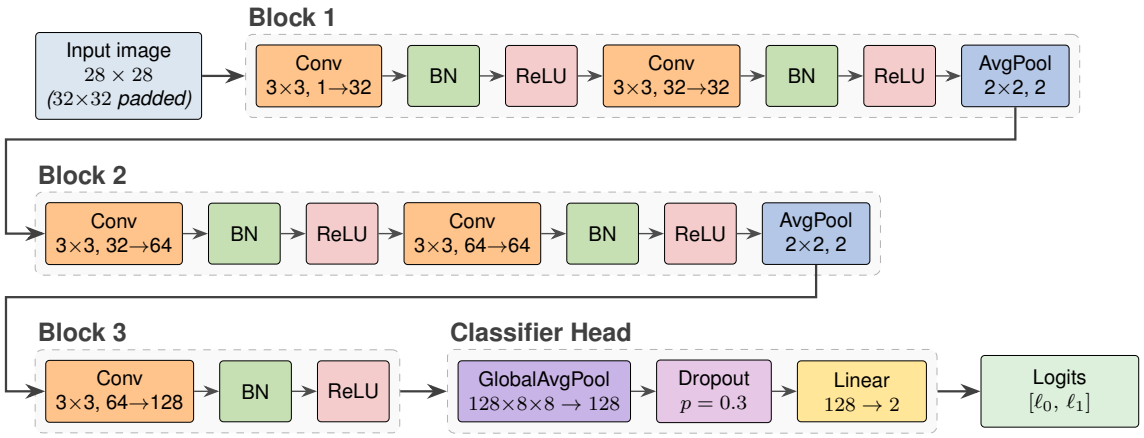

Figure 1. CNN architecture diagram.

To make the baseline network compatible with CKKS evaluation, we apply several modifications following Rovida and Leporati (2024). First, dropout is removed since it is only used during training, and each batch normalization layer is folded into the previous convolution. Next, we replace each ReLU with a degree-27 Chebyshev polynomial approximation over the $[-1, 1]$ interval. Since the polynomial approximation is valid only over this range, we rescale the inputs to each activation. For each layer l, we profile the pre-activation values of the trained model using all images from the training and validation splits of the dataset. We define the scale factor s_l from the maximum of the absolute pre-activation values, multiplied by a safety factor of 1.1, a conservative heuristic margin rather than a tuned value, intended to cover unseen inputs that exceed the profiled maximum. This scale is folded into the weights and biases of the convolution preceding the activation, analogously to batch-normalization folding.

4.3. Training

The baseline network was trained from scratch on the training split using cross-entropy loss. We used AdamW with an initial learning rate of 10^{-3}, weight decay of 10^{-4}, a batch size of 128, and trained the model for 30 epochs with a cosine learning-rate schedule. Input images were normalized with mean and standard deviation equal to 0.5 and zero-padded from 28×28 to 32×32 so that the spatial dimensions are powers of two, as required by the encrypted pipeline's packing layout. The only data augmentation was a random horizontal flip during training. After each epoch, the model was evaluated on the validation split, and the checkpoint with the highest validation AUC was selected as the baseline.

Replacing each ReLU with a degree-27 Chebyshev approximation introduces a small accuracy loss. To mitigate this loss, the HE-friendly network was fine-tuned using knowledge distillation from the trained baseline model. Fine-tuning used the same training split, but without data augmentation. We used AdamW with learning rate 10^{-4}, weight decay 10^{-4}, batch size 128, and trained for 30 epochs with a cosine learning-rate schedule. The loss followed the standard Hinton knowledge-distillation formulation, with temperature $T = 5$. Model selection was again based on validation AUC.

4.4. Hardware Platforms and Software Libraries

The benchmark is executed across three distinct hardware platforms representing a server and two tiers of clients. Operating as the server is a workstation equipped with an AMD Ryzen 9 9900X processor (12 cores, 24 threads, 4.4–5.6 GHz), 98.5 GB of DDR5 RAM, and NVMe storage, running Linux Ubuntu 24.04. A laptop serves as the high-tier client, featuring a 12th Gen Intel Core i7-1255U processor (10 cores, 12 threads, 3.50–4.70 GHz), 16 GB of RAM, and Ubuntu 24.04.

Representing the low-tier client is an edge device, a Raspberry Pi 3 Model B+ single-board computer with a 1.4 GHz 64-bit quad-core processor and 1 GB of RAM, running Raspberry Pi OS. To assess the energy footprint of the edge device, an Exbom UTS-10 USB multimeter is connected inline with the Raspberry Pi's power supply to measure its instantaneous voltage, current, and cumulative energy consumption during benchmark execution. For the software implementation, model training is conducted using PyTorch 2.11.0, while all CKKS operations utilize OpenFHE 1.5.1.

4.5. Measurement protocol

All measurements are taken under run conditions chosen to minimize variance. Background workloads are suspended at measurement time; the CPU frequency governor is fixed to performance; the laptop is on AC power. Each timing or energy measurement is repeated 100 times, and we report the mean, median, variance, and minimum–maximum range. The 624-image test set is the evaluation set for all metrics. For the encrypted inference accuracy, the full test set is run once on the workstation; for the per-image latency, inference is repeated 100 times for the same image.

4.6. Metrics

Classification quality is evaluated on the full 624-image test set for three model variants. The baseline CNN is the original PyTorch model with ReLU activations and batch normalization, and serves as the baseline reference. The HE-friendly variant, still evaluated in plaintext, applies the modifications required for CKKS compatibility (batch-normalization folding, polynomial activation replacement, and scale folding) and isolates the accuracy loss attributable to these architectural changes alone, independently of any encryption noise. Finally, the encrypted CKKS evaluation of the HE-friendly variant captures the additional impact of approximate arithmetic under homomorphic encryption.

We treat the pneumonia class as positive and report the four entries of the confusion matrix, namely true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN). From these counts we derive the accuracy, the sensitivity (recall), and the specificity, together with the per-class and macro-averaged F_1 scores, the latter

being the unweighted mean of the two per-class values. We also report the area under the receiver operating characteristic curve (AUC), a threshold-independent measure.

Client-side cryptographic overhead is measured as the wall-clock time required for secret-key generation, input encryption, and output decryption, separately on the laptop and on the Raspberry Pi. Server-side inference cost is measured as the wall-clock time of one encrypted forward pass on the workstation, further broken down by layer block to quantify the contribution of convolution, polynomial activation, pooling, and bootstrapping. We report the mean, median, variance, and minimum–maximum range. For the Raspberry Pi, we also report the energy spent per operation. Communication volume is reported as the total provisioning payload transmitted once per client during the setup phase and the per-inference exchange.

5. Implementation

The complete source code for this work is open-source and publicly available[1]. The remainder of this section details the encrypted forward pass, covering the ciphertext packing strategy and the convolution, pooling, and activation primitives.

5.1. Ciphertext Expansion and Packing Strategy

This subsection addresses the *packing strategy* checklist item. The encrypted forward pass is implemented in C++ using OpenFHE 1.5.1, following the optimized vector encoding strategy introduced by Rovida and Leporati (2024). The slots are arranged as $[c_0, c_1, \ldots, c_{C-1}]$ in channel-major order, where each c_i stores the $L \times L$ pixels of channel i in row-major order. The channel stride is therefore the padded spatial size: $32 \times 32 = 1{,}024$ at the input of Block 1, $16 \times 16 = 256$ from Block 2 onwards, and $8 \times 8 = 64$ from Block 3 onwards. Under this packing configuration, a single input ciphertext occupies 27.3 MB on disk. Relative to the padded 32×32 8-bit image (1024 bytes), this represents an expansion factor of approximately 2.7×10^4.

5.2. Convolutions

Each 3×3 convolution follows the procedure of Rovida and Leporati (2024) and is implemented in three stages. First, nine spatially shifted views of the input ciphertext are produced, one for each kernel position. The center position reuses the input directly. The four axial positions are obtained by rotations of ± 1 and $\pm L$ slots, where L is the row stride of the current block. The four diagonal positions are obtained by composing a column rotation with a row rotation, so the full 3×3 window is covered using only the four axial rotation keys $\{+1, -1, +L, -L\}$.

Second, the inner kernel is repeated C_{out} times, once per iteration c. On each iteration, the nine shifted ciphertexts are multiplied by nine weight plaintexts and summed. Because all input channels share a single ciphertext, one ciphertext–plaintext multiplication applies its weights to every input channel simultaneously. The weight plaintexts are not flat: they are packed so that, on iteration c, the slot range of each input channel carries the kernel weights for a different output channel, and the assignment cycles through all output channels as c grows. After the nine-term sum, the slot range of input channel c_{in}

[1] `https://github.com/LucasTruppel/he-cnn-pneumonia`

therefore holds one of the C_{in} partial products that will eventually combine into a specific output channel, where the choice of output channel depends on both c and c_{in}.

Third, the per-iteration partial sums are folded into a running accumulator that is rotated by $-L^2$ slots after each iteration. The rotation slides every input-channel block one slot range further along the ciphertext. Combined with the cyclic weight assignment from the previous step, it makes every partial product destined for output channel j, though produced on different iterations and in different input-channel slot ranges, converge to the same slot range and accumulate there. After all C_{out} iterations, the slot range $[jL^2,\ (j+1)L^2)$ holds the complete convolution output for channel j, with the sum over input channels already done. The bias plaintext, encoded under the same layout, is added at the end.

5.3. Average Pooling and Activation Function

The remaining encrypted primitives are simpler. A 2×2 average-pooling layer replaces each group of four neighboring pixels by its mean. Under the row-major slot packing, the four pixels in each window are obtained from the original ciphertext and from rotations by $+1$ for the right neighbor, $+L$ for the lower neighbor, and $+L+1$ for the lower-right neighbor. Adding these rotated ciphertexts to the original produces a 2×2 sum at every slot, including the overlapping windows discarded by the stride-2 output. We combine the scaling factor and stride selection into a single plaintext mask. The mask contains 0.25 at slots corresponding to valid stride-2 output positions and 0 elsewhere. A single ciphertext–plaintext multiplication therefore divides the sums by four and zeros out the slots that the next layer will not read. Thus, the full averaging step consumes only one multiplicative level.

The polynomial activation function replaces ReLU with a degree-27 Chebyshev approximation, obtained by sampling the function at the 28 Chebyshev nodes of the interval $[-1, 1]$ and constructing the unique interpolating polynomial of degree 27 through these points, expressed in the Chebyshev basis. At inference time, it is evaluated by OpenFHE's `EvalChebyshevSeries`.

6. Results and Discussion

This section reports the experimental results. Each subsection covers one checklist item, except for the first one, Classification metrics: Plaintext vs. Ciphertext, which is included only to establish the accuracy reference for the encrypted pipeline.

6.1. Classification metrics: Plaintext vs. Ciphertext

Table 1 reports the classification metrics on the full 624-image test set for the three model variants. The conversion from the baseline to the HE-friendly network introduces a small but measurable degradation. AUC decreases by 4.8×10^{-3}, from 0.9649 to 0.9601, and accuracy drops by 0.48 percentage points, from 85.42 to 84.94. The confusion matrix indicates that this loss occurs in the normal class: three images previously classified as normal are reassigned to pneumonia, reducing specificity while leaving sensitivity unchanged.

The transition from the HE-friendly network to its encrypted CKKS execution is, in contrast, essentially free from a practical standpoint. AUC changes by only 1×10^{-5},

Table 1. Classification metrics on the full 624-image PneumoniaMNIST test set.

	Baseline CNN	**HE-friendly CNN**	**Encrypted (CKKS)**	**Δ HE-friendly vs. Baseline**	**Δ Encrypted vs. HE-friendly**
AUC	0.9649	0.9601	0.9601	−0.0048	+0.00001
Accuracy	85.42	84.94	84.94	−0.48	0
Sensitivity	99.49	99.49	99.49	0	0
Specificity	61.97	60.68	60.68	−1.28	0
F_1 (macro)	0.8281	0.8216	0.8216	−0.0065	0
F_1 (pneumonia)	0.8950	0.8920	0.8920	−0.0031	0
F_1 (normal)	0.7612	0.7513	0.7513	−0.0098	0
TP/FP/TN/FN	388/89/145/2	388/92/142/2	388/92/142/2	0/ + 3/ − 3/0	0/0/0/0

and every other reported metric, including the full confusion matrix, is identical to the plaintext HE-friendly model. This confirms that the CKKS parameters and bootstrapping placement preserve sufficient precision for the entire forward pass: the approximation noise introduced by encrypted evaluation does not change a single one of the 624 results. The accuracy gap between the baseline and the encrypted model is therefore almost entirely a property of the HE-friendly architecture itself, not of homomorphic evaluation.

Across the encrypted-inference literature the datasets and architecture differ, and the comparable quantity is the degradation with respect to each work's own plaintext model: 0.93 percentage points for Rovida and Leporati (2024) on CIFAR-10, and between 0.20 and 0.87 percentage points for PipFHE. Our measured degradation of 0.48 points is consistent with these reported results. The exceptions are HyPHEN and UniHENN, which avoid this degradation by training with quadratic activations from the start instead of approximating an already trained ReLU.

6.2. Encrypted Inference Latency

Table 2 reports the wall-clock time of the encrypted forward pass on the workstation, decomposed by operation type and aggregated across all instances of each type in the circuit. Figure 2 illustrates the network pipeline and reports the mean execution time of each layer. The mean total inference latency is 88.0 s, with a standard deviation of 1.9 s and a minimum–maximum range of 85.7–96.5 s.

Table 2. Per-operation inference timings (s) for the encrypted PneumoniaMNIST CNN, aggregated by operation type and measured over 100 iterations.

Type	**Layers**	**Mean (s)**	**Median (s)**	**Min (s)**	**Max (s)**	**Std. Dev. (s)**	**% of total**
Convolution	5	48.855	48.640	47.016	52.634	1.135	55.5
Bootstrapping	6	30.070	29.921	29.017	33.853	0.891	34.2
Average pool + repack	2	3.442	3.432	3.346	3.852	0.083	3.9
Activation function	5	3.172	3.149	2.997	3.509	0.100	3.6
Global average pool	1	2.373	2.365	2.289	2.761	0.067	2.7
Fully connected	1	0.095	0.094	0.084	0.134	0.007	0.1
Slot masking	2	0.023	0.022	0.018	0.035	0.004	0.0
Total inference	—	88.034	87.693	85.682	96.506	1.912	100.0

Convolutions are the dominant cost, accounting for 55.5% of total latency. The five convolutional layers consume 48.9 s in aggregate, corresponding to an amortized cost of approximately 9.8 s per layer. Each convolution combines a large number of ciphertext–plaintext multiplications, which consume multiplicative levels. It also requires cyclic rotations, and these rotations rely on key switching, which is a major source of cost. Average pooling and global pooling also rely on plaintext multiplications, but at a much smaller scale.

Bootstrapping is the second-largest contributor, accounting for 34.2% of total latency. The six bootstrapping invocations take 30.1 s in total, or approximately 5.0 s each. Although an individual bootstrap is cheaper than an individual convolution, bootstrapping is invoked more frequently. Together, convolution and bootstrapping account for 89.7% of total inference latency.

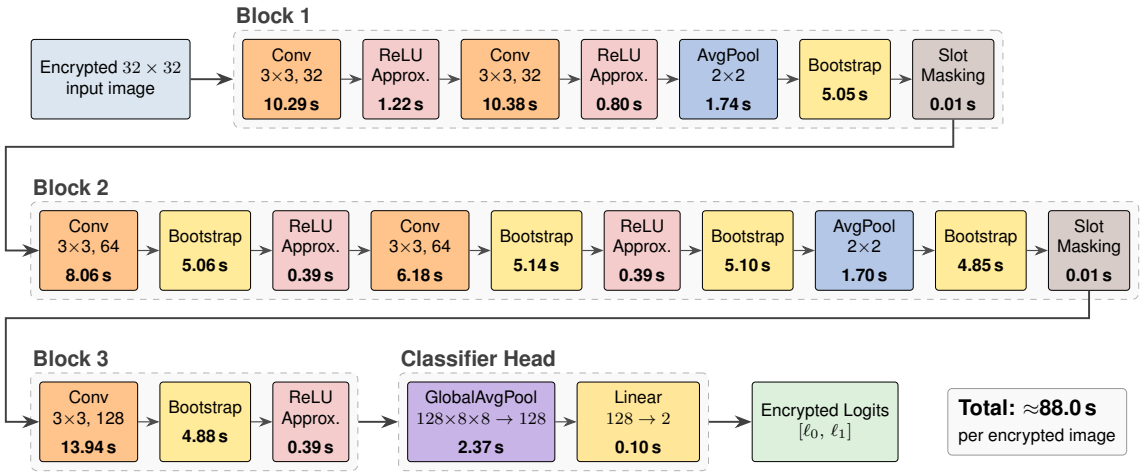

Figure 2. Encrypted inference pipeline with per-layer latency.

The remaining operations together account for only 10.3% of total latency. Average pooling and the slot repacking that follows it cost 3.4 s (3.9%), the five polynomial activations cost 3.2 s (3.6%), and the single global-average pool costs 2.4 s (2.7%). The polynomial activation cost is worth noting: despite the degree-27 Chebyshev approximation, the five activation layers together cost less than a single bootstrap invocation. The fully connected layer at the network output (0.095 s) and slot masking (0.023 s) contribute 0.1% of total inference time combined.

Direct comparisons with existing literature are difficult, as reported latencies depend heavily on network depth, hardware specifications, batch sizes, and the use of bootstrapping. Rovida and Leporati (2024), whose encoding is adopted in this work, take 260 to 336 s for the deeper ResNet-20, running single-threaded on a laptop. HyPHEN evaluates that same ResNet-20 in 37.6 s, faster than the shallower network used here, but with 64 threads, and in 1.40 s once the circuit is moved to a GPU. UniHENN requires 740 s for LeNet-5 on 1 thread. PipFHE reports 36.9 s per image, but amortized over a batch of 227 images, whereas the 88.0 s above is the latency of a single radiograph.

Peak resident memory during inference reaches 29.75 GiB on the workstation. Since the workstation used in this evaluation has 91.7 GiB of RAM, the peak corresponds to approximately 33% of total memory. However, because servers must handle

concurrent requests in practical deployments, scaling this system will require allocating approximately 30 GiB of memory per active request.

6.3. Client-Side Cryptographic Overhead and Energy Consumption

Table 3 reports the wall-clock times and peak physical memory usage of the client-side cryptographic operations on the two hardware tiers. For both clients, we measure the operations: secret-key and public-key generation, encryption of a 32×32 input image, and decryption of the two output logits. For the laptop, we additionally measure the four key-generation steps that produce the bundle transmitted to the server: the relinearization key, the rotation key bundle required by the CNN circuit, and the two automorphism key sets used by bootstrapping at slot counts of 16384 and 8192. These four steps are not measured on the Raspberry Pi because the bootstrapping-key generation alone exceeds the device's 1 GB of RAM by more than $13\times$.

Table 3. Client-side cryptographic operation timings (ms), peak memory (MiB), battery charge consumption (mAh), average power (W), and energy per iteration (J) across hardware tiers measured over 100 iterations.

Client	Operation	Mean Time (ms)	Median Time (ms)	Std. Dev. Time (ms)	Min Time (ms)	Max Time (ms)	Peak RAM (MiB)	Total Charge (mAh)	Avg Power (W)	Energy per Iter (J)
High-tier (Laptop)	KeyGen (sk/pk)	132.6	133.6	3.6	124.6	143.0	184.1	–	–	–
	Encrypt (image)	135.8	135.2	3.9	128.6	148.2	274.2	–	–	–
	Decrypt (logits)	82.0	81.6	2.8	74.9	90.1	274.2	–	–	–
	KeyGen (relin.)	221.9	222.3	4.9	212.2	230.6	313.7	–	–	–
	KeyGen (CNN rot.)	2,810.3	2,674.4	412.9	2,474.6	3,855.3	3,387.6	–	–	–
	KeyGen (Boot-16384)	21,773.8	13,607.1	23,038.2	13,202.8	87,087.0	13,856.0	–	–	–
	KeyGen (Boot-8192)	12,560.7	11,859.6	2,167.7	11,008.5	18,304.4	13,531.5	–	–	–
Low-tier (Raspberry Pi)	KeyGen (sk/pk)	1,414.5	1,416.4	39.3	1,357.4	1,482.3	183.6	25	3.2	4.5
	Encrypt (image)	2,158.6	2,161.4	20.6	2,094.4	2,191.4	270.9	36	3.0	6.5
	Decrypt (logits)	1,441.7	1,442.5	6.1	1,395.2	1,452.1	270.9	34	4.3	6.2

Note: Total charge (mAh), average power (W), and energy per iteration (J) were measured only on the Raspberry Pi, powered by a 5.03 V supply.

On the high-tier laptop, all three shared operations complete in well under 150 ms, with encryption the most expensive and decryption the lightest. The standard deviation stays below 4% of the mean for every operation, indicating stable measurements. The per-inference operations on the client, encrypting an input and decrypting the corresponding output, total approximately 218 ms, a negligible overhead relative to the server-side encrypted forward pass reported in Subsection 6.2. On the low-tier Raspberry Pi, the same operations take roughly an order of magnitude longer, so each inference adds approximately 3.60 s for input encryption and output decryption.

The cross-platform slowdown is not uniform across operations. The Raspberry Pi is $10.7\times$ slower than the laptop for key generation, $16.0\times$ slower for encryption, and $17.6\times$ slower for decryption. The practical implication is that encrypted inference remains feasible on the edge device, but the client-side overhead is shifted from milliseconds to seconds.

Peak physical memory usage shows a platform-independent behavior. KeyGen (sk/pk) peaks at approximately 184 MiB on both platforms, while encryption and decryption peak at approximately 270 MiB. This result establishes a practical lower bound on the memory required by a client device during inference: roughly 275 MiB of free RAM must

be available during encryption and decryption. The Raspberry Pi 3 Model B+ used in this evaluation, with 1 GB of RAM, satisfies this requirement, with the peak corresponding to approximately 27% of total memory. Devices with substantially less available memory, however, would require changes to the parameter set to run the protocol.

The evaluation-key generation steps, measured only on the laptop, are substantially more expensive than the previous operations and have a very different memory profile. The relinearization key is the cheapest component, requiring 221.9 ms and peaking at 313.7 MiB, which is comparable to a single encryption in both time and memory. The rotation-key bundle for the CNN circuit takes 2.81 s and peaks at 3.39 GiB, due to the generation of all rotation indices used by the forward pass. The two bootstrapping key sets dominate the setup cost: KeyGen Boot-16384 takes 21.77 s on average and peaks at 13.86 GiB, while KeyGen Boot-8192 takes 12.56 s and peaks at 13.53 GiB. Boot-16384 also shows the largest dispersion of any operation in the table, with a standard deviation of 23.04 s, a median of only 13.61 s, and a maximum of 87.09 s. The gap between the mean and median is consistent with paging as the working set approaches the laptop's physical memory limit.

From a device-requirements perspective, the 13.86 GiB peak is the limiting factor for the client. It is roughly fifty times larger than the per-inference memory floor of 275 MiB and prevents devices with less than approximately 14 GiB of free RAM from generating the provisioning bundle locally. This creates a clear asymmetry: a device may be able to perform the per-inference cryptographic operations, but still be unable to generate the evaluation keys. For this reason, the Raspberry Pi must obtain the evaluation-key bundle from a more capable device. However, the client only requires evaluation keys during protocol initialization. Alternatives to reduce client-side key sizes may include running smaller CNNs, reducing the degree of the activation function, or using key management systems such as KG$^+$ and BTS$^+$ [Cheon et al. 2026].

We next consider the energy cost of the workload on the Raspberry Pi. Energy consumption was measured at the USB power input of the edge device using an inline USB power meter operating at a nominal supply voltage of 5.03 V. Key generation was the least energy-intensive operation, consuming 4.5 J per execution. Encryption had the highest energy cost, at 6.5 J per operation, mainly because it had the longest execution time. However, its average instantaneous power draw, 3.0 W, was the lowest among the three operations. Decryption completed in approximately one-third less time than encryption, but showed the highest average power consumption, at 4.3 W. This resulted in an energy cost of 6.2 J per operation, comparable to that of encryption.

6.4. Cryptographic Parameters and Security Level

The parameters are chosen to satisfy the depth requirement of the encrypted forward pass described in Subsection 6.2, while maintaining at least 128 bits of classical security. The cyclotomic ring dimension is fixed at $N = 2^{16} = 65{,}536$, yielding $N/2 = 32{,}768$ slots per ciphertext. The CKKS scaling configuration uses a 52-bit first modulus and a 47-bit per-level scaling modulus. The remaining parameters are reported using OpenFHE's named constants: the rescaling technique is `FLEXIBLEAUTO`, key switching uses the `HYBRID` variant with three large digits, the secret-key distribution is `SPARSE_TERNARY`, and bootstrapping is configured with a level budget of $(3, 3)$.

The multiplicative depth of the cryptographic context was set to 25 levels based on empirical evaluation. This budget is divided into 13 levels available to the inference circuit between two consecutive bootstraps and an internal bootstrap depth of 12 levels. The maximum level observed during the forward pass occurs at the output of the average-pooling operation between blocks 2 and 3, where it reaches level 19. The level budget of $(3, 3)$ is the largest configuration that keeps the total ciphertext modulus within the standard security cap for $N = 2^{16}$.

6.5. Communication Overhead

In a practical deployment of CKKS-based encrypted inference, the client retains the secret key and the plaintext input, while the server holds the model weights and performs the encrypted forward pass. The homomorphic circuit is determined by the model architecture and packing strategy, which determine the CKKS parameters required for evaluation. In our deployment model, the network, the OpenFHE version, and the key-switching technique are fixed as an artifact published once and used by all clients. As a result, the parameters needed to instantiate the scheme are client-side constants, and parameter negotiation between the client and server is reduced to a local policy check.

Given these parameters, the client generates the secret key and the public evaluation material required by the server. This material comprises automorphism keys for the 26 circuit rotations determined by the packing strategy, together with the automorphism keys required by bootstrapping, whose indices are derived internally by OpenFHE from the slot count and level budget. In total, the server receives three rotation-key groups: one for the CNN circuit and two for bootstrapping, corresponding to the 16384-slot bootstraps in blocks 1–2 and the 8192-slot bootstrap in block 3. These groups occupy approximately 2.7 GB, 10.8 GB, and 9.4 GB, respectively, amounting to a combined rotation-key payload of about 23 GB. The remaining material, namely the relinearization key and the serialized cryptographic context, adds less than 110 MB. The public key is omitted because the server evaluates only client-provided ciphertexts and does not encrypt fresh data. This payload is transmitted once per client and reused across inferences, unless the secret key, model architecture, or CKKS parameters change.

After this setup step, each inference involves the exchange of only two ciphertexts: one sent by the client, encoding the 32×32 input image, and one returned by the server, encoding the two encrypted logits. At the chosen parameters, the input ciphertext is approximately 27.3 MB at the top of the modulus chain, while the output ciphertext is approximately 3.1 MB at the bottom of the chain. Thus, the 23 GB handshake is amortized over the lifetime of the secret key.

To translate these volumes into wall-clock transfer times, we consider two representative network configurations: a 100 Mbps low-tier access link and a high-tier fiber link at 1 Gbps. Ignoring protocol overhead, the 23 GB provisioning payload requires approximately 30.7 minutes at 100 Mbps and 3.1 minutes at 1 Gbps. The per-inference exchange of approximately 30.4 MB, by contrast, takes about 2.4 s and 0.24 s, respectively. Even on the slower link, the per-inference transfer adds only a few seconds to the server-side latency reported in Subsection 6.2. In a real-world deployment, a clinic pairs its X-ray station with an edge device, installs the key bundle once, and thereafter submits each radiograph when it is acquired. After a few hundred images, the one-time provision-

ing is no longer the dominant client cost, and a prediction returns in roughly a minute and a half, which is compatible with use during a medical consultation.

6.6. Threat Model

We adopt the honest-but-curious adversary model commonly used in Machine Learning as a Service (MLaaS) deployments [Tanuwidjaja et al. 2020]. The client is trusted and holds the secret key, the plaintext input image, and the decrypted prediction. The server is assumed to execute the encrypted forward pass correctly and return the corresponding output ciphertext, but it may inspect, store, and analyze any ciphertext, evaluation key, or intermediate value processed during inference. The protected assets are the input radiograph and the prediction derived from it. The model weights belong to the server and are never transmitted. The evaluated circuit is not secret, since the architecture, the packing strategy, and the CKKS parameters are public artifacts.

Under these assumptions, confidentiality follows from the indistinguishability under chosen-plaintext attack (IND-CPA) security of the RLWE-based CKKS scheme at the selected parameter set [Hwang et al. 2025]. We note that CKKS can be vulnerable to passive key-recovery attacks when decrypted approximate results are shared with third parties [Liu et al. 2025]. This threat is not exercised in our setting, since decryption is performed exclusively by the client, who does not release decrypted results to any external party. As shown in Subsection 6.3, a low-tier device cannot generate the evaluation keys locally from the secret key. In our setting, the client owns a second, more capable device that performs this step. The secret key is transferred between the client's own devices over an authenticated and confidential channel and never leaves the client's control.

7. Conclusion

This article presented a complete end-to-end benchmark of CKKS-encrypted CNN inference for binary pneumonia detection on PneumoniaMNIST, structured according to the seven-point reporting checklist proposed by Yanez and Yadav (2026), with each item reported in a dedicated subsection. The goal was to provide a transparent baseline for the practical cost of CKKS-based medical-image inference.

The cost of deploying the protocol on a low-tier client is concentrated in its initialization phase. This pattern is consistent with the gigabyte-scale evaluation keys reported for CKKS [Cheon et al. 2026], while its magnitude for encrypted CNN inference on medical images is measured here. The evaluation-key bundle peaks at 13.86 GiB during generation, well above the 1 GB available on the Raspberry Pi 3 Model B+, and must therefore be produced on a more capable host. Once provisioned, the client-side encryption/decryption overhead is modest: 218 ms on a laptop and 3.60 s on the Raspberry Pi. In terms of utility, the pipeline loses 0.48 percentage points of accuracy and 4.8×10^{-3} AUC relative to the baseline, with the loss arising from the HE-friendly conversion rather than from CKKS evaluation. The encrypted and plaintext HE-friendly models produce the same confusion matrix on the test set, with an AUC difference of only 1×10^{-5}.

Future work includes repeating the benchmark with a smaller CNN whose evaluation keys fit in the Raspberry Pi memory, so that the whole client side runs on the edge device, as well as evaluating key management systems such as KG$^+$ and BTS$^+$ [Cheon et al. 2026].

References

Acar, A., Aksu, H., Uluagac, A. S., and Conti, M. (2018). A survey on homomorphic encryption schemes: Theory and implementation. *ACM Computing Surveys*, 51(4).

Aharoni, E., Adir, A., Baruch, M., Drucker, N., Ezov, G., Farkash, A., Greenberg, L., Masalha, R., Moshkowich, G., Murik, D., Shaul, H., and Soceanu, O. (2023). HeLayers: A Tile Tensors Framework for Large Neural Networks on Encrypted Data. *Privacy Enhancing Technology Symposium (PETs) 2023*.

Cheon, J. H., Kang, M., and Park, J. H. (2026). Towards Lightweight CKKS: On Client Cost Efficiency. In *Proceedings of the ACM Asia Conference on Computer and Communications Security*, ASIA CCS '26, pages 1387–1398, New York, NY, USA. Association for Computing Machinery.

Choi, H., Kim, J., Kim, S., Park, S., Park, J., Choi, W., and Kim, H. (2024). UniHENN: Designing faster and more versatile homomorphic encryption-based CNNs without im2col. *IEEE Access*, 12:109323–109341.

Gentry, C. (2009). Fully homomorphic encryption using ideal lattices. In *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*, STOC '09, pages 169–178, New York, NY, USA. Association for Computing Machinery.

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, New York, USA. PMLR.

Gkoulalas-Divanis, A., Vatsalan, D., Karapiperis, D., and Kantarcioglu, M. (2021). Modern privacy-preserving record linkage techniques: An overview. *IEEE Transactions on Information Forensics and Security*, 16:4966–4987.

Hwang, I., Min, S., Seo, J., and Song, Y. (2025). On the security and privacy of CKKS-based homomorphic evaluation protocols. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 296–330. Springer.

Kim, D., Park, J., Kim, J., Kim, S., and Ahn, J. H. (2024). HyPHEN: A hybrid packing method and its optimizations for homomorphic encryption-based neural networks. *IEEE Access*, 12:3024–3038.

Kim, M., Jiang, X., Lauter, K., Ismayilzada, E., and Shams, S. (2022). Secure human action recognition by encrypted neural network inference. *Nature Communications*, 13(1):4799.

Krichen, M. (2023). Convolutional neural networks: A survey. *Computers*, 12(8):151.

Li, Z., Liu, F., Yang, W., Peng, S., and Zhou, J. (2022). A survey of convolutional neural networks: Analysis, applications, and prospects. *IEEE Transactions on Neural Networks and Learning Systems*, 33(12):6999–7019.

Litjens, G., Kooi, T., Bejnordi, B. E., Setio, A. A. A., Ciompi, F., Ghafoorian, M., Van Der Laak, J. A., Van Ginneken, B., and Sánchez, C. I. (2017). A survey on deep learning in medical image analysis. *Medical Image Analysis*, 42:60–88.

Liu, W., You, L., Shao, Y., Shen, X., Hu, G., Shi, J., and Gao, S. (2025). From accuracy to approximation: A survey on approximate homomorphic encryption and its applications. *Computer Science Review*, 55:100689.

Marcolla, C., Sucasas, V., Manzano, M., Bassoli, R., Fitzek, F. H. P., and Aaraj, N. (2022). Survey on fully homomorphic encryption, theory, and applications. *Proceedings of the IEEE*, 110(10):1572–1609.

Rovida, L. and Leporati, A. (2024). Encrypted image classification with low memory footprint using fully homomorphic encryption. *International Journal of Neural Systems*, 34(05):2450025.

Slama, M. F., Guerrouache, H., Challal, Y., Benatchba, K., and Baghdadi, R. (2025). Lightening encrypted convolutions: A GPU-optimized approach to private inference. In *2025 IEEE Conference on Communications and Network Security (CNS)*, pages 1–9.

Tanuwidjaja, H. C., Choi, R., Baek, S., and Kim, K. (2020). Privacy-preserving deep learning on machine learning as a service—a comprehensive survey. *IEEE Access*, 8:167425–167447.

Wang, T., Ye, Z., Huang, T., Wang, C., Ying, K., and Huang, K. (2026). PipFHE: Resource-efficient privacy-preserving deep CNN inference via padded batch packing and channel merging over FHE. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2026(1):1–25.

Xu, C. and Wang, Y. (2025). Optimization of CNN inference for multi-image with fully homomorphic encryption. *Journal of King Saud University Computer and Information Sciences*, 37(9):260.

Yamashita, R., Nishio, M., Do, R. K. G., and Togashi, K. (2018). Convolutional neural networks: an overview and application in radiology. *Insights into Imaging*, 9(4):611–629.

Yanez, P. and Yadav, N. (2026). Homomorphic encryption for secure healthcare artificial intelligence. *Discover Artificial Intelligence*, 6(1):200.

Yang, J., Shi, R., Wei, D., Liu, Z., Zhao, L., Ke, B., Pfister, H., and Ni, B. (2023). MedMNIST v2-A large-scale lightweight benchmark for 2D and 3D biomedical image classification. *Scientific Data*, 10(1):41.

Ye, Z., Wang, T., Huang, T., Li, Y., Wang, C., Cheung, R. C., and Huang, K. (2026). HTCNN: High-throughput batch CNN inference with homomorphic encryption. *IEEE Transactions on Dependable and Secure Computing*, 23(2):2400–2411.