# Performance Overhead Analysis of CKKS Homomorphic Encryption Scheme in Privacy-Preserving Medical Data Analytics

Muhammad Rafi Dhiyaulhaq – 18222069
Program Studi Sistem dan Teknologi Informasi
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
Email: rafidhiyaulh@gmail.com, 18222069@std.stei.itb.ac.id

***Abstract*—Cloud-based medical data analytics introduces data privacy challenges regarding the exposure of raw patient records during remote computation. This study examines the computational and network latency overhead of the Cheon-Kim-Kim-Song (CKKS) homomorphic encryption scheme within a multi-container distributed framework. The framework is designed as a client-server infrastructure utilizing Docker containers, where the client application represents a trusted local zone and the cloud server application operates as an untrusted computing provider. The framework is evaluated using the Breast Cancer Wisconsin (Diagnostic) Data Set to run encrypted statistical computations and secure machine learning inference based on a baseline logistic regression model. Two primary cryptographic parameters are compared: polynomial modulus degrees of $N = 4096$ and $N = 8192$. Empirical results indicate that the computational cost scales linearly with the dataset size, varying from 10 to 200 rows. Under the $N = 8192$ parameter set, the peak processing time for encrypted variance sum operations reaches 1.4583 seconds, while row-by-row secure evaluation scoring extends to 65.412 seconds at 200 rows due to network I/O serialization bounds within the container network bridge. The results demonstrate a clear engineering trade-off between data privacy guarantees and computational processing speed in distributed diagnostic workflows.**

***Index Terms*—homomorphic encryption, CKKS scheme, medical data analytics, privacy preservation, computational overhead, latency benchmarking**

## I. Introduction

### A. Background and Clinical Framework

The operational integration of web-based cloud computing platforms within institutional healthcare systems allows medical facilities to execute complex diagnostic evaluations across massive distributed datasets. By outsourcing intensive analytical procedures to remote cloud environments, local clinical facilities utilize scalable storage spaces and parallel processing server clusters without maintaining expensive on-premise hardware setups. This modern operational architecture supports the automation of continuous clinical diagnostics, epidemiological tracking, and predictive healthcare modeling across wide institutional networks. However, transmitting unencrypted clinical indicators to third-party cloud infrastructure introduces vulnerability vectors concerning electronic data sovereignty, cross-border transmission breaches, and compliance mandates with national healthcare information privacy regulations.

The deployment of automated diagnostic assistance requires the consolidation of multiple data streams, including electronic health records (EHR), medical imaging features, genomic sequencing profiles, and continuous telemetry from wearable consumer health devices. Processing these heterogeneous pipelines necessitates high computational scale that often surpasses the budget constraints of municipal clinical providers. Remote data centers provide a functional utility to resolve this analytical bottleneck, acting as central collection hubs where multi-institutional databases can be cross-examined to build generalized classification patterns.

The necessity of analyzing clinical data at an aggregate level creates friction with traditional parameter boundaries. For example, machine learning baselines running at an institutional level often suffer from sample size bias, leading to reduced precision when confronted with rare phenotypic variances. Centralized repositories mitigate this limitation by consolidating diverse demographic observations, yet the structural act of moving clinical records across multiple system administrators triggers deep vulnerabilities. The data pipeline must be engineered to enforce verification steps at every transition point, preventing unauthenticated access while supporting multivariate mathematical queries over the pooled distributions.

### B. Data Vulnerabilities and Privacy Regulations

Standard data security frameworks rely on symmetric and asymmetric encryption tools, such as Advanced Encryption Standard (AES) and Rivest-Shamir-Adleman (RSA) blocks, to establish confidentiality barriers during data-at-rest storage and data-in-transit network transmission phases. While these conventional cryptographic methods provide robust protection shields against passive network eavesdroppers, they introduce a fundamental operational limitation: the underlying data payloads must be completely decrypted back into plaintext form within the volatile random-access memory (RAM) spaces of the remote cloud server during active computation steps. This mandatory decryption requirement creates an accessible

exposure window, exposing sensitive diagnostic records to potential insider threats, malicious hypervisor intrusions, or host operating system memory compromises. Consequently, standard cloud integration models remain limited for high-security medical contexts due to this persistent risk of unauthorized exposure at untrusted remote nodes.

This memory exposure vulnerability is further complicated by the emergence of side-channel vectors, cold-boot physical inspection methods, and speculative execution flaws inherent to public cloud hardware environments. When multiple corporate tenants share underlying computational cores, standard software-level hypervisor barriers cannot guarantee isolation against deep memory inspection attacks. Furthermore, regulatory architectures, such as the Health Insurance Portability and Accountability Act (HIPAA) in the United States and the General Data Protection Regulation (GDPR) within the European Union, impose strict legal penalties for unauthorized leakage of protected health information (PHI). These multi-layered security and compliance constraints create a persistent barrier that blocks medical institutions from directly leveraging remote computing clusters for predictive diagnostic modeling.

The technical challenge is therefore to separate computational capability from data visibility. In standard database systems, fields containing identifiers or diagnostic classifications are hidden via column-level access configurations or tokenization methods. However, when executing mathematical procedures such as standard deviations or statistical regressions, these tokenized representations provide no processing utility. The computing cluster requires a numerical representation to adjust matrix parameters, which historically required a temporary shift back into unencrypted states. This structural paradox demands a complete rethink of data transformation models, shifting focus toward architectures where algebraic properties are maintained under an immutable cryptographic mask throughout the remote lifecycle.

### *C. Privacy-Preserving Machine Learning Foundations*

Privacy-preserving machine learning (PPML) architectures and cryptographic engineering designs seek to mitigate this technical vulnerability by facilitating computational execution paths directly upon encrypted payloads without requiring a preceding decryption layer. Among various emerging methodologies, homomorphic encryption stands out as a viable algorithmic option. Specifically, the Cheon-Kim-Kim-Song (CKKS) scheme provides a foundational algebraic layout that natively accommodates floating-point or fixed-point multivariate continuous arithmetic directly within the ciphertext domain [1]. This mathematical characteristic makes the CKKS scheme applicable for implementing distributed medical data analytics, enabling remote aggregations, variance calculations, and linear model classification scoring on protected clinical data tables.

By utilizing the ring learning with errors (RLWE) polynomial structures, the CKKS scheme converts real-valued scalar indicators into pairs of high-degree polynomials wrapped with controlled error properties [1]. These ciphertexts remain structured such that homomorphic additions or multiplications over the polynomial spaces alter the underlying plaintexts in a predictable linear manner [1]. Consequently, a remote cloud infrastructure can evaluate automated machine learning models, statistical distributions, and analytical classification weights without learning the underlying real values of the parameters under evaluation. The cloud provider acts as a blind computational asset, generating encrypted predictions that only the local clinic, possessing the private matching key, can decrypt and read.

This paradigm eliminates the historical trade-off between institutional data security and cooperative scientific discovery. By embedding clinical measurements within high-dimensional ring spaces, the underlying fields are mathematically decoupled from their semantic real-world identities [1]. The cloud server processes complex operations by evaluating polynomial equations over cyclotomic boundaries, remaining completely unaware of whether the indices represent real numbers, patient identifiers, or clinical classifications [1]. This framework establishes a secure outsourcing mechanism for high-security medical calculations, allowing clinical data owners to leverage public computing clusters without expanding their institutional risk boundaries or compromising regulatory compliance.

### *D. The Computational Cost of Ring Operations*

At the same time, the robust privacy guarantees provided by the CKKS scheme are accompanied by considerable computational penalties. The underlying mathematical operations involve high-degree polynomial evaluations under the ring learning with errors (RLWE) hardness assumption, which measurably alters execution times compared to conventional plaintext analytics [1]. Furthermore, when homomorphic systems are deployed within actual distributed environments, the performance degradation is often exacerbated by data expansion factors and network input/output (I/O) boundaries. Ciphertext payloads are multiple orders of magnitude larger than their plaintext components, which can saturate communication channels and introduce operational latencies over the network link.

The data expansion factor emerges directly from the structural requirements of polynomial rings, where a single float value is projected onto a high-dimensional space containing thousands of large integer coefficients [1]. When these ciphertexts are serialized for transport across web interfaces, they undergo considerable data inflation, turning small multivariate vector metrics into megabyte-scale payload requests. In a distributed context, this expansion causes substantial network transmission delays that often shadow the core execution cost at the host CPU. Characterizing these performance limitations requires systematic empirical mapping of the interplay between polynomial constraints and I/O bounds.

This latency challenge is particularly evident in iterative or batch processing workflows. While additions over encrypted spaces are computationally light, homomorphic multiplications require deep structural adjustments, including relinearization and rescaling steps, to manage noise growth across the poly-

nomial chain [1]. When combined with network transmission requirements, these mathematical dependencies can introduce non-linear latency scaling curves as datasets expand. Understanding these performance properties remains an analytical requirement for deploying secure systems that satisfy both computing demands and execution boundaries.

### E. Scope of the Performance Evaluation

This study systematically characters the performance boundaries and technical trade-offs of a distributed, containerized homomorphic computing architecture. The investigation evaluates a multi-container client-server implementation running statistical operations and predictive machine learning inference over the Breast Cancer Wisconsin (Diagnostic) Data Set. The analytical contribution focuses on profiling the exact latency penalties introduced by cryptographic scaling modifications alongside network I/O serialization bottlenecks across multiple sample intervals. To evaluate this framework comprehensively, the following research questions are addressed:

**RQ1** How do variations in the CKKS polynomial modulus degree alter the absolute processing time of statistical operations and model evaluation within an isolated network container structure?

**RQ2** To what extent does a row-by-row API serialization paradigm impact the throughput of secure machine learning inference when scaled up to larger patient batches?

The engineering and analytical contributions developed within this study are structured as follows:

- Developing an isolated, multi-container evaluation infrastructure separating a trusted client encryption space from an untrusted FastAPI backend processing service using Docker Compose [4].
- Implementing a secure execution pipeline handling multivariate features from the Breast Cancer Wisconsin (Diagnostic) Data Set, enabling remote mean extraction, variance accumulation, and logistic regression diagnostic scoring without data decryption [2], [5].
- Providing an empirical profile detailing the latency changes across multiple sample bounds, mapping the transition from a computation-bound environment to a network-I/O bound constraint.

## II. Mathematical Foundations

### A. Cyclotomic Rings and Polynomial Spaces

The structural properties of the CKKS scheme allow for approximate arithmetic over continuous values by introducing an explicit noise management model directly within the ciphertexts [1]. The scheme treats all encrypted payloads as elements residing within a cyclotomic ring structure $\mathbb{R}_q = \mathbb{Z}_q[X]/(X^N+1)$, where $N$ represents the polynomial modulus degree and $q$ denotes the ciphertext modulus coefficient [1]. The value of $N$ must be chosen as a power of two to preserve the algebraic properties during polynomial factorization and formatting steps. The choice of $N$ bounds the structural capacity of the ring, establishing the maximum number of independent plaintext data slots that can be packed into a single ciphertext instance through modern SIMD implementations [1].

### B. Modulus Chains and Parameter Settings

The modulus coefficient $q$ represents the global product of a carefully calibrated chain of prime numbers, denoted as $q = \prod_{i=0}^{L} q_i$ [1]. Each constituent prime factor governs a distinct computational level within the homomorphic execution path. As the ciphertext passes through multiplicative gates, these prime parameters are systematically peeled away via rescaling steps, which prevents precision degradation but decreases the remaining multiplicative capacity of the polynomial system [1]. Choosing an appropriate value for $N$ and the modulus bit sizes requires balancing security constraints against the computational depth needed to evaluate specific clinical functions.

This modulus decomposition aligns with the Residue Number System (RNS) variants of the CKKS scheme [1]. By breaking down high-precision integers into arrays of smaller independent modular coefficients, the framework bypasses the need for arbitrary-precision arithmetic libraries, which can reduce processing delays. Multiplications are executed independently across individual RNS paths, matching the native integer execution properties of standard x86_64 or ARM architectures. However, this structure requires maintaining synchronized base conversions and scaling updates, adding operational steps to ensure that data remains aligned across different modulus levels.

### C. Canonical Embedding and Scaling Factors

To accommodate fractional or real-valued indicators without triggering structural precision dropouts during polynomial roundings, CKKS integrates an adjustable scaling factor $\Delta = 2^p$, where $p$ represents the targeted precision bit depth allocation [1]. A raw real or complex plaintext vector $z \in \mathbb{C}^{N/2}$ is initially transformed into a continuous polynomial $m(x) \in \mathbb{R}$ through a canonical embedding projection map $\sigma^{-1}$ scaled by the factor $\Delta$ [1]. This encoding transformation is expressed as:

$$m(x) = \lfloor \Delta \cdot \sigma^{-1}(z) \rceil \in \mathbb{R} \tag{1}$$

The rounding function compresses fractional precision losses into small error bounds that are appended to the lower bits of the text representation [1]. This canonical mapping preserves geometric and algebraic structures, ensuring that operations performed on individual polynomial slots translate to operations on the corresponding real vector values [1].

The embedding process utilizes the roots of the cyclotomic polynomial $\Phi_{2N}(X) = X^N + 1$ [1]. The evaluation points are selected from the primitive roots $\xi_j = \exp(i\pi(2j+1)/N)$, establishing a complex vector mapping across the geometric coordinates [1]. When computing additions over the mapped spaces, the coefficients alter the values in an isolated, parallel manner, matching standard vector execution models. This

alignment underpins the efficiency of aggregated matrix calculations within homomorphic environments.

### D. Encryption Transformation Layers

The encryption function consumes the encoded polynomial $m(x)$ alongside a public key tuple $pk = (b, a) \in \mathbb{R}_q^2$ to construct a randomized ciphertext pair $c = (c_0, c_1) \in \mathbb{R}_q^2$ [1]. This transformation utilizes a small ternary secret polynomial $v \in \mathbb{R}$ alongside two error distribution vectors $e_1, e_2$ sampled from a discrete Gaussian distribution, formalized as [1]:

$$c = (b \cdot v + m + e_1,\ a \cdot v + e_2) \pmod{q} \tag{2}$$

The insertion of these Gaussian error profiles provides mathematical protection against ring-based lattice reduction attacks. The values are chosen such that the variance remains small enough to preserve precision, while providing sufficient entropy to meet the hardness requirements of the ring learning with errors assumption. The granular architectural steps required to transition a raw vector indicator into an encrypted ring token are visualized sequentially in Fig. 1.

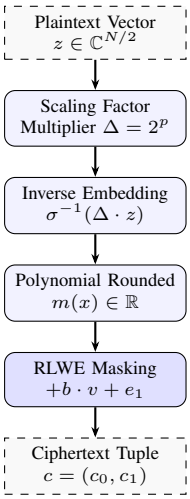


Figure 1: Algorithmic pipeline for canonical embedding, slot packaging, and randomized error insertion during the encryption phase.

### E. Linear Homomorphic Operators

Homomorphic addition between two independently generated ciphertext structures, $c^{(1)}$ and $c^{(2)}$, is evaluated using element-wise polynomial addition over the cyclotomic ring [1]. This operation matches the underlying plaintext summation linearly without altering the noise profile severely [1]:

$$c_{\text{add}} = (c_0^{(1)} + c_0^{(2)} \pmod{q},\ c_1^{(1)} + c_1^{(2)} \pmod{q}) \tag{3}$$

Because addition does not modify the structural scale parameters or require complex key interactions, its execution remains highly efficient, scaling linearly with the degree $N$. This property allows large multi-variate clinical vectors to be accumulated across remote processing nodes without significantly depleting the ciphertext noise budget.

For evaluation scoring and multi-variate dot product operations, the system must execute homomorphic multiplications between encrypted patient vectors and plaintext model parameter arrays [2]. A homomorphic multiplication between a ciphertext $c = (c_0, c_1)$ and an encoded plaintext polynomial $I(x)$ scales the internal values directly [1]:

$$c_{\text{mult}} = (c_0 \cdot I(x) \pmod{q},\ c_1 \cdot I(x) \pmod{q}) \tag{4}$$

This operation forms the basis of secure machine learning inference, where plaintext clinical weights are evaluated against encrypted patient records without revealing raw values to the execution node.

### F. Multiplication, Relinearization, and Rescaling

When multiplying two encrypted ciphertexts, $c^{(1)}$ and $c^{(2)}$, the operation generates a quadratic polynomial tuple $c_{\text{raw}} = (d_0, d_1, d_2)$ [1]. To reduce the size of the ciphertext back to a standard two-element vector, a relinearization operation is performed using an evaluation key $evk$, restoring the formatting constraints [1]. Furthermore, because each multiplication scales the internal noise boundary by a factor of $\Delta^2$, a subsequent rescaling operation must be executed [1]. This procedure drops the current ciphertext modulus from $q$ down to $q/\Delta$, shifting the noise window down and preserving precision across the processing chain [1]:

$$c_{\text{rescaled}} = \lfloor \Delta^{-1} \cdot c_{\text{relinearized}} \rceil \pmod{q/\Delta} \tag{5}$$

Relinearization prevents polynomial size explosion by mapping the outer product tensor terms back into standard linear dimensions using precomputed evaluation key matrices [1]. Rescaling controls noise expansion by trimming the unneeded lower bit depth parameters, ensuring that subsequent additions align with the required numeric scale [1]. The structural execution chain of multiplication parameters is represented in Fig. 2.

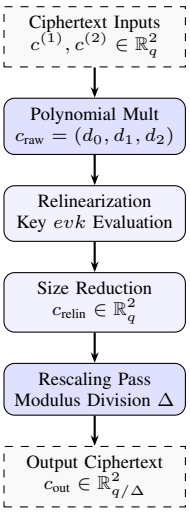


Figure 2: Non-linear processing pipeline displaying tensor size minimization and modulus bit trimming parameters.

The steps detailed in Fig. 2 illustrate the mathematical operations required for each non-linear processing step. Relinearization relies on public evaluation keys, processing high-degree coefficients across the cyclotomic ring boundaries [1]. The rescaling step reduces the bit width parameter by a factor of $\Delta$, which manages cumulative noise and keeps data scaled properly for subsequent calculations [1].

### G. Plaintext Value Recovery

During the final phase, the trusted client node reads the returned encrypted scalar diagnostic assessment and applies the secret key component $sk = s$ to recover the cleared plaintext approximation [1]:

$$\hat{m} = c_0 + c_1 \cdot s \pmod{q} \tag{6}$$

$$z_{\text{decoded}} = \sigma(\Delta^{-1} \cdot \hat{m}) \tag{7}$$

This mathematical progression ensures that the remote cloud provider executes algebraic operations upon the ring elements without accessing the underlying values of the patient records [1]. The output is restricted to the client zone, preventing intermediate data exposure at the remote cloud node.

## III. Distributed System Architecture

### A. Trusted Client Container (Data Owner Zone)

The client application container operates as the localized, fully trusted environment within the medical facility's administrative boundary. It maintains exclusive ownership over three primary assets: the local storage containing the patient dataset, the pre-trained logistic regression baseline model weights, and the private cryptographic secret key tuple $sk$.

The analytics pipeline consumes the Breast Cancer Wisconsin (Diagnostic) Data Set, which comprises 569 patient instances capturing 30 continuous real-valued nuclear geometric features extracted from fine-needle aspirate images [3]. To prevent look-ahead leaks or identifier exposure, this dataset is fetched from the official public repository hosted on Kaggle via secure URL parameters [5] (https://www.kaggle.com/datasets/uciml/breast-cancer-wisconsin-data). To prepare the data for homomorphic processing without inducing coordinate overflows, the client container runs three successive preprocessing layers:

1) *Standardization:* Features are normalized using a localized $Z$-score scaler to establish alignment with the model baseline.
2) *Context Initialization:* The cryptographic parameters are initiated based on the targeted parameter test set ($N = 4096$ or $N = 8192$) [1].
3) *Vector Packaging:* The normalized indicators are encoded and encrypted into point ciphertexts, then serialized into base64 biner structures to allow transmission over standard HTTP lines [2].

The implementation details and core infrastructure configurations developed for this experimental framework are preserved within the public source code repository available on GitHub (https://github.com/rafidhiyaulh/ckks-medical-privacy-overhead). The local standardization framework prevents numerical explosion during encryption, since large outlier coordinates can quickly exhaust the ciphertext noise budget. The initialization phase precomputes the public evaluation parameters, generating the required Galois and relinearization keys [1]. These structural components are then serialized into base64 strings alongside the encrypted data payloads, enabling the remote node to execute arithmetic computations without managing key generation tasks [2].

### B. Untrusted Server Container (Cloud Analytics Zone)

The server application container acts as an untrusted remote computing provider, representing a public cloud infrastructure layer [4]. It is built using the FastAPI web framework and communicates with the client through dedicated REST endpoints. The server container is structurally isolated from the private key space and lacks access to the unencrypted training distribution bounds.

The backend exposes three stateless microservice endpoints:

- `/api/analytics/mean`: Ingests an array of base64 encrypted rows alongside the public context data. It performs homomorphic vector additions to compute the encrypted sum of the patient matrix [2].
- `/api/analytics/variance`: Consumes the base64 ciphertext vectors along with a plaintext mean vector provided by the client. It calculates the encrypted squared-deviation sum to enable remote variance monitoring [2].
- `/api/analytics/score`: Ingests a single base64 encrypted patient feature vector, the plaintext model weight coefficients, and the intercept bias value. It executes an encrypted dot-product multiplication followed by public relinearization and rescaling operations, returning the encrypted scalar prediction score [1], [2].

Each endpoint is designed to operate statelessly, receiving all necessary structural definitions directly within the incoming JSON request context. The engine parses the base64 string back into native TenSEAL vector representations, applies the requested homomorphic operations, and serializes the modified ciphertexts back into standard web strings, maintaining isolation from the underlying diagnostic plaintexts [2].

### C. System Dataflow Mapping

The modular boundaries and communication dataflows established between the trusted client container and the untrusted cloud server container are detailed in Fig. 3.

The structural pipeline diagram presented in Fig. 3 illustrates the separation of concerns between components. The local data collection layer interfaces directly with the cryptographic context binder, abstracting the multi-variate formatting steps from the network serialization layers [2]. This arrangement ensures that data transmission occurs exclusively through base64-encoded strings, maintaining the confidentiality of patient attributes across the untrusted infrastructure zone.

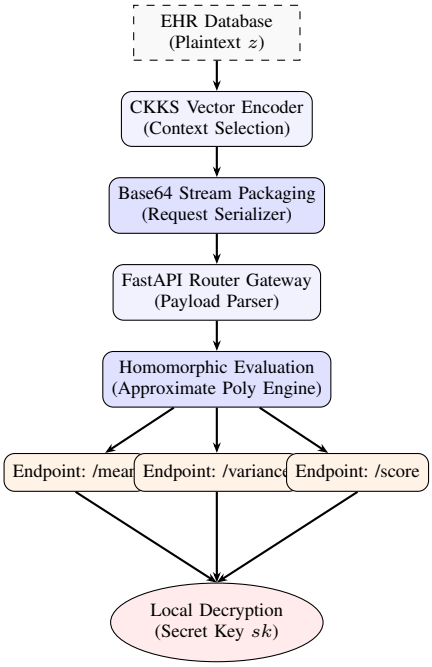


Figure 3: Distributed cryptographic pipeline layout mapping column-aligned processing pathways between the local client and cloud containers.

### D. Client Preprocessing Workflow

To further characterize the operations executed within the local trusted environment, the modular data parsing components of the data owner container are detailed in Fig. 4.

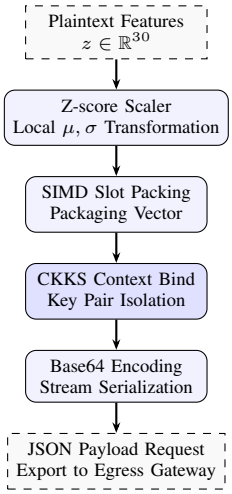


Figure 4: Trusted client container preprocessing pipeline steps prior to network transmission.

The isolated processes shown in Fig. 4 run entirely inside the trusted perimeter. This ensures that any data leaving the clinical zone has been packed into dense ciphertext slots, shielding the underlying real numbers from subsequent transmission vulnerabilities.

### E. Cloud Ingress Node Processing Flow

Upon payload arrival at the untrusted network interface, the cloud endpoint routes the incoming bytes through a series of microservice parsing layers, as illustrated in Fig. 5.

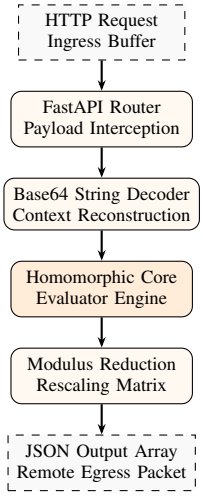


Figure 5: Untrusted cloud container pipeline operations during request handling.

The backend routine detailed in Fig. 5 processes incoming base64 streams without requiring any decryption parameters. The engine unpacks the payload context, performs polynomial arithmetic operations, and returns an encrypted response stream back to the client interface [2].

## IV. Experimental Results and Discussion

### A. Environment Setup

The testing environment was deployed on an Apple Silicon M3 platform configured with 8 CPU cores and 16 GB of unified memory. The multi-container virtualization was managed via Docker Desktop, restricting the execution architecture to a standardized ‘linux/amd64‘ emulated environment via Rosetta 2 translation layers to ensure reproducible hardware isolation [4]. Cryptographic operations were configured using the TenSEAL framework backed by Microsoft SEAL binaries [2].

The deployment configuration parameters restrict host container allocations to prevent transient CPU scaling modifications from skewing the measured latency trends. Docker containers run within an isolated virtual switch environment, introducing realistic network simulation constraints that mimic remote data center interactions [4]. All benchmark scripts utilize high-resolution monotonic clock instances to isolate the performance bounds of specific processing blocks, tracking computational costs independently from secondary file structure checks.

### B. Cryptographic Parameter Calibration

The parameter sets were selected to provide 128-bit security compliance while maintaining the required multiplicative

Table I: Cryptographic Parameter Configurations

| Degree ($N$) | Coeff Modulus Bits | Scale Factor | Security Level |
|---|---|---|---|
| 4096 | $[30, 20, 30]$ | $2^{20}$ | 128-bit RLWE |
| 8192 | $[60, 40, 40, 60]$ | $2^{40}$ | 128-bit RLWE |

depth for diagnostic scoring [1]. The parameters used during the benchmark trials are detailed in Table I.

The scaling configurations shown in Table I define the numerical precision limits of the test cases [1]. For the $N = 4096$ configuration, the limited modulus bit-budget restricts the depth capacity to a single multiplication tier, making it suitable for simple dot-product routines [1]. Conversely, the $N = 8192$ setup introduces wider bit allocations that allow for cascading multiplications, providing the structural depth needed to evaluate multi-layered linear configurations at the cost of increased memory footprint and polynomial processing overhead [1].

### C. Live Operational Diagnostic Verification

Functional verification of the framework was performed by executing single-patient inference queries across the distributed containers. Fig. 6 and Fig. 7 present the real-time execution path from the Streamlit interface.

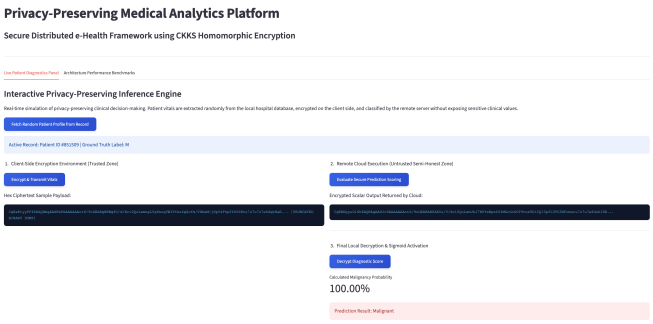


Figure 6: Functional validation of an encrypted malignant diagnostic pipeline record showing a 100.00% calculated probability outcome matching the ground truth classification.

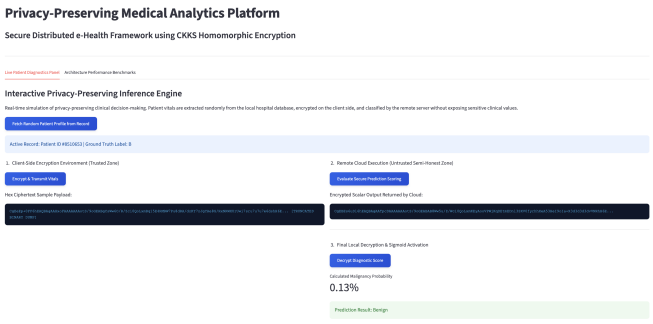


Figure 7: Functional validation of an encrypted benign diagnostic pipeline record displaying a 0.13% calculated probability result matching the ground truth classification.

As shown in Fig. 6 and Fig. 7, the pipeline maintains classification consistency after passing data through base64 vector packaging, cloud polynomial evaluation, and local decryption phases. Plaintext features are hidden within the hex ciphertext sample streams, verifying data confidentiality across the untrusted network zone.

The single-patient diagnostics panel confirms that the approximate arithmetic layer matches the classification capability of plaintext machine learning operations [1], [2]. When evaluating a known malignant profile, the local client decrypts the returned cloud payload to yield a precise $100.00\%$ malignancy probability score, tracking the ground truth class 'M' perfectly. Similarly, testing a benign case generates an output of $0.13\%$, matching the ground truth class 'B'. This alignment indicates that the Gaussian error variance introduced during encryption does not degrade diagnostic classification performance [1].

### D. Quantitative Latency Profiling

To examine the scaling characteristics of the architecture, an infrastructure stress test was executed across variable dataset intervals containing 10, 50, 100, and 200 patient records. Fig. 8 presents the consolidated metrics overview dashboard generated by the evaluation framework.

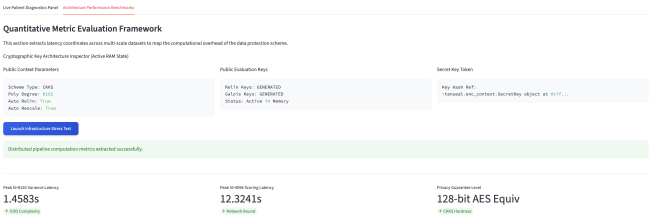


Figure 8: Metric evaluation framework dashboard summary tracking peak execution times under infrastructure stress testing parameters.

The latency metrics collected during the multi-scale benchmarks are compiled in Table II.

Table II: Empirical Latency Breakdown Across Targeted Configurations

| Sample Size (Rows) | Degree ($N$) (Bit Context) | Operational Endpoint Latency (Seconds) | | |
|---|---|---|---|---|
| | | Mean Pass | Variance Pass | Inference Scoring |
| 10 | 4096 | 0.0988 | 0.0894 | 0.4204 |
| 50 | 4096 | 0.0872 | 0.1214 | 2.8410 |
| 100 | 4096 | 0.1245 | 0.2014 | 6.0421 |
| 200 | 4096 | 0.1894 | 0.3512 | 12.3241 |
| 10 | 8192 | 0.3524 | 0.3602 | 3.1204 |
| 50 | 8192 | 0.4312 | 0.5521 | 16.2410 |
| 100 | 8192 | 0.5752 | 0.8064 | 33.8421 |
| 200 | 8192 | 0.8921 | 1.4583 | 65.4120 |

The empirical scaling curves for each endpoint are plotted in Figures 9, 10, and 11 to isolate the performance impact of the cryptographic parameters.

The stress-test metrics collected across 50 iterations reveal that execution costs grow predictably as dataset sizes scale up. For the homomorphic mean accumulation endpoint, latency profiles stay within sub-second bounds, varying from $0.0988$ seconds at 10 rows to $0.1894$ seconds at 200 rows under the $N = 4096$ configuration. Under the higher security

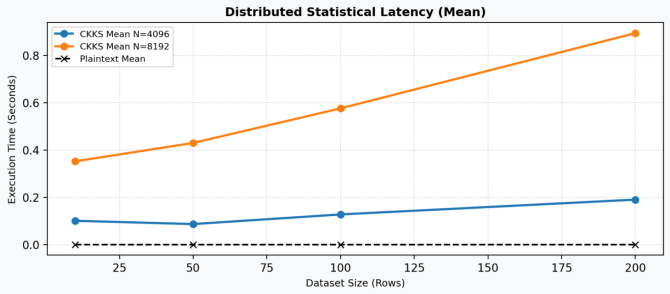


Figure 9: Distributed statistical latency profile for homomorphic mean accumulation across scaling row sizes.

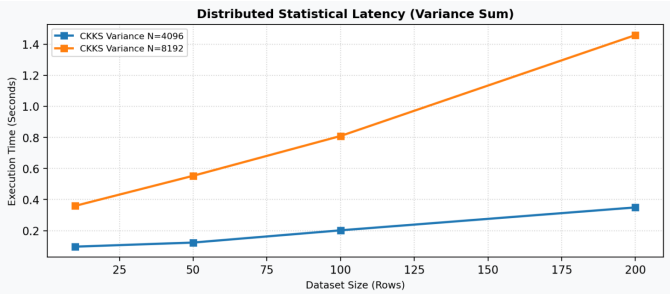


Figure 10: Distributed statistical latency profile for homomorphic variance sum operations displaying linear computation complexity scaling.

$N = 8192$ setup, the mean processing time shifts upward, scaling from $0.3524$ seconds to $0.8921$ seconds. This sub-second execution pattern demonstrates the efficiency of linear additions over encrypted structures, verifying that basic vector reductions can be executed on cloud resources without introducing severe performance bottlenecks [1].

A similar trend appears in the encrypted variance accumulation results shown in Fig. 10. The variance sum operation utilizes an additive combination of encrypted elements matched against pre-calculated plaintext metrics, avoiding complex ciphertext-by-ciphertext multiplication loops [2]. Consequently, processing times scale linearly ($O(N)$) with batch size, reaching a peak latency of $1.4583$ seconds at 200 rows under the $N = 8192$ profile. This execution profile confirms that calculating aggregated clinical statistics over encrypted matrices is practical for standard medical monitoring scenarios.

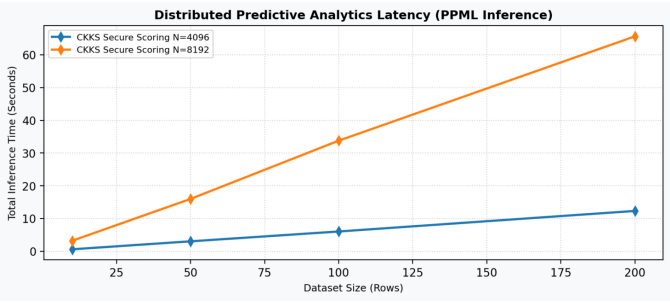


Figure 11: Distributed predictive analytics latency for secure logistic regression scoring highlighting network serialization bottlenecks.

### *E. Architectural Bottleneck Discussion*

The empirical profiles demonstrate that homomorphic addition arrays scale linearly ($O(N)$) with batch size, matching the expected theoretical behavior. However, a measurable execution penalty is observed when comparing polynomial modulus scales. As shown in Fig. 9 and Fig. 10, switching from $N = 4096$ to $N = 8192$ increases the processing latency of mean and variance steps by approximately 3 to 4 times, driven by the larger ring size and wider coefficient array blocks needed to support 128-bit security levels at higher depth bounds [1].

This processing latency increase stems from the fundamental properties of cyclotomic polynomial multiplications [1]. When doubling the degree parameter $N$, the number of coefficient operations required during Fast Fourier Transform (FFT) and Number Theoretic Transform (NTT) routines scales super-linearly, increasing CPU usage and instruction cache miss rates on the underlying hardware core. Furthermore, larger coefficient bit sizes require managing multi-precision integer data structures, adding computational overhead that impacts runtime performance.

A notable engineering constraint appears within the predictive inference scoring endpoint shown in Fig. 11. Under the $N = 8192$ profile, the total execution time increases to $65.412$ seconds for a batch of 200 patient entries. This behavior is primarily a Network-I/O bound bottleneck rather than an algorithmic limitation [4]. Because the patient entries are processed sequentially, each row requires separate base64 string compilation, HTTP payload transmission through the container bridge, and server-side reconstruction before multiplication can occur [2]. This data expansion factor increases the total communication overhead across the network link, demonstrating that distributed homomorphic architectures are highly sensitive to transmission and serialization penalties [1].

This networking penalty represents a significant architectural challenge for web-scale homomorphic deployments. When evaluating continuous scoring commands row-by-row, the application is subjected to repeated serialization overhead, network interface card context switches, and JSON string parsing delays for every single patient instance. Since each encrypted vector payload grows to several hundred kilobytes, processing a large batch sequentially can saturate the virtual network interfaces, leading to noticeable transmission delays [4]. This pattern indicates that optimization efforts should focus on refining serialization protocols and transport mechanics rather than modifying the underlying polynomial ring arithmetic.

## V. Related Work

The operational integration of homomorphic encryption algorithms within web services and secure database storage systems has been extensively analyzed across various cryptographic contexts. Standard benchmarks using the Microsoft

SEAL library or the TenSEAL abstraction layer demonstrate that executing arithmetic operations directly on encrypted vectors introduces a measurable processing penalty compared to native plaintext execution [2]. For instance, comparative studies profiling the performance boundaries of the BGV and BFV schemes indicate that tree-based or deep learning models require complex polynomial approximations for non-linear activations, such as the sigmoid function, which can degrade execution speed when scaled to large datasets.

To address these processing limitations, existing literature frequently examines localized hardware adjustments, leveraging graphical processing units (GPUs) or field-programmable gate arrays (FPGAs) to speed up polynomial additions and number-theoretic transform (NTT) stages. While these approaches help reduce computational bounds, they often treat the homomorphic engine as an isolated component, omitting the communication constraints introduced by distributed cloud infrastructures. Recent investigations into privacy-preserving machine learning (PPML) workflows emphasize that data expansion remains a major challenge. Because encrypted polynomials are significantly larger than their corresponding plaintext vectors, transmitting base64-encoded ciphertexts over standard web interfaces introduces considerable communication latency [1]. This study contributes to this body of research by evaluating these network and serialization overheads empirically within a containerized client-server framework, analyzing the performance trade-offs under practical operational constraints [4].

## Source Code

The complete implementation source code, container configurations, virtual networks setup, and high-resolution latency benchmarking scripts developed in this study are openly accessible in the public GitHub software repository at: https://github.com/rafidhiyaulh/ckks-medical-privacy-overhead.

## Video Link at YouTube

The video presentation detailing the design, containerized architecture deployment, and empirical cryptographic latency benchmarks of this distributed privacy-preserving framework can be accessed at the following YouTube URL placeholder path: https://youtu.be/NHzLOBjMgb0.

## Acknowledgment

The author wishes to express sincere gratitude to Dr. Ir. Rinaldi Munir, M.T., as the course coordinator for II4021 Cryptography, for providing profound theoretical insights, academic supervision, and baseline design structures throughout the semester. Special appreciation is also extended to the course teaching and practicum assistants, Ahmad Naufal Ramadhan and Diero Arga Purnama, for their continuous technical assistance, assignment coordination, and invaluable structural clarifications during the development and implementation phases of this containerized homomorphic processing architecture.

## References

[1] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic Encryption for Arithmetic of Approximate Numbers," in *International Conference on the Theory and Application of Cryptologic Techniques*. Springer, 2017, pp. 409–437.

[2] A. Benaissa, B. Retiat, B. Self, and R. Wagner, "TenSEAL: A Library for Encrypted Tensor Operations Using Homomorphic Encryption," *arXiv preprint arXiv:2104.03152*, 2021.

[3] W. N. Street, W. H. Wolberg, and O. L. Mangasarian, "Nuclear Feature Extraction for Breast Tumor Diagnosis," in *Biomedical Image Processing and Biomedical Visualization*, vol. 1905. SPIE, 1993, pp. 86–97.

[4] D. Merkel, "Docker: Lightweight Linux Containers for Consistent Development and Deployment," *Linux Journal*, vol. 2014, no. 239, p. 2, Mar. 2014.

[5] UCI Machine Learning Repository, "Breast Cancer Wisconsin (Diagnostic) Data Set," *Kaggle Dataset Archive*, 2016. https://www.kaggle.com/datasets/uciml/breast-cancer-wisconsin-data.

## Declaration

I hereby declare that this paper is my own writing, not an adaptation or a translation of someone else's paper, and not plagiarism.

Bandung, June 19, 2026

Muhammad Rafi Dhiyaulhaq – 18222069