Performance Benchmarking of FHE Libraries for Privacy-Preserving Machine Learning

Biswajit Mandal[1], Arup Mazumder[2], Sohan Paul[3]
[1]biswajit22@iiserb.ac.in
[2]mazumderj33@gmail.com
[3] 24f2009295@ds.study.iitm.ac.in

Supervisors: Dr. Amit Kumar Chauhan, Mr. Ayan Chattopadhyay

Date: August 12, 2025

Q→NU
Proactively Quantum™

Abstract

A detailed benchmarking study of Fully Homomorphic Encryption (FHE) libraries for machine learning applications, with a focus on TenSEAL library based on Microsoft's SEAL, is presented in this report. The study assesses computational overhead, memory consumption, ciphertext expansion factors, and noise propagation across various training paradigms. The results tell us significant trade-offs between security, accuracy, and computational efficiency, with BFV performing well in exact arithmetic scenarios and CKKS performing better for approximate computations. A Docker-based architecture is developed, which offers isolation, scalability, and reproducible environments to benchmark privacy-preserving machine learning. By setting clear baseline performance metrics and ensuring results can be reliably reproduced, our containerized benchmarking framework makes it easier to methodically evaluate how well privacy-preserving machine learning systems perform in realistic, fully homomorphic encryption (FHE) deployments.

Keywords: Fully Homomorphic Encryption, TenSEAL, CKKS, BFV, Machine Learning, Privacy-Preserving Computing, Docker, Benchmarking, Containerization.

Acknowledgments

We would like to deeply acknowledge QNu Labs for giving us the great opportunity of collaborating with them. Special thanks to our supervisors Dr. Amit Kumar Chauhan and Mr. Ayan Chattopadhyay for their valuable inputs on our weekly meetings. Their contributions have been truly insightful and motivated us to explore a different aspect of post quantum cryptography. Last but not the least, we are grateful to our institutes IISER Bhopal and IIT Madras for their constant supports and resources provided by them.

Contents

List of Figures

Chapter 1

Introduction

Data privacy is important due to the widespread use of machine learning applications. Conventional machine learning methods require that data be processed in plaintext, which leaves sensitive information vulnerable. The cryptographic primitive called FHE allows computation on encrypted data without the need for decryption, protecting privacy throughout the computations. But practical implementations remained computationally prohibitive for several years, recent advancements in lattice-based cryptography, particularly the BFV [Bra12, FV12] scheme by Brakerski, Fan, and Vercauteren in 2012, the GSW framework by Gentry, Sahai, and Waters in 2013, and the CKKS [CKKS17] scheme by Cheon, Kim, Kim, and Song in 2017, have made FHE increasingly viable for real-world applications. The theoretical enhancement of FHE were established by Gentry in his seminal 2009 work [Gen09], in which he constructed the first fully homomorphic encryption scheme using ideal lattices, demonstrating the theoretical feasibility of arbitrary computation on encrypted data. TenSEAL [BRCB21] build on Microsoft's SEAL [SEA23] is a Python library that FHE operations while preserving computational efficiency, and it implements both CKKS and BFV schemes, giving developers the option to choose between approximate and exact arithmetic depending on the needs of their applications.

1.1 Problem statement and research objectives

Although FHE holds great potential in theory, there is a considerable gap between cryptographic theory and real-world machine learning implementation; existing benchmarking efforts frequently lack systematic evaluation frameworks and do not take into account realistic deployment scenarios with appropriate isolation and resource management; and the computational overhead, memory requirements, and accuracy trade-offs associated with various FHE schemes remain poorly characterized for common machine learning algorithms. Our work analyzes the performance of CKKS [CKKS17] and BFV [Bra12, FV12] schemes for linear and non-linear classification tasks on different training mode. This study also looks at practical considerations for implementing FHE-based machine learning , paying special attention to the effects of noise growth and ciphertext expansion factors.

1.2 Contributions and Significance

We provide thorough performance characterization for these core FHE schemes by systematically comparing CKKS [CKKS17] and BFV [Bra12, FV12] schemes across multiple performance dimensions, such as execution time, memory usage, and accuracy preservation. This work makes several important contributions to the field of privacy-preserving machine learning. The creation of a containerized, reproducible benchmarking infrastructure allows for systematic evaluation of FHE performance across various deployment scenarios. A Docker-based client-server architecture is designed and implemented to simulate realistic deployment scenarios and provide the isolation and resource management required for accurate performance measurement. The custom implementations of Linear Means Classifier and Logistic Regression, optimized for homomorphic computation, show useful approaches to adapting classical machine learning algorithms for encrypted data processing. These contributions collectively advance the state of practice in privacy-preserving machine learning and offer useful insights for researchers and system architects. The performance analysis of metrics, such as resource utilization, noise growth, and ciphertext expansion factors, lays the groundwork for future research and practical deployment decisions.

Chapter 2

Theoretical Background

2.1 Mathematical Foundations

The foundation of FHE schemes is the *Ring Learning with Errors* (RLWE [LPR13]) problem, which is thought to be challenging even for adversaries with quantum capabilities.

$$R_q = \mathbb{Z}_q[x]/(x^N + 1)$$

The security of FHE schemes comes from the difficulty of separating RLWE samples $(a, a \cdot s + e) \in R_q \times R_q$ from uniformly random pairs, where s is a secret, a is uniformly random, and e is a small error term. R_q denotes the cyclotomic ring of degree N with modulus q, where N is a power of two.

2.2 Fully Homomorphic Encryption

It is possible to perform *arbitrary computation on encrypted data* using Fully Homomorphic Encryption (FHE) without having to first decrypt it.

Let, $\mathsf{Enc}(m)$ represents the encryption of a confidential message m and $\mathsf{Dec}(c)$ represents the decryption of ciphertext c. If one wants to evaluate the function f on encrypted message c:

$$\mathsf{Dec}(\mathsf{Eval}(f, \mathsf{Enc}(m))) = f(m)$$

where, f is efficiently computable function such as a circuit.

FHE Essential Elements

- *Key Generation:* $\mathsf{KeyGen}(1^\lambda)$: Produces public and/or secret keys.
- *Encryption:* $\mathsf{Enc}(pk, m)$: Uses the public key pk to encrypt a plaintext message m.
- *Evaluation:* $\mathsf{Eval}(evk, f, c)$: Uses the evaluation key evk to apply the function f to encrypted data c.
- *Decryption:* $\mathsf{Dec}(sk, c')$: Unpacks the resultant ciphertext c' with the secret key sk.

The CKKS (Cheon–Kim–Kim–Song) scheme [CKKS17] and BFV (Brakerski–Fan–Vercauteren) [Bra12, FV12] are lattice-based Fully Homomorphic Encryption (FHE) schemes for *approximate* arithmetic on encrypted data and *exact* arithmetic respectively . Unlike exact-arithmetic schemes like BFV or BGV, CKKS encodes plaintexts as complex (or real) vectors and supports efficient homomorphic addition and multiplication with controllable approximation error. This feature makes CKKS [CKKS17] ideal for privacy-preserving machine learning (PPML) and numerical data analytic, where exactness is not strictly necessary. Where BFV [Bra12, FV12] enables precise results modulo a plaintext modulus t which makes BFV ideal for applications where numerical accuracy is crucial, like secure multiparty computation, encrypted database queries, and exact integer arithmetic.

2.3 Theoretical Background of CKKS

2.3.1 Estimated Encoding

One of CKKS's unique features is its *approximate encoding* of complex vectors: given a plaintext vector $\mathbf{z} \in \mathbb{C}^{N/2}$, the encoder uses an invertible embedding based on the *canonical embedding* of cyclotomic fields to map $\mathbf{z}$ into a polynomial $m(x) \in R_q$. This mapping adds a scaling factor Δ to maintain precision:

$$m'(x) = \lfloor \Delta \cdot m(x) \rfloor \bmod q.$$

Re-scaling is necessary to control growth because the scaling factor changes multiplicatively with multiplications during computation.

2.3.2 Homomorphic Operations

The primary ciphertext operations that CKKS supports are as follows:

- *Addition:* Adding encrypted vectors component-by-component with minimal impact on noise growth.
- *Multiplication:* Multiplication of encrypted vectors by elements, increasing the scaling factor and noise.
- *Re-scaling:* To decrease scale and modulus, divide the ciphertext coefficients by a factor, usually Δ.
- *Rotation:* uses special rotation keys to shift encrypted vector slots cyclically.

2.3.3 Management of Noise and Precision

The *modulus switching* and *re-scaling* techniques help manage noise and prevent overflow beyond the decryption bound. The choice of parameters (N, q, Δ) directly affects precision, performance, and security. Every homomorphic operation increases the noise embedded in ciphertexts, which in CKKS corresponds to both encryption error and approximation error from scaling.

Security Considerations: Since CKKS uses approximate values, small approximation errors have no effect on the security analysis. The security of CKKS is derived from

the hardness assumption of the RLWE [LPR13] problem in R_q. Generally, parameter selection aims for a classical security level of at least 128 bits, based on standard lattice-estimation tools like LWE Estimator [APS15].

2.3.4 Applications

With its support for SIMD-style packed operations, CKKS [CKKS17] further enhances throughput for vectorized workloads and has been successfully used in statistical analysis, encrypted database queries, and privacy-preserving machine learning inference, where computations are tolerant of small floating-point errors.

In conclusion, CKKS [CKKS17] provides an adaptable and effective framework for approximate encrypted computation, sacrificing accuracy in favor of speed and precision control, which is frequently used in practical numerical applications.

2.4 Theoretical Background of BFV

2.4.1 Encoding of Plaintext

BFV [Bra12, FV12] uses a scaling factor $\Delta = \lfloor q/t \rfloor$ to encode messages m from the ring $R_t = \mathbb{Z}_t[x]/(x^N + 1)$ into R_q. In particular, the encoded message is

$$m'(x) = \Delta \cdot m(x) \bmod q.$$

Following decryption and homomorphic operations, this mapping maintains exact modular arithmetic modulo t.

2.4.2 Homomorphic Operations

The following are the main functions that BFV supports:

- In R_t, ciphertext addition is equivalent to the modular addition of plaintexts.
- *Multiplication:* Modular multiplication in R_t is equivalent to ciphertext multiplication, which is followed by a relinearization step to minimize the size of the ciphertext. To control noise while maintaining the underlying plaintext,
- *Modulus Switching:* lowers the ciphertext modulus q.

2.4.3 Noise Growth and Management

The depth of supported computations is limited by the initial choice of parameters (N, q, t), unless bootstrapping is used. In BFV, every homomorphic operation increases the noise embedded in the ciphertext, and correct decryption is only possible if the noise stays below a threshold relative to q. Noise management techniques include *modulus switching* and careful parameter selection.

Security Considerations Because BFV performs exact arithmetic modulo t, there is no additional approximation error beyond the encryption noise. The scheme's security is linked to RLWE hardness in R_q. The selection of parameters uses lattice-estimation tools [APS15] to target standard security levels (e.g., 128-bit security).

2.4.4 Applications

When accuracy is needed in privacy-preserving computation scenarios, such as secure electronic voting tallying, encrypted database operations, processing genomics data, and financial computations, BFV is frequently utilized.

In conclusion, the BFV [Bra12, FV12] scheme is a practical option for integer-based encrypted computation since it offers a reliable precision and a robust security foundation for exact modular arithmetic through an effective FHE framework.

2.5 Encrypted Data Machine Learning: Difficulties and Adjustments

Privacy-preserving machine learning poses some challenges: Conventional algorithms need to be redesigned to account for the basic limitations of homomorphic computation, which necessitates careful consideration of noise management, circuit depth, and operation complexity. Algorithmic approaches to addition and multiplication operations are limited by the limited operation set available in FHE systems; non-polynomial functions, like activation functions in neural networks, require polynomial approximations, which introduces approximation errors and computational overhead; noise accumulation through computation chains places strict limits on circuit depth, necessitating algorithm modifications to minimize multiplicative depth. While later studies by Bourse et al. and Boemer et al. investigated deep learning applications with different optimization techniques, early work by Goldwasser et al. showed that private machine learning through cryptographic protocols was feasible, and Mishra et al.'s DELTA system offered useful insights into implementing privacy-preserving inference systems at scale.

Chapter 3

System Architecture and Containerized Infrastructure Design

3.1 Distributed Architecture Framework

The benchmarking system employs a micro-services architecture implemented through Docker containers, providing the isolation, scalability, and reproducibility required for systematic FHE performance evaluation. The design philosophy separates computational concerns into three primary service components, each responsible for distinct aspects of the privacy-preserving machine learning pipeline. The client service handles data encryption and result decryption operations, maintaining exclusive access to secret key material while providing encrypted data to downstream services. The server service performs homomorphic computations and model training operations, operating exclusively on encrypted data without access to secret key material. The benchmark service orchestrates experimental execution and collects comprehensive performance metrics across all system components. This separation of concerns enables realistic simulation of distributed privacy-preserving machine learning scenarios while providing the measurement infrastructure necessary for comprehensive performance evaluation. The architecture supports various deployment configurations, from single-node development environments to distributed production systems with multiple computational nodes.

3.2 Docker Containerization Strategy

The containerization strategy addresses several critical requirements for systematic benchmarking of FHE systems. Container isolation ensures that resource measurements accurately reflect the computational overhead of FHE operations without interference from other system processes. The standardized execution environment eliminates variability between different host systems, enabling reproducible performance measurements across diverse hardware configurations. Resource management through Docker's control groups enables precise measurement of computational overhead, memory consumption, and I/O characteristics. The containerized approach facilitates horizontal scaling through orchestration platforms while maintaining measurement accuracy across distributed deployments.

3.3 Docker Infrastructure Implementation

The Docker infrastructure implementation centers around a carefully designed Dockerfile that optimizes for both performance and measurement accuracy:

```
FROM python:3.10
WORKDIR /app
ENV PYTHONUNBUFFERED=1
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY . .
RUN mkdir -p /shared
VOLUME ["/shared"]
CMD sh -c "python-u/app/${ROLE}.py"
```

Listing 3.1: Dockerfile for benchmarking

The selection of Python 3.10 as the base image provides optimal compatibility with TenSEAL's native extensions while maintaining recent security patches and performance improvements. The specific version ensures consistent behavior across different deployment environments and eliminates version-related performance variations.

The PYTHONUNBUFFERED=1 environment variable ensures real-time logging output, which proves crucial for monitoring long-running FHE computations and identifying performance bottlenecks during experimental execution. This configuration enables real-time observation of computational progress and facilitates debugging of complex homomorphic operations.

The shared volume strategy enables secure context and data exchange between containers while maintaining service isolation. The /shared directory provides a controlled communication channel for encrypted data transfer, public context sharing, and result collection without compromising the security model or measurement accuracy.

The dynamic role assignment through the $ROLE environment variable enables a single container image to fulfill multiple service roles, reducing storage overhead and simplifying deployment management. This approach ensures consistency across service implementations while enabling specialized configuration for different computational roles.

3.4 Service Orchestration with Docker Compose

Listing 3.2: Docker Compose configuration for FHE benchmarking

```
version "3.8"

services
fhe_server
build
context .
dockerfile Dockerfile
image tenseal_fhe_server
container_name tenseal_fhe_server
volumes
- shared_volume/shared
```

```
environment
- ROLE=server

fhe_client
build
context .
dockerfile Dockerfile
image tenseal_fhe_client
container_name tenseal_fhe_client
volumes
- shared_volume/shared
environment
- ROLE=client

fhe_benchmark
build
context .
dockerfile Dockerfile
image tenseal_fhe_benchmark
container_name tenseal_fhe_benchmark
volumes
- shared_volume/shared
- ./results_ckks/app/results_ckks
- ./results_bfv/app/results_bfv
environment
- ROLE=main

volumes
shared_volume
```

The service orchestration design implements implicit dependency management through shared volume synchronization, ensuring proper initialization order without explicit dependency declarations that could introduce timing artifacts in performance measurements. Each service container operates with isolated resource name-spaces while maintaining controlled communication channels through the shared volume system.

Named volumes provide persistent storage for experimental results while enabling clean separation between ephemeral computational state and long-term result storage. The shared volume architecture ensures that cryptographic contexts and encrypted data can flow between services while maintaining proper isolation boundaries.

Host-mounted volumes for result persistence ensure that benchmark results survive container life cycle events, enabling longitudinal analysis and comparison across multiple experimental runs. This design facilitates systematic performance evaluation while maintaining data integrity across different deployment scenarios.

3.5 Inter-Service Communication Protocol

The inter-service communication protocol operates through file-based message passing via the shared volume system, providing both security and performance advantages over network-based communication. The protocol design minimizes communication overhead while ensuring that cryptographic material is properly protected throughout

the computation pipeline.

Context serialization and de-serialization operations follow a structured protocol where the client service generates cryptographic contexts and serializes them for server consumption. The server service loads public contexts for homomorphic operations while maintaining proper isolation from secret key material. This design ensures that the security model remains intact while enabling practical distributed computation.

The communication protocol includes comprehensive error handling and validation mechanisms to ensure data integrity throughout the computation pipeline. Serialized ciphertext validation prevents corruption-induced failures while maintaining system reliability across different deployment configurations.

Chapter 4

Implementation Framework and Algorithm Adaptations

4.1 TenSEAL Integration Architecture

The python class TenSealManager defined in our code is encapsulating scheme-specific operations and offering algorithm implementations a consistent API. The integration with TenSEAL [BRCB21] necessitated the creation of a unified abstraction layer that optimizes for performance characteristics unique to each FHE scheme while providing consistent interfaces across various schemes. The global scale parameter 2^{21} is selected to achieve adequate precision for floating-point operations . The process of parameter optimization involved extensive experimentation with coefficient modulus configurations to achieve optimal balance between computational depth and noise budget. When it comes to CKKS [CKKS17] operations, the configuration provides sufficient precision for machine learning operations while maintaining practical computational performance.

4.2 Linear Means Classifier: Foundation and Application Theory

The Linear Means is one of our ML algorithm for classification to showcase FHE capabilities. It works on the principle of minimum distance classification, assigning test vectors to the class with the closest mean vector in Euclidean space. The implementation uses the conventional method of calculating class mean using arithmetic mean operations for plaintext training scenarios, and the mathematical formulation entails computing

$$\mu_c = \frac{1}{|S_c|} \sum_{x_i \in S_c} x_i \tag{4.1}$$

for every class c, where the set of training samples that belong to class c is represented by S_c. To overcome homomorphic computation constraints, the encrypted training implementation needed to undergo basic algorithmic changes. Instead of executing division operations in a homomorphic manner, which would result in a large computational overhead and noise accumulation, the implementation computes encrypted class sums and preserves plaintext class counts.

4.3 Logistic Regression: Non-linear Function Approximation

Logistic regression implementation on encrypted data presents significant challenges due to the non-polynomial nature of the sigmoid activation function. Traditional implementations rely on the exponential function, which cannot be computed homomorphically without expensive bootstrapping operations or polynomial approximations.

The sigmoid approximation [CGH+18] strategy employs a carefully optimized third-degree polynomial:

$$\sigma(x) \approx 0.5 + 0.197x - 0.004x^3$$

This approximation provides reasonable accuracy in the effective domain $[-5, 5]$ while maintaining minimal multiplicative depth. The coefficient selection process involved least-squares optimization over the sigmoid function's practical operating range.

The encrypted training implementation employs a hybrid approach that balances privacy preservation with computational feasibility. While forward propagation operations occur entirely on encrypted data, gradient computation requires selective decryption to maintain algorithmic convergence. This design choice represents a practical compromise between pure homomorphic computation and algorithmic effectiveness.

4.4 Prediction Framework and Distance Computation

The prediction framework implements a unified interface supporting multiple schemes and training modes while optimizing for computational efficiency. The design abstracts scheme-specific details while providing consistent prediction interfaces across different algorithmic configurations. For Linear Means Classifier predictions, the system computes Euclidean distances between encrypted test vectors and class means through homomorphic operations. The distance computation employs the standard formula

$$d^2 = (x - \mu)^T (x - \mu) \tag{4.2}$$

implemented through homomorphic subtraction, element-wise multiplication, and summation operations. The distance computation process carefully manages noise accumulation through strategic operation ordering and parameter selection. By minimizing the multiplicative depth of distance calculations, the system maintains acceptable noise levels while preserving computational accuracy.

Chapter 5

Experimental Methodology and Performance Metrics

5.1 Datasets Characteristics and Preprocessing

The Boston Housing datasets provides realistic characteristics for machine learning evaluation while maintaining manageable computational complexity for Fully Homomorphic Encryption (FHE) operations. The datasets contains 506 instances with 13 continuous and discrete attributes, representing various housing market factors in the Boston metropolitan area.

Preprocessing operations include standardization through z-score normalization to ensure optimal performance with FHE schemes. The continuous target variable is discretized through a median split to create binary classification tasks suitable for the implemented algorithms. This preprocessing approach ensures that the evaluation focuses on FHE-specific performance characteristics rather than datasets-specific complexities.

5.2 Experimental Configuration Parameters

The experimental configuration employs carefully selected parameters to balance computational efficiency with security requirements. For CKKS operations, the polynomial modulus degree of 8192 provides sufficient security while maintaining practical performance characteristics. The coefficient modulus configuration ensures adequate noise budget for the required computational depth while minimizing ciphertext sizes.

BFV configuration employs a 20-bit prime modulus (1032193) that provides sufficient range for integer operations while maintaining optimal performance characteristics. The polynomial modulus degree matches the CKKS configuration to enable fair performance comparisons between schemes.

All the experiments were performed on a system with the configuration: 13th Gen Intel® Core™ i9-13900 CPU, 32 cores, 2.00 GHz, 64 GB RAM, Windows 11 OS.

5.3 Comprehensive Performance Measurement Framework

The benchmarking framework captures performance characteristics across multiple dimensions to provide a comprehensive evaluation of FHE system performance. Temporal metrics include preparation time for key generation and data encryption, training time for model parameter computation, and inference time for prediction operations.

Accuracy metrics encompass traditional machine learning evaluation criteria including classification accuracy and mean squared error, supplemented with FHE-specific noise error quantification. Resource metrics capture CPU utilization patterns and memory consumption characteristics across different operational phases.

Cryptographic metrics focus on ciphertext expansion factors, noise error magnitudes, and security parameter validation. These metrics provide insights into the practical deployment characteristics of different FHE schemes and operational configurations.

5.4 Systematic Benchmarking Protocol

The benchmarking protocol follows a systematic approach that ensures reproducible measurements while capturing comprehensive performance characteristics. Each experimental run begins with fresh container initialization to eliminate potential state-dependent performance variations.

The measurement protocol employs high-precision timing mechanisms to capture temporal characteristics while utilizing system monitoring tools to measure resource consumption patterns. The systematic approach ensures that performance measurements accurately reflect FHE operational overhead without interference from measurement artifacts.

Chapter 6

Results and Performance Characterization

6.1 Computational Performance Analysis

The comprehensive performance analysis reveals significant differences between CKKS [CKKS17] and BFV [Bra12, FV12] schemes across all operational phases, with distinct patterns emerging for different training modes and machine learning algorithms.

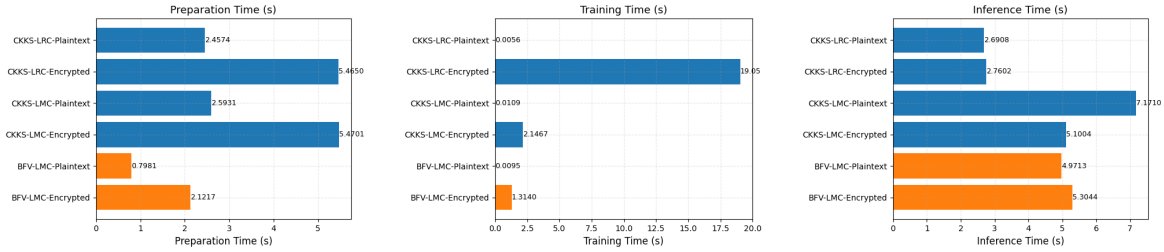
6.1.1 Performance Overview

The performance of the FHE schemes varies significantly across different computational stages. The core trade-offs involve computational overhead introduced by encryption, with some counter-intuitive results observed during the inference phase. Table 6.1 provides a consolidated summary of the execution times extracted from the provided charts.

Table 6.1: Comprehensive Performance and Quality Summary

		Time Performance (s)			Model Quality		Memory & Precision	
Scheme	**Data**	**Prep.**	**Train.**	**Infer.**	**Accuracy**	**MSE**	**Noise**	**Expansion**
BFV-LMC	Plaintext	0.80	0.01	4.97	0.82	0.18	1972.29	4158.21
	Encrypted	2.12	1.31	5.30	0.47	0.53	1972.29	4158.34
CKKS-LMC	Plaintext	2.59	0.01	7.17	0.79	0.21	1.36×10^{-3}	4212.50
	Encrypted	5.47	2.15	5.10	0.49	0.51	1.29×10^{-3}	4212.17
CKKS-LRC	Plaintext	2.46	0.01	2.69	0.71	0.29	1.82×10^{-3}	4213.21
	Encrypted	5.47	19.05	2.76	0.00*	N/A*	1.42×10^{-3}	4213.26

*The CKKS-LRC model failed to produce a meaningful result when trained on encrypted data, likely due to excessive noise accumulation from the extremely long training time. Accuracy is effectively zero, and MSE was not computable. The accuracy chart axis is assumed to represent scores (e.g., 0.82 = 82%), not percentages.



(a) Preparation Time (b) Training Time (c) Inference Time

Figure 6.1: Performance comparison of FHE schemes across the preparation, training, and inference stages.

6.1.2 Detailed Stage Analysis

Preparation Time

During the data preparation stage, computations on encrypted data are consistently slower than on plaintext. This is expected due to the computational overhead of the encryption process itself.

- The BFV-LMC scheme is the most efficient for preparation, both for plaintext (0.80s) and encrypted (2.12s) data.
- The CKKS schemes (both LRC and LMC) exhibit significantly higher preparation times, approximately 2.5-3 times slower than BFV on plaintext and over 2.5 times slower on encrypted data.

Training Time

The training phase highlights the most dramatic performance differences.

- Plaintext training is exceptionally fast across all configurations, with execution times being negligible (all under 0.011s).
- Encrypted training introduces a substantial performance penalty. The overhead is most extreme for the CKKS-LRC scheme, whose training time explodes from 0.0056s on plaintext to 19.05s on encrypted data. This makes it largely impractical for applications requiring frequent retraining.
- BFV-LMC demonstrates the best performance for training on encrypted data (1.31s), followed by CKKS-LMC (2.15s).

Inference Time

The inference stage reveals the most complex and noteworthy results.

- The CKKS-LRC model provides the fastest inference, both for plaintext (2.69s) and encrypted (2.76s) data, with minimal overhead from encryption.
- A significant and counter-intuitive result is observed for the CKKS-LMC scheme: inference on encrypted data (5.10s) is markedly faster than on plaintext data (7.17s).

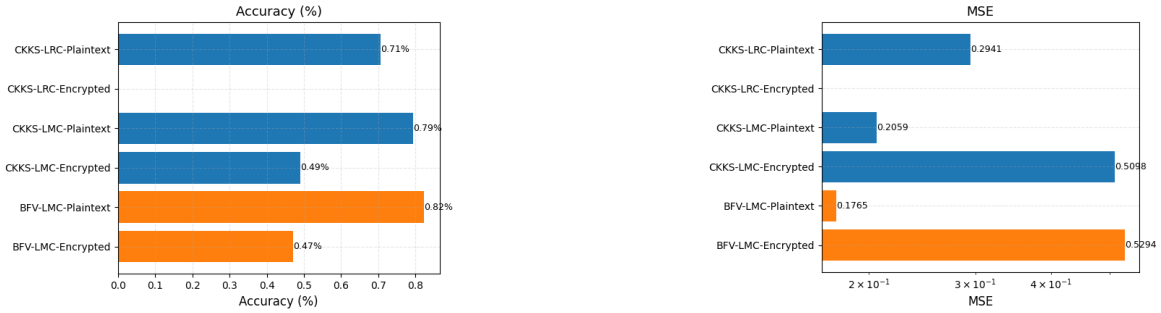
This suggests that the implementation may leverage parallel processing capabilities (e.g., Single Instruction, Multiple Data - SIMD) on packed ciphertexts more efficiently than iterating over individual plaintext values.

- The BFV-LMC scheme shows a small and expected increase in inference time for encrypted data compared to plaintext (5.30s vs. 4.97s).

6.1.3 Analysis of Accuracy and Mean Squared Error (MSE)

The ultimate goal is to produce an accurate model, but encryption poses a major challenge.

- *Plaintext Models*: All models trained on plaintext achieve high accuracy, with BFV-LMC leading at 82% and maintaining the lowest MSE.
- *Encrypted Models*: A severe degradation in quality is observed across the board. Accuracy scores are nearly halved, and MSE values more than double. This is a direct consequence of performing approximate arithmetic on encrypted data. CKKS-LMC performs best among the encrypted models with 49% accuracy.
- *Model Failure*: The CKKS-LRC model completely fails when trained on encrypted data, yielding zero accuracy. Its excessive training time likely leads to catastrophic noise accumulation, rendering the final model useless.



(a) Accuracy comparison (b) MSE comparison

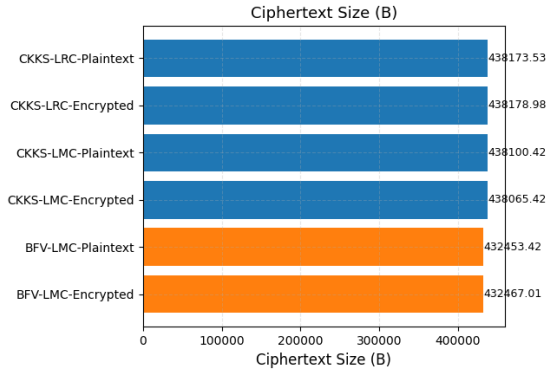
Figure 6.2: Performance metrics for CKKS and BFV schemes under plaintext and encrypted settings.

6.1.4 Analysis of Ciphertext Expansion and Noise

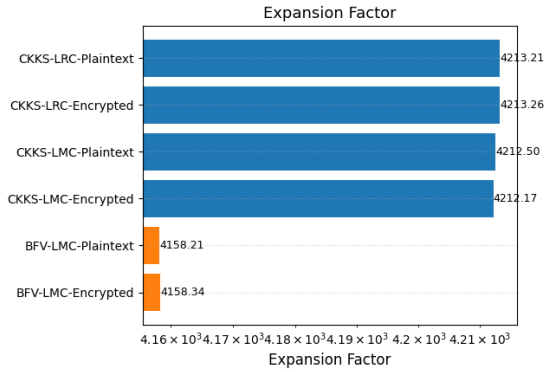
The analysis reveals significant trade-offs between the BFV and CKKS schemes, particularly concerning their data expansion and noise handling properties. While both schemes introduce a substantial data size overhead, they differ fundamentally in their approach to noise, which dictates their suitability for different computational tasks. Table 6.1 consolidates the data from the provided charts.

6.1.5 Data Size and Expansion Factor

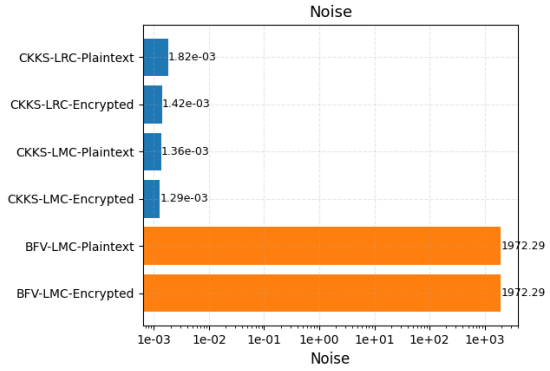
A fundamental characteristic of homomorphic encryption is the significant increase in data size when converting from plaintext to ciphertext.



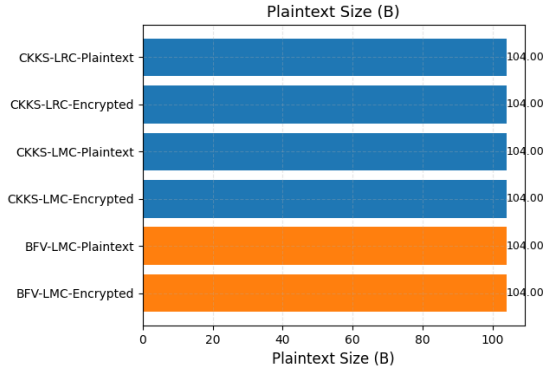
(a) Ciphertext Size (B)



(b) Expansion Factor



(c) Noise



(d) Plaintext Size (B)

Figure 6.3: Comparison of memory and noise metrics.

- *Plaintext Size*: The underlying plaintext data size was constant at 104 bytes across all experiments, providing a stable baseline for comparison.
- *Ciphertext Size*: As a consequence of encryption, the data size expands dramatically. Ciphertexts for all schemes are approximately 430 kB, representing a massive overhead.
- *Expansion Factor*: This metric, defined as $\frac{\text{Ciphertext Size}}{\text{Plaintext Size}}$, quantifies the overhead. Both schemes exhibit an expansion factor of over 4000x. The CKKS scheme (~4213x) has a slightly higher expansion factor than the BFV scheme (~4158x), resulting in marginally larger ciphertexts for the same input.

6.1.6 Noise Analysis

The noise chart highlights the most critical functional difference between the BFV and CKKS schemes. The x-axis of the noise chart is logarithmic, indicating vast differences in magnitude.

- CKKS Scheme: This scheme is designed for approximate arithmetic on real or complex numbers. It exhibits extremely low noise levels, on the order of $\mathbf{1.2 \times 10^{-3}}$ to $\mathbf{1.8 \times 10^{-3}}$. This inherent high precision makes it ideal for applications like machine learning, where small floating-point errors are acceptable and necessary.

- BFV Scheme: This scheme is designed for exact arithmetic on integers. The observed "noise" value of 1972.29 does not represent computational imprecision but rather the initial noise budget or noise ceiling. In BFV, noise is a resource that gets consumed with each homomorphic operation. A large initial budget allows for many operations to be performed before the noise level grows too high and corrupts the message. The result of a BFV computation remains exact as long as the noise stays within this budget.

Observations

The choice between BFV and CKKS depends directly on the application's requirements for numerical precision and the type of data being processed.

- BFV is the appropriate choice for applications requiring *exact integer arithmetic*, such as secure voting or private database queries. Its large noise budget supports deep computational circuits, and it offers a slightly more compact data representation than CKKS.
- CKKS is the superior choice for applications involving *approximate arithmetic on real numbers*, such as privacy-preserving machine learning. Its extremely low computational noise ensures high precision for results, which is essential for the convergence and accuracy of ML models. This precision comes at the cost of a marginally larger memory footprint.

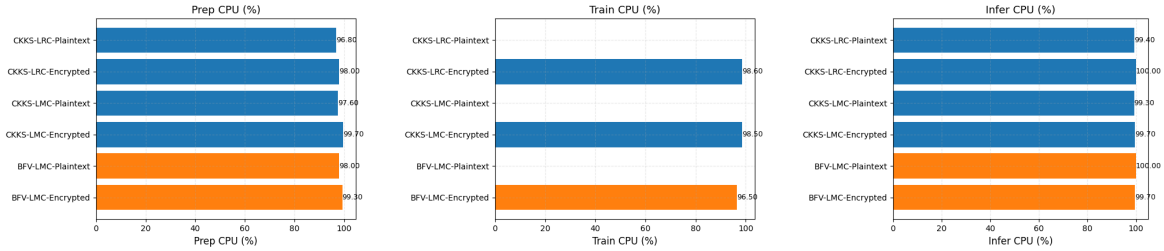
6.1.7 Resources Utilization Analysis

The performance evaluation of CPU and RAM utilization metrics reveals the computational trade-offs between CKKS and BFV schemes for both plaintext and encrypted training modes.

Analysis of CPU Utilization

Figures 6.4(a) 6.4(b) and 6.4(c) present the CPU utilization during the preparation and training phases and inference respectively.

- In the preparation phase, both schemes demonstrate high CPU utilization ($96 - -100\%$), with encrypted modes slightly exceeding plaintext modes due to additional encryption overhead.
- In the training phase, CPU utilization remains above 96% for all cases, with minimal differences between plaintext and encrypted executions, indicating that training is compute-bound regardless of encryption status.
- In the inference time, CPU utilization remains above 99% for all cases. The effect of encryption on CPU load is negligible during inference.



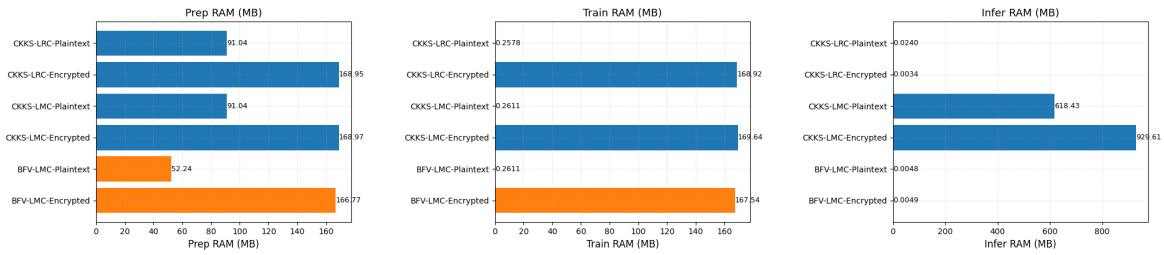
(a) Preparation CPU (%) (b) Training CPU (%) (c) Infer CPU (%)

Figure 6.4: Performance metrics for CKKS and BFV schemes in plaintext and encrypted modes.

Analysis of RAM Usage

This report compares RAM consumption for CKKS and BFV schemes in plaintext and encrypted modes across three phases: Preparation, Training, and Inference respectively.

- Preparation Phase:
  1. Encryption raises RAM use to around 166 MB–169 MB for all schemes.
  2. CKKS plaintext modes use almost double the RAM of BFV-LMC-Plaintext.
- Training Phase:
  1. LRC variants have negligible memory requirements (<0.03 MB).
  2. CKKS-LMC shows a huge gap between plaintext and encrypted modes.
- Inference Phase:
  1. Encrypted modes require ≈167 MB–170 MB, whereas plaintext modes consume only ≈0.26 MB.
  2. CKKS-LMC-Encrypted has the highest training RAM usage (169.64 MB).
  3. CKKS-LMC-Encrypted consumes the most memory during inference (929.61 MB).



(a) Prep RAM (MB) (b) Train RAM (MB) (c) Infer RAM(MB)

Figure 6.5: Performance metrics for CKKS and BFV schemes in plaintext and encrypted modes.

Overall Summary

- Encryption dramatically increases RAM consumption in all phases, except in CKKS-LMC during inference, where the jump is extremely large.
- The inference stage shows the largest variability, with CKKS-LMC-Encrypted consuming $\approx 1\,\text{GB}$.
- For low-RAM environments, BFV-LMC-Plaintext and CKKS-LRC variants are the most memory-efficient options.

6.2 Conclusion and Future Work

- Observations:
  1. Noise growth (training on encrypted data): for BFV is 1972.29 relatively higher than ckks. As a result: very poor accuracy: 47% and MSE: 52%.
     - It would be nice to investigate the reason behind poor performance.
  2. Training time : for "ckks-LRC-encrypted" is higher than "ckks-LMC-encrypted" as expected. But surprisingly, inference time for "ckks-LRC-encrypted" is much lower than "ckks-LMC-encrypted".
  3. RAM Usage: During inference for the case "ckks-LRC-encrypted" RAM : 0.0034 MB which is very much lower than the case "ckks-LMC-encrypted" (929.61 MB).
- Assume in the datasets, there are some crucial features with both data types *float* and *int*.
  - But, schemes ckks works for floating point arithmetic and BFV works for integer arithmetic.
  - How do we tackle such scenario?
  - It is important for the model accuracy perspective.
- For Logistic Regression, polynomial approximation of sigmoid function is necessary.
  - A degree 3 polynomial from [CGH+18], which approximate the sigmoid function in the range $[-5, 5]$.
  - $\sigma(x) = 0.5 + 0.197x - 0.004x^3$.
  - How can one train Logistic Regression by BFV?
  - Also in our study, we defined own training method, which is not working properly. It is subject to being thoroughly investigated.

Bibliography

[APS15] Martin R. Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, 9(3):169–203, October 2015.

[Bra12] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, volume 7417 of *Lecture Notes in Computer Science*, pages 868–886. Springer, 2012.

[BRCB21] Ayoub Benaissa, Bilal Retiat, Bogdan Cebere, and Alaa Eddine Belfedhal. Tenseal: A library for encrypted tensor operations using homomorphic encryption, 2021.

[CGH+18] Hao Chen, Ran Gilad-Bachrach, Kyoohyung Han, Zhicong Huang, Amir Jalali, Kim Laine, and Kristin E. Lauter. Logistic regression over encrypted data from fully homomorphic encryption. *IACR Cryptol. ePrint Arch.*, page 462, 2018.

[CKKS17] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. Homomorphic encryption for arithmetic of approximate numbers. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I*, volume 10624 of *Lecture Notes in Computer Science*, pages 409–437. Springer, 2017.

[FV12] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptol. ePrint Arch.*, page 144, 2012.

[Gen09] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. crypto.stanford.edu/craig.

[LPR13] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. *J. ACM*, 60(6):43:1–43:35, 2013.

[SEA23] Microsoft SEAL (release 4.1). https://github.com/Microsoft/SEAL, January 2023. Microsoft Research, Redmond, WA.