National College of Ireland

Effective Optimization strategies to utilize Fully Homomorphic Encryption in Cloud Platforms

MSc Research Project
MSc in Cloud Computing

Prajwal Seethur Raveendra
Student ID: 22228811

School of Computing
National College of Ireland

Supervisor: Aqeel Kazmi

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Prajwal Seethur Raveendra
Student ID:	22228811
Programme:	MSc in Cloud Computing **Year:** 2024
Module:	MSc Research Project
Supervisor:	Aqeel Kazmi
Submission Due Date:	12/08/2024
Project Title:	Effective Optimization strategies to utilize Fully Homomorphic Encryption in Cloud Platforms

Word Count:7897 **Page Count:** 20

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Prajwal Seethur Raveendra

Date: 12/08/2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Effective Optimization strategies to utilize Fully Homomorphic Encryption in Cloud Platforms

Prajwal Seethur Raveendra
22228811

Abstract

In the age of big data, meeting the data regulatory requirements is a fundamental need for companies who store and process private data in the cloud. The GDPR states that any personal information processed by any business has to be done with consent and with high security. Privacy-preserving models can be implemented in cloud environment to store and process personal data. These models guarantee privacy protection of each individual's private information but implanting these models can be computationally intensive. Homomorphic Encryption is privacy-preserving model that performs arithmetic operations on encrypted data without decryption. The first Fully Homomorphic Encryption scheme was invented in 2009 but it was not fit for real world applications because of its high computational requirements. Previous research focused on algorithmic advancements while this research aims to provide an efficient approach to utilize the existing Fully Homomorphic Encryption algorithms(schemes). A framework was developed using data splitting, parallel processing and AWS Lambda. The tenseal python library was used to homomorphically encrypt the values of a synthetic health care dataset. The evaluation was performed and results are reported. The results show a drastic 94% CPU time reduction when compared to other FHE schemes and a significant 82.88% reduction in memory usage after using data splitting and parallel processing.

1 Introduction

Cloud computing has been on a rise since the last few decades, taking over the IT industry. Many companies like META, ESPN and LinkedIn have made huge profits by utilizing cloud services from companies like AWS and Microsoft Azure. More than 50% of the companies in the world use the public cloud and 85% of all corporate organizations predicted to move a "cloud-first" strategy by the end of 2025 (Goasduff, 2021). Despite the fact that many businesses are shifting to the cloud, security and privacy remain a concern, which causes them to reconsider. There have been many cases where hackers were able to successfully breach into the company's data and steal private information of the customers. Take for example the notorious 2011 Sony PlayStation Network Outage during which millions of customer data such as email, address, passwords and credit card information were stolen overnight and cost the company a lot of money to recover from the damage dealt (Arthur, 2011). A cloud storage company Dropbox also had their data breached due to a bug that temporarily led any user to access anybody's account by using any random password and because of this a lot of sensitive data were leaked online (Pepitone, 2011). Such instances brought attention to the risk associated with cloud services. Indeed, a great degree of security

is currently provided by all cloud providers, and they guarantee the privacy of their data, but how can we be certain of that? Since we are unaware of the locations where our data is stored. The cloud data centres are only open to authorized personnel. To further strengthen security and guarantee greater privacy, AWS even stopped offering student visits inside the data centres. Even now, a lot of businesses are hesitant to move their critical data to the cloud. The majority of businesses appear to be hesitant to move to the public cloud due to concerns about security and privacy, so it is very important to make sure that company data, especially the sensitive ones, are handled carefully by providing some sense of privacy to the data.

In order to maintain confidentiality of private data, companies use privacy-preserving models to protect their sensitive information. Privacy-preserving models are mechanisms used to protect the sensitive data that are migrated to the external untrusted environments. It helps in maintaining the confidentiality, privacy and integrity of sensitive data and at the same time retain the originality of the data for it to be effective when used for analysis or calculations. When data is outsourced to cloud settings, we have to give up a lot of control over the data since the cloud service provider handles everything, including infrastructure maintenance and monitoring. This is really one of the main reasons why a lot of businesses are hesitant to shift to the cloud; they think that maintaining their confidential information on-site will increase transparency. Therefore, before outsourcing sensitive data to any cloud environment, it is crucial to adopt privacy-enhancing technologies to conceal or anonymize it in order to address the privacy dilemma. These models are also essential for maintaining compliance with GDPR regulations and maintaining confidentiality. Fairness, transparency, data minimization, and secure storage of personal data are all emphasized by the GDPR, and clear consent from data subjects is a crucial need. By following guidelines, cloud service companies may gain their consumers' confidence (Paul Voigt, 2017).

In cloud computing, a wide variety of privacy-preserving approaches are employed. Several popular ones are Secure Multi-Party Computation, Differential Privacy, Homomorphic Encryption, Data Anonymization/Masking, Federated Learning, and others. Every one of these approaches has benefits and drawbacks, so businesses must do in-depth study before selecting a model that best meets their needs while keeping in mind the compromises that come with a given choice. But when it comes to strictly about computation, Homomorphic Encryption is the most prominent option since it performs computations on encrypted data. The traditional way of performing computations on sensitive data is by using classic Encryption, which requires decryption before the actual computation. Homomorphic Encryption However takes it the next level by performing calculations on Encrypted data (cipher Text) without having to decrypt it. For instance, Healthcare and Financial data are very much sensitive and they are to be handled very carefully and ethically. In a healthcare setting, Useful insights can be taken extracted from patients' data by analysing the data but it should be ethical, as in no individual's sensitive information (like PII) be revealed, and make sure that it is done with consent. Homomorphic Encryption ensures the privacy and security of such sensitive data while providing useful insights.

Since calculations are done on cipher text, it up a lot of the computational resources which in turn leads to overheads. Researchers are still actively trying to come up with new schemes to make it practical. Since Craig Gentry's first scheme on Fully Homomorphic Encryption in 2009 till recent advancements in this field of study, Computational Intensity still seems to be a major drawback. Large arbitrary multiplication of numbers and a technique called Bootstrapping are the most computationally intensive tasks. There have been many schemes after Gentry with different Bootstrapping approach and algorithms all together to try and minimize the computational intensity but is still very hard to make it practical in the business world. Although, there are other schemes that is possible to have a practical implantation of Homomorphic Encryption with limited amount of multiplication depth like Somewhat Homomorphic Encryption (Brenner, 2012).

1.1 Research Questions

- How can the privacy preservation and computational efficiency of Fully Homomorphic Encryption be optimized for secure cloud environments in large datasets?
- How to effectively utilize Fully Homomorphic Encryption on sensitive data in untrusted cloud environment?

1.2 Objective of the research

The main goal of this research is to balance privacy and computational cost of FHE and demonstrate a more practical approach by using existing FHE tools and library. The approach in this research is very much different from others as the goal is to develop a framework using data splitting and parallel processing. The idea is to split the dataset based on sensitivity and perform FHE operations only on the ones which are very sensitive and process the calculations parallelly to further enhance the performance. In previous works, different algorithmic approaches have been taken to optimize the performance of FHE, like enhanced bootstrapping techniques or different noise management mechanisms. However, in this research, the existing algorithms (FHE schemes) are used in a more efficient way.

2 Related Work

The related work is grouped as follows, section 2.1 to 2.5 has all of the papers that proposed a new FHE Scheme and the section 2.6 has the papers that utilized hardware accelerators to enhance FHE performance.

2.1 Gentry's Initial Ideal Lattice based FHE

As part of his thesis, Stanford University graduate student Craig Gentry invented the first fully homomorphic encryption technique in 2009. Gentry was the first to suggest use an ideal-lattice based approach for fully homomorphic encryption. A mathematical structure

called an ideal lattice is employed in cryptography, specifically in encrypted calculations. Additionally, the concept of bootstrapping—which is essential to FHE—is introduced in this study. It is used to refresh the encrypted text and minimize the error vector's size or also known as “noise”, which grows larger with each operation. Simply said, error vectors are random vectors that are appended to the vector point during the encryption process. When the decryption process takes place, these error vectors are introduced to restore the original plain text. Gentry's Fully Homomorphic Encryption (FHE) approach uses a specific "squashing" mechanism to deal with the noise that builds up in cipher texts after several homomorphic operations. Because of bootstrapping and large multiplication operations, fully homomorphic encryption can be computationally expensive; this problem has been addressed by several studies since Gentry (Gentry, 2009). Since the goal of this research is to solve this problem, it is crucial to understand the findings of earlier studies and the novel strategies they suggested for enhancing FHE performance.

Numerous studies have been conducted in the wake of Gentry's work. Regev, for example, offers a classical public-key cryptosystem that is more effective than previous lattice-based systems. The size of the public key in earlier cryptosystems was around O(n4), while the size of the message grew after encryption by roughly O(n2). More realistic and effective, the proposed cryptosystem increases message size by around O(n) and decreases public key size to about O(n2). An extension of Regev (2009), Lyubashevsky et al. (2010) developed a novel Learn with Errors approach over integers. Since the encryption system is predicated on the difficulty of solving the approximate-gcd issue, any attack against it may be turned into a strategy for solving the problem. The technique uses a public key that is a string of numbers drawn from a certain distribution and a private key that is an odd number chosen at random. Although Lyubashevsky et al. (2010) scheme allows more FHE operations before the noise in the cipher texts becomes excessive, the cipher texts' noise level still limits this approach. Additionally, ring-LWE, an algebraic variant of LWE, is proposed in this study. But there are challenges in resolving the computational overhead problem (Regev, 2009).

2.2 BGV Scheme

The Brakerski-Gentry-Vaikuntanathan (BGV) Fully Homomorphic Encryption (FHE) scheme marks a significant step forward in the development of practical homomorphic encryption. Expanding upon Craig Gentry's groundbreaking work from 2009, which was the first to show that FHE was feasible, the BGV scheme sought to resolve the inefficiencies and high computational costs related to Gentry's first FHE scheme. Optimizing bootstrapping procedures and introducing more effective noise control strategies, the BGV system leverages the Ring Learning with Errors (Ring-LWE) problem as its security foundation. This made it possible to implement FHE in more useful ways that could still preserve the secrecy and integrity of the underlying data while carrying out arbitrary calculations on encrypted data. The BGV scheme's support for levelled homomorphic encryption, which allows for a set number of ciphertext operations without the need for regular bootstrapping, is one of its key advances. This new optimized approach of bootstrapping is brilliant because not all operations require bootstrapping, some simple arithmetic operations does not require

bootstrapping which consumed much of the computational resources and time. This improvement greatly lowers the computing cost, increasing the scheme's applicability in practical settings. Furthermore, the BGV technique maintains control over noise levels by using modular arithmetic for noise management, which keeps ciphertexts decryptable even after repeated operations. BGV offers strong security guarantees and robust performance, based on well-established hardness assumptions in lattice-based cryptography, and strikes a balance between efficiency, flexibility, and security in comparison to other FHE schemes such as the original Gentry scheme, the Brakerski-Vaikuntanathan (BV) scheme, and the Fan-Vercauteren (FV) scheme. The BGV scheme is used in many sectors that requires secure data processing in untrusted environment, especially in cloud computing. It is also used in encrypted machine learning to the train the models with sensitive data in securely and efficiently (Zvika Brakerski, 2014).

2.3 FV Scheme

The authors of the paper “somewhat practical fully homomorphic encryption” adapt Brakerski's FHE technique to the ring-LWE scenario, therefore expanding upon it. The FHE strategy is based on the Learning with Errors (LWE) issue. The goal of this shift is to optimize relinearization procedures, which will speed up computations and minimize the amount of relinearization keys. Relinearization is a mechanism that converts the complex cipher texts, that accumulates after homomorphic multiplication, into a much simpler and decryptable cipher texts. As mentioned previously homomorphic multiplication can be quite computationally intensive and relinearization helps in optimizing these operations in particular. Relinearization is similar to bootstrapping but it is only dedicated to simplifying homomorphic multiplication in cipher texts. In-depth analyses of many homomorphic processes, including bootstrapping, relinearization, and multiplication, are presented in the work, together with exact worst-case limitations on the noise these operations generate. In particular, a modulus switching approach is used to speed the analysis of the bootstrapping stage of the scheme. Additionally, the authors provide certain guidelines that guarantee a particular degree of security, making the system entirely homomorphic in these circumstances (Vercauteren, 2012).

The study presents an RLWE-based encryption technique that keeps the fundamental design of Brakerski's FHE while introducing practicality-enhancing modifications. The enhanced relinearization approach, which streamlines the procedure and lowers computer cost, is the scheme's main novelty. Furthermore, noise accumulation—a major problem in FHE schemes—is further minimized by bootstrapping, which makes use of a straightforward modulus switching technique. The robustness of the approach is shown by the authors' meticulous noise analysis and security parameter choices. It is anticipated that this work will provide a more probable solution for FHE applications.

2.4 TFHE Scheme

In (Chillotti, 2019), Fast Fully Homomorphic Encryption over Torus (TFHE), an additional approach was put forth. This also was predicated on the approximate gcd problem. Because TFHE's bootstrapping method is significantly quicker than that of other FHE schemes, it is a good fit for homomorphic encryption in cloud computing. TFHE is popularly known for its C++ implementation. The TFHE approach aims to maximize efficiency in terms of cipher text size, memory use, and computational cost. It does this by fusing symmetric key encryption with bit-wise operations. Additionally, it makes rotation operations easier, which enables efficient evaluation of circuits containing loops or variable-length operations.

2.5 CKKS Scheme

Cheon et al.'s study "Homomorphic Encryption for Arithmetic of Approximate Numbers" solves these issues by presenting a new HE technique tailored exclusively for approximation arithmetic. The core idea is to include noise as part of the inaccuracy inherent in approximation computations, allowing for speedy and secure arithmetic operations that prioritize large figures. This approach employs a rescaling process that truncates ciphertexts to control the size of the plaintext, successfully limiting accuracy loss. The decryption structure of this approach returns an approximate value of the plaintext with preset accuracy, which is a considerable advance over prior methods that frequently resulted in the destruction of most significant bits (MSBs) during homomorphic operations. The authors base their approach on the Ring Learning with Errors (RLWE) issue and use the BGV scheme's multiplication mechanism to increase the ciphertext modulus. They offer a batching approach that transfers plaintext polynomials to message vectors of complex numbers using a canonical embedding map, keeping the precision of plaintext after encoding while without bloating the magnitude of mistakes (Jung Hee Cheon A. K., 2017).

After their first paper on approximate arithmetic in FHE, a noteworthy addition is made by Jung Hee Cheon et al. with their complete RNS variation of approximation HE, which addresses performance issues in earlier implementations. Constraints on ciphertext modulus selection made it impossible for previous HE schemes, like the HEAAN, to integrate RNS decomposition with the Number Theoretic Transformation (NTT). By putting forth a unique form for the ciphertext modulus that permits the use of RNS decomposition and NTT conversion, Cheon et al. get around these problems. Because their approach only uses word-size operations instead of multi-precision arithmetic, it is more efficient and performs far better for fundamental operations such as homomorphic multiplication and decryption (Jung Hee Cheon K. H., 2019).

Performance optimization in polynomial arithmetic in HE has been greatly aided by the adoption of the double-CRT representation. Gentry et al. have emphasized the double-CRT technique, which breaks down polynomials into smaller components using the Chinese Remainder Theorem (CRT) to facilitate efficient calculations in smaller rings. This invention guarantees that homomorphic operations may be performed effectively without resorting to bigger integer representations, such as modulus flipping and rescaling. As a result, their method performs on par with, if not better than, other cutting-edge HE libraries, exhibiting

notable speedups in a variety of operations, including those crucial for machine learning applications.

The researchers demonstrate the usefulness of their complete RNS variation in practice by providing experimental findings that corroborate their contributions. Significant performance advantages are achieved by their implementation, most notably in lowering computation times for complicated tasks like constructing logistic regression models from encrypted data. This development is essential to bringing HE to be practical, especially in industries like banking and healthcare that need safe data processing. Thus, the complete RNS variation of approximation HE is an important discovery in cryptography research, as it allows new opportunities for privacy-preserving solutions by efficiently performing complex calculations on encrypted data.

2.6 Hardware Acceleration Approach

Cheon and team introduced the CKKS system, which is appropriate for applications needing numerical accuracy since it allows approximation arithmetic on encrypted data. Still, bootstrapping requires a lot of resources to run. In order to increase FHE performance, recent research has concentrated on utilizing hardware acceleration and improving bootstrapping.

These issues are resolved by the paper's suggested BTS (Bootstrappable, Technology-driven, Secure) accelerator, which optimizes calculations using homomorphic encryption. In contrast to other methods, BTS uses coefficient-level parallelism (CLP) to effectively manage large-scale calculations while balancing data transport and processing. In benchmarks like logistic regression and ResNet-20 inference, this architecture achieves up to 5,556× speedup, indicating considerable performance improvements (Kim, 2022).

By tackling the substantial computational overhead connected with the bootstrapping procedure, the paper "Over 100x Faster Bootstrapping in Fully Homomorphic Encryption through Memory-centric Optimization with GPUs" by Wonkyung Jung et al. expands on the body of work already done in the field of fully homomorphic encryption (FHE). The time-consuming nature of bootstrapping, which can take several minutes on standard hardware, has prevented practical applications from taking full use of FHE's infinite operations capabilities on encrypted data. While other attempts, including the PrivFT technique and the open-source library cuFHE, have made progress in using GPUs to speed up different homomorphic encryption algorithms, none have successfully targeted the RNS form of CKKS or executed bootstrapping at this degree of efficiency. The paper's listed related studies emphasize the uniqueness of the authors' methodology, drawing attention to the lack of GPU-accelerated bootstrapping solutions for CKKS and laying the groundwork for their contributions (Wonkyung Jung, 2021).

With an amortized bootstrapping time of 0.423 microseconds per bit, the authors achieve a 7.02× speedup for a single CKKS multiplication by a thorough study and implementation of memory-centric optimizations such as kernel fusion and reordering main functions. This

performance is 257× better than single-threaded CPU implementations and outperforms current GPU implementations. Furthermore, the authors show a 40× speedup over an 8-thread CPU implementation by incorporating these optimizations into a logistic regression model, demonstrating the usefulness of their work for privacy-preserving machine learning applications. The implementation of safe, effective data analysis tools in practical settings is made possible by this noteworthy development in lowering the computing load of FHE bootstrapping.

(Wang, 2012) employ a distinct methodology to address the computational intensity issue in Fully Homomorphic Encryption. The study suggests a more efficient method that will assist lower a large portion of Gentry's algorithm's computational overhead by making use of GPU-GPU acceleration and parallel processing technologies. The main findings of this experimental investigation are noteworthy since they demonstrate a notable improvement in performance when using the NVIDIA C2050 GPU. The paper's findings show a speedup of 8 for decryption, 8 for encryption, and 7.6 for recrypt when compared to the CPU implementation. Barrett's modular reduction method on a GPU and Strassen's FFT-based approach were used by the authors in an effort to accelerate the computationally expensive million-bit modular multiplication in FHE.

3 Research Methodology

Significant advancements have been made in the field of Fully Homomorphic Encryption both algorithmically and in terms of utilizing hardware accelerators for specific FHE operations to improve the overall performance. Researchers have come up with new schemes and ideas to reduce the computational intensity but there is little to none practical implementations of FHE in the business world. The main objective of this research is to come up with a better approach to make use of the existing Fully Homomorphic Encryption Schemes to make it more practical in real world application. The proposed solution is to adapt data splitting and parallel processing to efficiently perform Homomorphic calculations on large-scale sensitive dataset. This section details the specific approach taken to address the research questions.

3.1 Process overflow

There are many different FHE libraries and choosing which ones to implement is crucial because each schemes offer different level of security and efficiency. For example, the CKKS scheme has good noise control and faster than other schemes but it is based on approximate arithmetic, meaning the result is just an approximate value of the output but not the exact number. There are many different libraries that implement FHE, few popular ones are Microsoft SEAL, Helib, Pyseal, Concrete-python, Tenseal and pyfhel. Each of these libraries offer different schemes.

One of the goals of this research is to show that it is possible to use the existing tools/libraries available in Fully Homomorphic Encryption and develop a framework that effectively

utilizes FHE. The main idea is to split the dataset based on sensitivity of the data. In a dataset, not all columns have sensitive or personal information. Encrypting the whole dataset will take up a lot of resources and processing power. So, in order to effectively utilize FHE, we can split the dataset into two sections. One chunk of the dataset can have sensitive information and the other chunk can have non sensitive information. In this way we can perform Homomorphic operations only on the sensitive chunk and perform regular calculations on the non-sensitive data. This should potentially reduce much of the computational intensity of these operations.

To further improve the efficiency and performance of FHE operations on large scale dataset we can utilize parallel processing technologies. By distributing the workload across multiple processors, parallel processing will help take much of the load off of single CPU and will help avoid the CPUs to overheat. Certain FHE operations like long arbitrary multiplications can take a lot of time to execute. A certain noise is added to every Homomorphic operation to make the calculations secure. After each FHE operations, especially multiplication, the noise accumulates to a point where it becomes very hard to manage that noise. Bootstrapping was introduced to help with noise management and excessive bootstrapping itself can be intensive. So, integrating parallel processing in our code can help in faster bootstrapping and multiplication.

3.2 Setup of scenarios/case studies

Proper methodology needs to be followed to effectively assure the validity, reliability, and reproducibility of research findings, allowing for correct conclusions and valuable contributions to the field. Various case studies are to be carried out to get valuable information of the proposed approach by checking if the approach is valid when implemented. Various Fully Homomorphic Encryption Schemes will be compared in terms of CPU time, CPU usage and memory utilization using a simple Homomorphic operations like addition and multiplication by integrating the proposed approach. In the case studies, three FHE schemes have been compared, namely CKKS scheme, BGV scheme and BFV scheme, with the proposed methodology or framework. In order to conduct these case studies, we will have to setup a Linux environment because it works well with most of the available FHE libraries.

3.3 Framework Development

The end goal of this research is to develop a framework or a tool that effectively utilizes Fully Homomorphic Encryption. The reason for a framework development is because there aren’t any ready to use tools for companies to utilize Fully Homomorphic Encryption. The framework was built using various cloud technologies like AWS lambda handle homomorphic calculations and normal calculations and API gateway to invoke the lambda function from the client environment. The framework is built to handle huge datasets that contains thousands of sensitive data. The research specifically targets health care data

because it contains patients' personal information. Various analysis can be done with the data ethically by protecting the patient's privacy in a faster and more efficient manner.

4 Design Specification

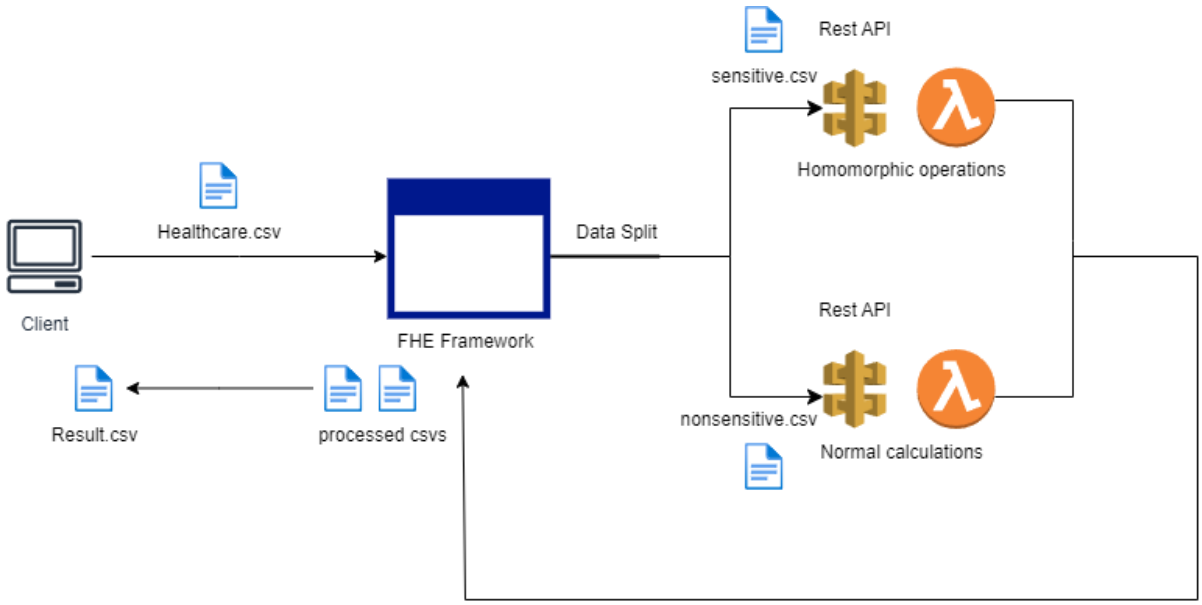


Figure 1: Architectural diagram of the developed framework

In this section, the entire process overflow is explained. The client script contains the URL of the REST API of the lambda functions and splits the dataset into two. One has sensitive data like patient id, age and PPI or email, the other chunk of dataset has non-sensitive patient health information like Blood pressure or body temperature. Two lambda functions are implemented to handle homomorphic (secure) calculations and the other handles normal calculations. Lambda is Function-as-a-service mechanism that allows you to focus on code and not worry about the underlying infrastructure. Lambda functions are usually used to perform a specific task of an application. These Lambda functions are invoked through API Gateway (REST) that is integrated to each lambda function. API Gateway is a fully managed API service managed by AWS. In this diagram, API Gateway takes the request from the client script, sends the request to the appropriate lambda function. The sensitive csv is sent to the sensitive lambda handler and it the data is sent in a JSON format. The JSON formatted data is converted back to its original state and encrypted calculations are performed on the dataset and the calculations leverage parallel processing. Parallel processing allows to process multiple tasks concurrently. This will help speed up the homomorphic operations performed on the lambda function. The results are converted back to JSON format and sent back to the client script. The non-sensitive chunk of the dataset is sent to the non-sensitive data handler and the calculations are done without encryption. After processing the data, it is sent back to the client through REST API. After receiving both of the processed outputs from their respective lambda functions, both of these data chunks are converted back to their original format and are rejoined, and the final processed output is displayed.

5 Implementation

This section shows how the proposed approach was implemented and also mentions what tools and languages were used to develop the framework. The purpose of this research is to demonstrate how the existing Fully Homomorphic Encryption libraries can be efficaciously used in huge datasets. Hence, the framework was built on top of the CKKS scheme using the tenseal python library.

5.1 Technologies and tools used

Serverless technology allows you to focus on coding without having to worry about the underlying infrastructure, including OS and server maintenance. AWS lambda has the ability to scale up and down the compute resources based on demand. Two lambda functions were programmed and deployed on to the AWS cloud. One lambda function handles homomorphic operations and the other handles normal arithmetic operations. This is a better and more optimized approach in secure data processing of sensitive data. By distributing the load to two different lambda functions, the computational intensity is greatly reduced. The lambda function that handles the sensitive (homomorphic operations) dataset has the python version 3.10 because the tenseal python library faced some dependency issues while trying to install that on a lambda server with the latest python version 3.12. The data frame is turned into a dictionary before sending it to the lambda function because the lambda functions are invoked using a payload that is in a dictionary format.

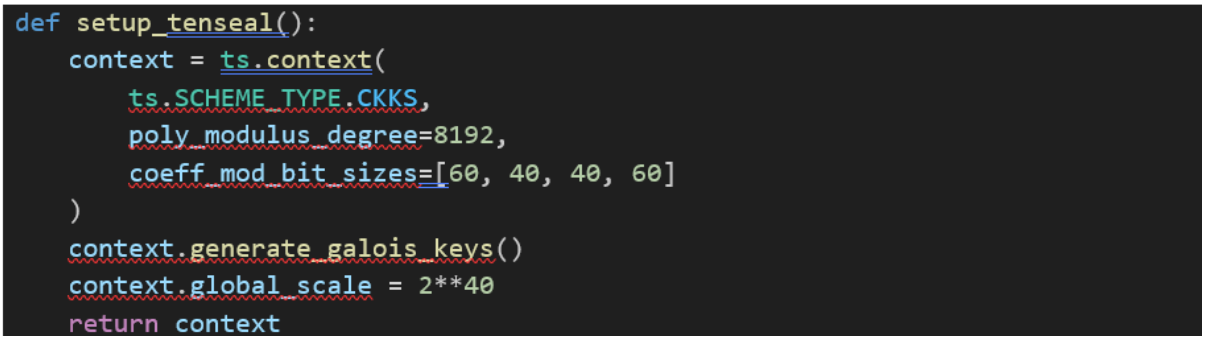
A “REST” API is integrated into the lambda function so that it can be invoked through a HTTP request. The datasets are sent to their respective lambda functions in the form of a JSON payload through API Gateway. The lambda functions take ‘data’ and ‘operation’ parameters payload from the client. ‘data’ contains the sensitive or non-sensitive chunk of the dataset and ‘operation’ specifies what kind of operation is done on the dataset. The Cross-Origin Resource Sharing (CORS) mechanism should be enabled in order for the client to be able to communicate with the lambda function. If not, the client application will not be able to access or invoke the lambda function through HTTP request.

The lambda functions were programmed using Python. Python is a high-level programming language and easy to read. The Homomorphic Encryption is done using python’s tenseal library. The tenseal library supports both the CKKS and the BFV schemes. I decided to use the CKKS scheme for homomorphic encryption because it is designed for approximate arithmetic and effectively utilizes hardware accelerators. To manipulate the data formats, the pandas library is used. It is used to read the dataset from the csv file and convert that into a data frame, so that it can be encrypted and meaningful calculations can be performed on it. The python built-in library ‘multiprocessing’ is used to perform the operations parallelly by breaking down the task into chunks and distributing them across multiple processors. A client

script is written to split the dataset and access the lambda function using the REST API Gateway.

5.2 Parameter Selection

The Tenseal context is set for initializing the encryption parameter before encrypting the data frame by choosing the type of Fully Homomorphic encryption Scheme used, In this case CKKS scheme. The polynomial modulus degree is set to 8192. This parameter is carefully set to balance performance and security because large polynomial modulus degree will be expensive computationally even though it increases performance. The degree of polynomial modulus specifies how hard the hidden mathematical problem will be. There has been many research conducted, trying to find the best parameter for polynomial degree modulus and 8192 is said to be appropriate for practical implementations of FHE schemes (Fan et al., 2024). The coefficient modular bit sizes determine the level of precession the resulting numbers will offer. The generate galois keys parameter is used when rotation operation is performed. The global scale parameter is simply a factor used to encode and decode values. The main purpose of this parameter is to maintain a necessary level of precession while performing homomorphic operations.



```
def setup_tenseal():
    context = ts.context(
        ts.SCHEME_TYPE.CKKS,
        poly_modulus_degree=8192,
        coeff_mod_bit_sizes=[60, 40, 40, 60]
    )
    context.generate_galois_keys()
    context.global_scale = 2**40
    return context
```

Figure 2: Tenseal Context Setup

The above code is a code snippet of teasel context set up. In the BGV and BFV scheme, the plaintext modulus is also specified and these parameters are also to be chosen appropriately to balance security and computational cost (Albrecht, 2021).

5.3 Homomorphic Operations

The framework itself supports three homomorphic arithmetic operations namely addition, multiplication and average. The addition operation performs sum of all the values in a column, the multiplication multiplies the values and the average finds the average of the values. Each of these values are encrypted and to encrypt each and every value, it is passed inside a for loop. The code snippet blow shows how average is calculated securely in the Lambda function.

```
if df[column].dtype in ['int64', 'float64']:
            if operation == 'average':
                encrypted_sum = encrypt_value(0, context)
                count = len(df)
                for i in range(count):
                    encrypted_value = encrypt_value(df[column].iloc[i],context)
                    encrypted_sum += encrypted_value
                encrypted_avg = encrypted_sum * (1.0 / count)
                results[column] = decrypt_value(encrypted_avg)
```

Figure 3: Encrypted Operation Example

More meaningful operations can be implemented. This is just to showcase that this approach viable.

5.4 Processed output

The client script combines the two datasets received from the lambda functions and displays the final output. The diagram below shows how the data is manipulated and processed back to the original structure of the dataset.

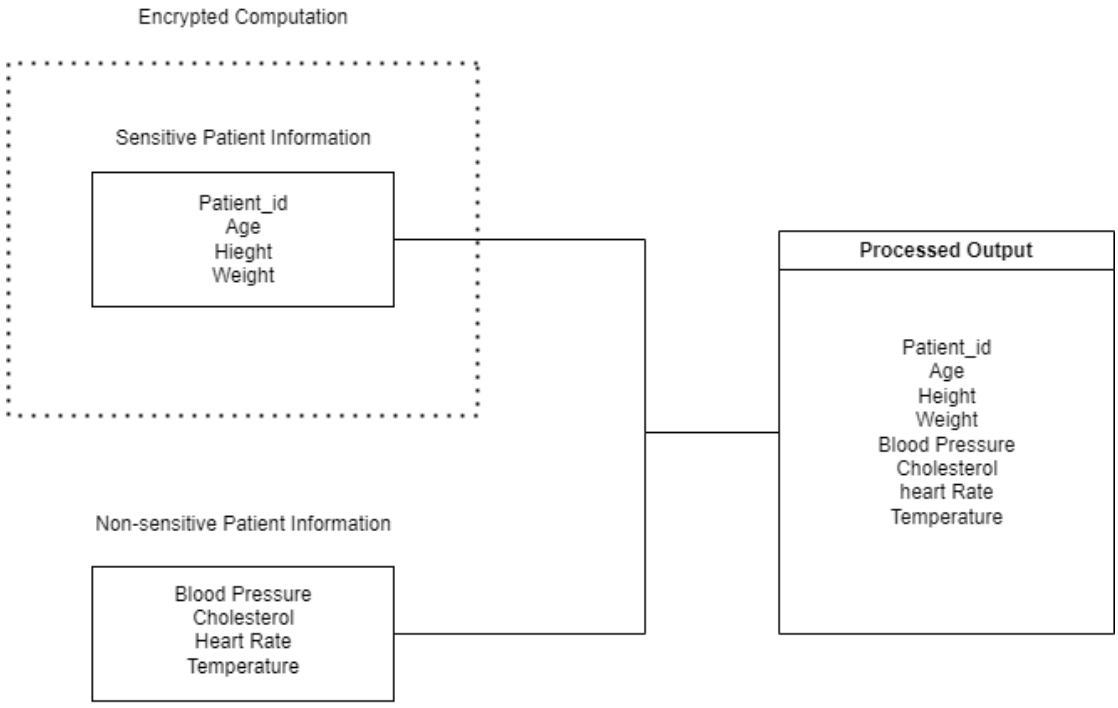


Figure 4: Dataset diagram after data splitting

5.5 Challenges and Resolutions

Initially when the lambda function that handles the homomorphic computations was deployed, it encountered timeout errors due to the time complexity of the encrypted operations. The default timeout for a lambda function is 5 seconds and it had to be increased

to 15 mins. During this process the assigned memory usage was also increased to 150 MB. In the real world the dataset sizes will be quite huge so the server that handles the homomorphic operation should have advance hardware components.

Ran into several formatting errors because the payload was not sent in the right format. The dataset is first converted into a dictionary format and the data sent from the lambda is also in the dictionary JSON format.

Dependency errors had occurred during the deployment of the lambda function. After doing some research it was found that the python version on the local machine where the code is written should be the same as the python version on the lambda function. The python version was updated manually in the WSL machine.

6 Evaluation

This section contains the comparisons of different Fully Homomorphic schemes with the developed framework and see how it performs in terms of execution time, CPU usage, memory used, voluntary context switches and involuntary context switches. Synthetic data has been used through out the case studies and every column of the dataset carries out addition operation to check how they perform in when compared with each other. For the purpose of this case study, a synthetic dataset containing 6000 rows has been used for evaluation. The dataset has 8 columns in which 4 of them contain sensitive information, like patient id and age, and the remaining are not sensitive. As the dataset is split into two in the framework, only the sensitive chunk is homomorphically encrypted. But in the other 3 FHE programs, the entire dataset is homomorphically encrypted. The case studies were conducted on a Windows Subsystem for Linux system with 13th gen intel core i7 processor, 32GB DDR5 memory, 1TB SSD and NVIDIA GeForce RTX 4070 Laptop GPU. In order to get accurate results, Garbage collection has been integrated in each of the python codes to avoid memory leaks and automate memory management. A separate script was written without using the lambda functions for the framework. The data is split and processed on the local machine to provide fair and consistent results.

6.1 Time Elapsed

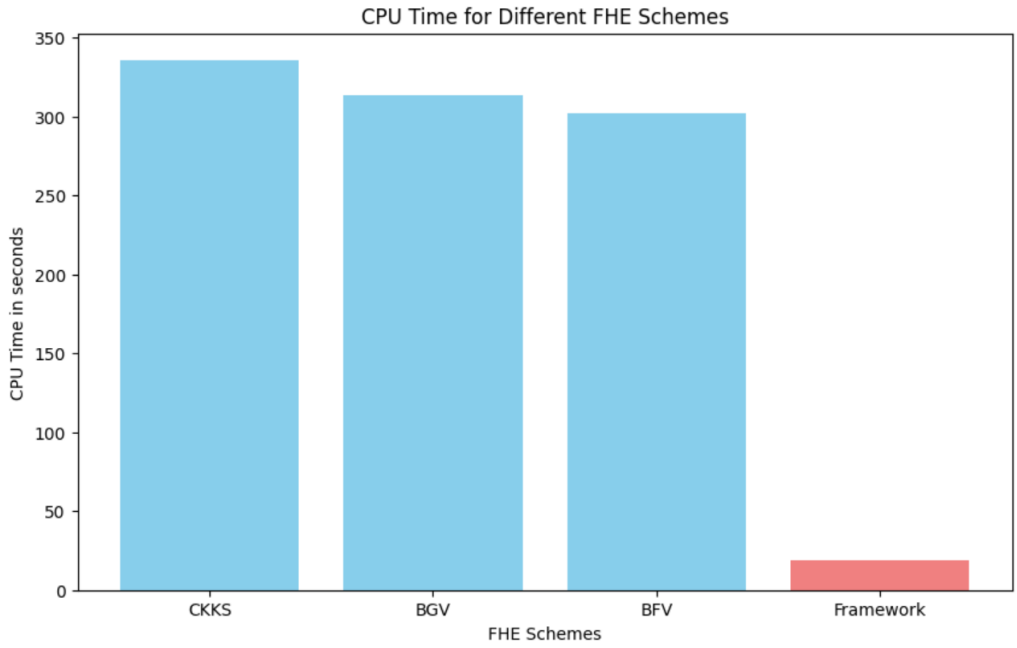


Figure 5:CPU Time for Different FHE Schemes

CPU time is one of the most important metrices to check how much time the processor takes to finish a task. Faster CPU time indicates faster and smoother processing capabilities of a program. The above bar chart illustrates the time taken by each the Fully Homomorphic Encryption schemes to complete its execution when compared to the developed framework. The total execution time is calculated by taking the difference of the start time and the end time.

6.2 CPU Usage

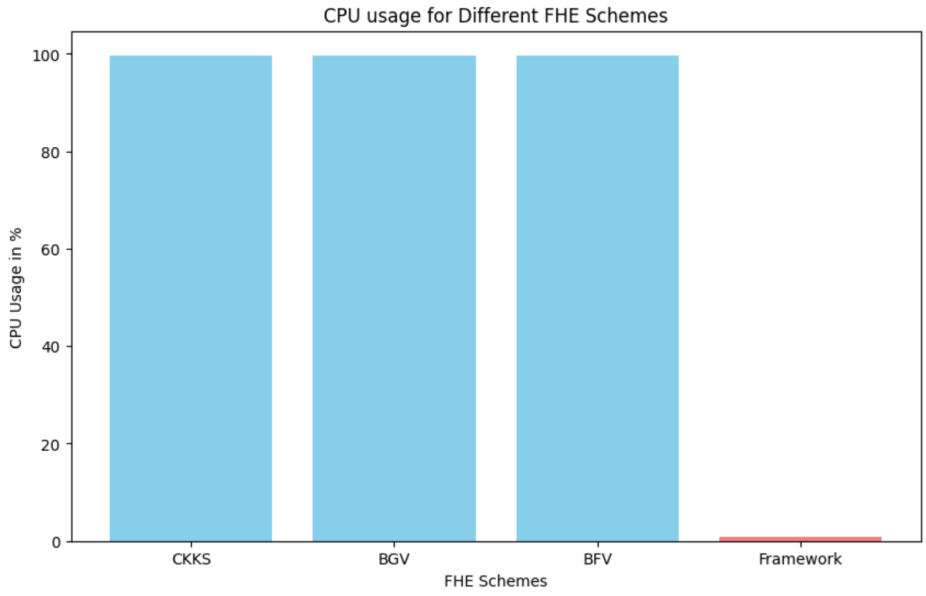


Figure 6:CPU Usage for Different FHE Schemes

Fully Homomorphic Encryption schemes are known to be computationally intensive because they utilize most amount of CPU resources available. The figure above shows the CPU usage

of each FHE schemes during the execution in comparison to the developed framework. Lower CPU consumption is preferable for FHE to be more practical in the real world especially for companies with insufficient resources. The CPU Usage of the python scripts were calculated using the psutil library.

6.3 Memory Usage

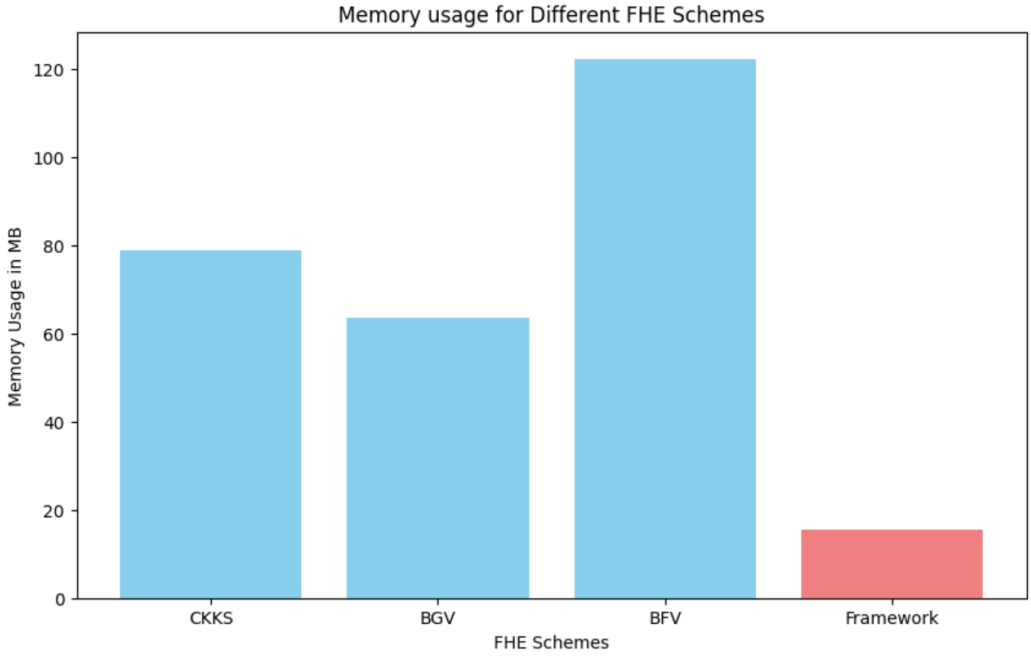


Figure 7:Memory Usage for Different FHE Schemes

Memory usage refers to the amount of Random-Access memory potentially used during the execution of code. Less memory usage means the program utilized only a small portion of the memory during the completion of its task. Effective memory utilization is crucial when it comes to Fully Homomorphic Encryption because of the noise accumulation after consecutive Homomorphic operations and large cipher text sizes, it can take up much of the memory space and only some left for the remaining calculations. It is desirable to consume less memory during execution. The above histogram shows the memory usage of each FHE schemes in comparison to the developed framework. Memory usage is calculated using the python library called memory profiler.

Bootstrapping is a mechanism that refreshes the noise in the cipher text when the noise gets too large. This also one of the reasons why FHE is so computationally intensive. Since only the sensitive columns are encrypted, bootstrapping is not regularly triggered when compared to the traditional schemes. Moreover, as the data is spread across multiple cores, the noise accumulation is also less compared to the traditional schemes.

6.4 Voluntary Context Switches

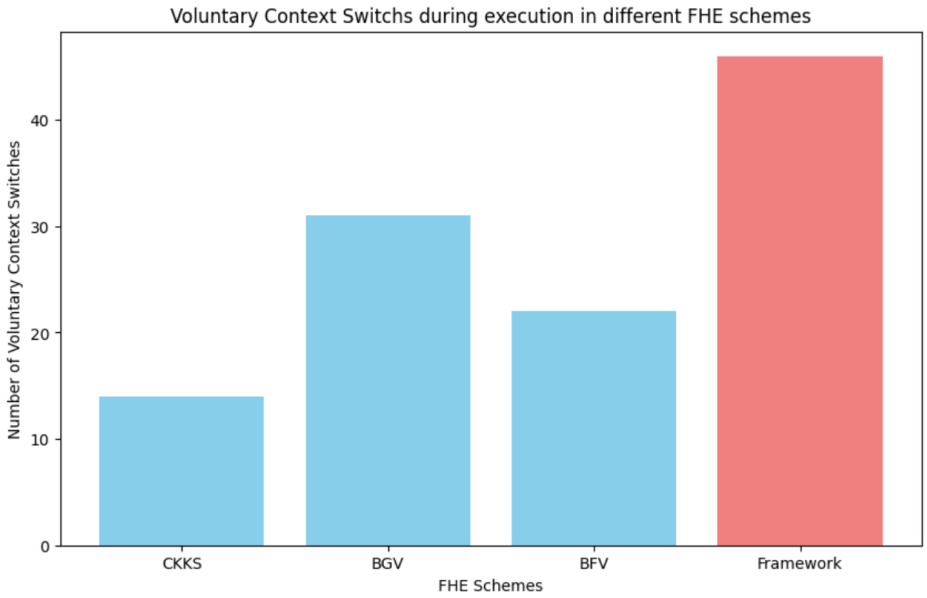


Figure 8:Voluntary Context Switches during Execution in Different FHE Schemes

Voluntary Context Switches arise when a process voluntarily surrenders the CPU while waiting for external operation (I/O operation). If a process is able to voluntarily give up control over the CPU, then it is safe to say that it has completed its assigned task within the allotted time stamp. Voluntary Context switches does not always mean it's a bad thing but frequent voluntary context switches can sometimes lead to overheads. The above chart demonstrates the number Voluntary Context Switches occurred during homomorphic operation on a synthetic dataset.

6.5 Involuntary Context Switches

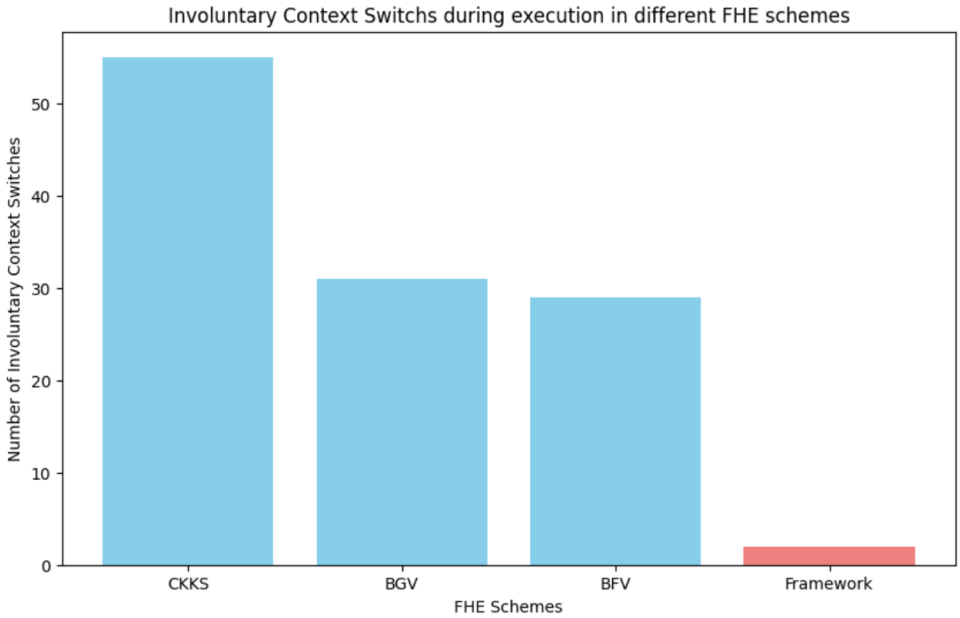


Figure 9:Involuntary Context Switches during Execution in Different FHE Schemes

Involuntary Context Switches happen when a process is running for too long and the Operating System suspends the process from the CPU. Excessive Involuntary Context Switches can heavily impact the performance. If the scheduler has to forcefully take control over the CPU means that the process was not able to finish the given task on time. The above bar graph displays the number of Involuntary Context Switches that occur during the execution of different FHE schemes when compared to the developed framework.

6.6 Discussion

From the above graphs we can get meaningful insights on how the developed framework performs when compared to other Fully Homomorphic encryption schemes in terms of CPU time, CPU usage, Memory usage and context switches.

CPU Time: The developed framework showcases the fastest execution time when compared to the other 3 FHE schemes. Since only the sensitive columns are Homomorphically encrypted, it takes much of the load off the CPU resulting in faster execution time. Normal arithmetic operations do not take much time to execute. It took 18.68 seconds to compute addition operation on the dataset while the other 3 FHE schemes took around 300 seconds to complete execution on the entire dataset. The Homomorphic calculations are processed parallelly which makes the calculations much faster.

CPU Usage: The developed FHE framework seem to have brought down the computational intensity. The proposed approach effectively balances the trade-off between computational complexity and protecting user privacy. The frame work utilized only 0.80% of the CPU since the tasks are offloaded to multiple processors instead of one. The other FHE schemes on the other hand recorded 99% of the CPU resources. The framework is efficiently utilizing the CPU and hence it drastically reduces overhead occurrences. The noise inside the cipher texts almost doubles than the one before after each operation and since only a part of the dataset is homomorphically processed, CPU is not overused.

Memory Usage: The framework uses less memory when compared to the FHE schemes because the data is almost reduced to half and there is not much bootstrapping necessary to refresh the noise since the noise accumulation is much less when compared to the schemes without data splitting. The BGV scheme used the most memory because it generates larger key size when compared to other FHE schemes and the bootstrapping technique is also not as efficient as others. The framework used 15.36 Mega Bytes of RAM during execution, which is significantly less compared to other schemes.

Voluntary Context Switches: By looking at the figure it is clear that the developed framework has the most amount of voluntary context switches with 46. Although Voluntary Context Switches does not always mean it’s a bad thing but it can have an impact on the performance when it happens too frequently. This can be because of large I/O operations or long waiting periods for external resources. This can be a problem when dealing with large datasets with 100,000 rows. This matter can be looked into by further investigating and the code can be optimized to reduce the voluntary context switches.

Involuntary Context Switches: The number of Involuntary Context Switches that happened during the execution was around 1 or 3 times in the framework. The CKKS scheme had the most amount of Involuntary Context Switches and due to its frequent disruption from the OS

it affects its overall performance and hence it took the most amount of time to complete execution when compared to others.

Schemes	CKKS	BGV	BFV	Developed Framework
CPU Time	335.93 seconds	313.37 seconds	301.88 seconds	18.68 seconds
CPU Usage	99.70%	99.68%	99.67%	0.80%
Memory usage	78.77 MB	63.50 MB	122.118 MB	15.36 MB
Voluntary Context Switches	14	31	22	46
Involuntary Context Switches	55	31	29	1

7 Conclusion and Future Work

A novel framework was successfully implemented using serverless architecture in AWS lambda. The main contribution of this research lies in splitting the dataset based on sensitivity and perform secure calculations by utilizing multiprocessing only on the sensitive chunk of the dataset. The framework was successfully able to send and retrieve data from the Lambda function through API Gateway. The framework was compared to the CKKS, BGV and BFV scheme where the whole dataset was Homomorphically encrypted regardless of sensitivity. After thorough evaluation the results show that the proposed approach out performs in almost every metrices including CPU Time, CPU Usage, Memory Usage and Involuntary Context Switches. But the number of Voluntary Context switches was the highest when compared to other Fully homomorphic Encryption Schemes. Since only a part of the dataset is homomorphically encrypted and the process is distributed, the CPU has more resources available which reduces the chances of overheads drastically.

The framework itself is not perfect and it can be further optimized by reducing the network buffers. AWS Lambda is a lightweight serverless architecture with limited amount of computing resources. The incoming and outgoing payload requests can only take around 20 MB. This maybe be more than enough for general purpose applications with basic functionalities but in FHE things like context setup and cipher text sizes can lead to large payload sizes. Moreover, if you're dealing with large datasets the payload invocation size can exceed past the maximum limit. Different technology with less network buffers and more processing capabilities can be setup to deploy the framework on to the cloud. More meaningful operations can be implemented like variance and median to get meaningful insights. A context aware bootstrapping mechanism can be implemented using machine learning to trigger bootstrapping only when absolutely necessary to improve the overall performance.

8 References

Arthur, B. Q. (2011, April 26). *PlayStation Network hackers accessed data*. The Guardian. https://www.theguardian.com/technology/2011/apr/26/playstation-network-hackers-data

Brenner, M., Perl, H., & Smith, M. (2012, July). Practical Applications of Homomorphic Encryption. In *SECRYPT* (pp. 5-14).

Fan, S., Deng, X., Tian, Z., Hu, Z., Chang, L., Hou, R., ... & Zhang, M. (2024). Taiyi: A high-performance CKKS accelerator for Practical Fully Homomorphic Encryption. *arXiv preprint arXiv:2403.10188*.

Gentry, C. (2009, May). Fully homomorphic encryption using ideal lattices. In *Proceedings of the forty-first annual ACM symposium on Theory of computing* (pp. 169-178).

Goasduff, L. (2021, November 10). *Gartner says cloud will be the centerpiece of new digital experiences*. Gartner. https://www.gartner.com/en/newsroom/press-releases/2021-11-10-gartner-says-cloud-will-be-the-centerpiece-of-new-digital-experiences

Chillotti, I., Gama, N., Georgieva, M., & Izabachène, M. (2020). TFHE: fast fully homomorphic encryption over the torus. *Journal of Cryptology*, *33*(1), 34-91.

Cheon, J. H., Kim, A., Kim, M., & Song, Y. (2017). Homomorphic encryption for arithmetic of approximate numbers. In *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 23* (pp. 409-437). Springer International Publishing.

Cheon, Jung Hee, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. "A full RNS variant of approximate homomorphic encryption." In *Selected Areas in Cryptography–SAC 2018: 25th International Conference, Calgary, AB, Canada, August 15–17, 2018, Revised Selected Papers 25*, pp. 347-368. Springer International Publishing, 2019.

Kim, S., Kim, J., Kim, M. J., Jung, W., Kim, J., Rhu, M., & Ahn, J. H. (2022, June). Bts: An accelerator for bootstrappable fully homomorphic encryption. In *Proceedings of the 49th annual international symposium on computer architecture* (pp. 711-725).

Albrecht, M., Chase, M., Chen, H., Ding, J., Goldwasser, S., Gorbunov, S., ... & Vaikuntanathan, V. (2021). Homomorphic encryption standard. *Protecting privacy through homomorphic encryption*, 31-62.

Voigt, P., & Von dem Bussche, A. (2017). The eu general data protection regulation (gdpr). *A Practical Guide, 1st Ed., Cham: Springer International Publishing*, *10*(3152676), 10-5555.

Pepitone, J. (2011, June 22). *Dropbox says it has fixed security flaw that exposed passwords*. CNN. https://money.cnn.com/2011/06/22/technology/dropbox_passwords/

Regev, O. (2009). On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM (JACM)*, *56*(6), 1-40.

Fan, J., & Vercauteren, F. (2012). Somewhat practical fully homomorphic encryption. *Cryptology ePrint Archive*.

Wang, W., Hu, Y., Chen, L., Huang, X., & Sunar, B. (2012, September). Accelerating fully homomorphic encryption using GPU. In *2012 IEEE conference on high performance extreme computing* (pp. 1-5). IEEE.

Jung, W., Kim, S., Ahn, J. H., Cheon, J. H., & Lee, Y. (2021). Over 100x faster bootstrapping in fully homomorphic encryption through memory-centric optimization with GPUs. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 114-148.

Brakerski, Z., & Vaikuntanathan, V. (2014). Efficient fully homomorphic encryption from (standard) LWE. *SIAM Journal on computing*, *43*(2), 831-871.