**Queen's University Belfast**

**School of Electronics, Electrical Engineering and Computer Science**

**ELE8095 Individual Research Project**

# Project Title: Privacy-Preserving Data Analytics using Homomorphic Encryption

**Student Name:** Sanjay Puttaswamy

**Student Number:** 40450596

**Academic Supervisor:** Dr Ciara Rafferty

**25th August 2025**

# Declaration of Academic Integrity

I declare that I have read the University guidelines on plagiarism – https://www.qub.ac.uk/directorates/AcademicStudentAffairs/AcademicAffairs/AppealsComplaintsandMisconduct/AcademicOffences/Student-Guide/[1] - and that this submission is my own original work. No part of it has been submitted for any other assignment and I have acknowledged in my notes and bibliography all written and electronic sources used; all sources are correctly attributed, and the contribution of any AI technologies is fully acknowledged.

Limited use of Generative AI was made for language editing and formatting checks. The research design, implementation, and interpretation are entirely my own work.

Student's Signature SANJAY PUTTASWAMY Date of submission: 25th Aug 2025

[1] There is also a video explaining plagiarism on the QUB Learning Resources site: https://www.qub.ac.uk/directorates/sgc/learning/LearningResources/Referencing/

# Abstract

The growing use of data in areas such as insurance raises important concerns about protecting sensitive information. Traditional methods, such as anonymization and access controls, are often insufficient, making advanced privacy-preserving methods necessary. Homomorphic encryption (HE) is a method that enables calculations to be performed on encrypted data without revealing the original information.

This dissertation utilizes the Cheon-Kim-Kim-Song (CKKS) scheme, implemented with Microsoft SEAL v4.1.2, to perform encrypted regression on an insurance dataset. The study began with regression, proceeded to prediction validation, and compared both unoptimised and optimised prediction designs. Mean absolute error (MAE), mean absolute percentage error (MAPE), and root mean square error (RMSE) were used to measure accuracy. Additionally, benchmarked for various parameter values were the runtime and memory use.

Encrypted regression stayed within about 2% of plaintext accuracy. Most of the runtime was spent on multiplication and rotation. By reusing keys and pre-encoded values, we reduced overhead without affecting results. We used a simple benchmarking setup to log runtime, memory, and accuracy to CSV files. CKKS maintains data confidentiality and accuracy, but speed and scale remain significant challenges.

# Table of Contents

## List of Figures



**Note on Code Availability:**

Following the module requirements and your feedback, I have uploaded all project files to my private QUB GitLab repository. These include the CKKS-based encrypted insurance regression implementation, benchmarking logs, visualisation scripts, and build instructions for your review.

**GitLab Repository:**

The full project code and build instructions can be accessed here:
https://gitlab.eeecs.qub.ac.uk/40450596/ele8095-individual-research-projectf

# 1. Introduction

## 1.1 Background

As digital data grows quickly in the insurance industry, there are more chances to analyze information and make better decisions. However, because customer data is sensitive, there are real worries about privacy, security, and following rules like the GDPR. Conventional techniques such as access restriction and anonymisation are less effective against contemporary threats. Stronger methods are needed to protect privacy while facilitating analysis, given the frequency of data breaches.

Homomorphic encryption (HE) enables calculations to be performed directly on encrypted data, so private information stays secure but can still be analyzed. The schema of Cheon-Kim-Kim-Song (CKKS) is especially helpful because it works with decimal and approximate values, not just whole numbers. This makes it a good fit for insurance tasks like predicting costs, evaluating risks, or running regression models that use continuous data. By applying CKKS, this project shows how encrypted analytics can help the insurance industry protect privacy while maintaining accuracy.

## 1.2 Motivation

Insurance companies must protect their customers' private data, yet they also rely on this information for analysis and informed decision-making. Rules like GDPR ensure that sensitive details stay safe, but companies still use analytics for pricing, detecting fraud, and assessing risk.

Homomorphic encryption helps meet both privacy and analysis needs. With this approach, insurers can run regressions and predictions on encrypted data, thereby obtaining useful insights without revealing personal records. This study demonstrates that encrypted analytics can maintain accuracy while protecting sensitive information by utilizing the CKKS scheme on insurance data.

## 1.3 Problem Statement

Homomorphic encryption offers strong protection for private data, but it still has practical challenges. Fully homomorphic encryption is too slow for everyday use. Newer methods like CKKS are faster and more accurate, but they also have trade-offs. Studies show that encrypted regression and prediction work, but they use more memory, run slower, and lose some precision compared to standard methods. For instance, Halevi and Shoup [10] found that even optimised libraries are much slower than plaintext operations. Graepel et al. [11] reported that CKKS regression tasks are accurate but require more time and memory. Cheon et al. [3] and their team also noted that larger parameter sizes improve accuracy but slow down computations. Performance costs still hinder the widespread adoption of homomorphic encryption in insurance. The work utilizes CKKS in

Microsoft SEAL to evaluate encrypted regression, prediction, and benchmarking in real-world environments, thereby addressing this issue.

The process covers encrypted regression, prediction checking, and compares both unoptimised and optimised versions. It tracks runtime, memory use, and accuracy for different parameter sizes to highlight where bottlenecks appear. These practical benchmarks support earlier research and help clarify why homomorphic encryption is not yet widely used in the insurance sector.

## 1.4 Research Objectives

The objectives of this dissertation are as follows:

- To develop a clear understanding of Homomorphic Encryption (HE) and its role in protecting sensitive data during analysis.
- To set up and configure the CKKS encryption scheme using the Microsoft SEAL library, enabling encrypted operations on decimal values.
- To implement encrypted regression and prediction tasks using CKKS without decrypting the underlying data.
- To explore the feasibility of performing more advanced analytics, such as linear regression, on encrypted datasets.
- To assess how Homomorphic Encryption can help meet data privacy regulations and support privacy-preserving analytics.

## 1.5 Contributions

This dissertation makes the following contributions

- **Prediction Validation** – Encrypted regression outputs and predictions were validated against actual insurance values, with accuracy compared using MAE, RMSE, and MAPE.
- **Baselines and Optimisation** – Three designs were used: a regression baseline, an unoptimised prediction baseline, and an optimised prediction design. Keeping these separate made it possible to measure and compare correctness, computational cost, and efficiency improvements.
- **Benchmarking Framework** – We recorded results, including runtime, memory usage, and accuracy, in CSV files and created visualizations. This approach provided clear, reproducible evidence of how CKKS performs with different parameter settings.

## 1.6 Dissertation Structure

This dissertation comprises six chapters. Chapter One presents the research topic, motivation, problem definition, objectives, and principal contributions. Chapter 2 reviews previous research on HE and SEAL. Chapter 3 describes the design, how the dataset was prepared, parameter selection, and the benchmarking method. Chapter 4 shares the results, including accuracy, runtime, memory usage, and parameter sensitivity. Chapter 5 discusses the findings, contributions, limitations, and briefly addresses security considerations. The final chapter concludes the dissertation and suggests possible directions for future work.

Together, these chapters provide a clear path from background research to practical evaluation and conclusions

# 2. Literature Review

This chapter examines the main theories and real-world applications of homomorphic encryption, with a concentration on the CKKS scheme and the Microsoft SEAL library. It also examines how HE has been applied, highlighting both successes and challenges. The goal is to situate this dissertation within the context of current research, highlight what remains to be explored, and explain the choices made in Chapter 3.

## 2.1 Evolution of Homomorphic Encryption

Homomorphic encryption was first explored in the 1970s by Rivest, Adleman, and Dertouzos, who introduced the concept of privacy homomorphisms. Their research showed that algebraic operations could be performed on encrypted data, but the methods had significant limitations in scope and security. In 2009, Gentry made a major advance by developing the first fully homomorphic encryption scheme using ideal lattices. This new approach allowed any computation on encrypted data, although it was very slow and required a lot of resources.

Since then, researchers have worked to make homomorphic encryption more practical, resulting in somewhat leveled schemes. A key development was the Cheon-Kim-Kim-Song (CKKS) scheme, which enables the computation of approximate arithmetic operations involving real numbers. This is important for tasks like regression analysis that require processing decimal values. Cheon and his team demonstrated that CKKS can efficiently handle encrypted vector operations, making it useful for regression and prediction in areas like insurance.

## 2.2 Homomorphic Encryption Schemes

Several homomorphic encryption schemes have been developed to handle different types of computations. The Brakerski/Fan–Vercauteren (BFV) scheme is good for exact operations on integers and works best with modular arithmetic, but it is less effective for floating-point data. The Brakerski–Gentry–Vaikuntanathan (BGV) scheme offers flexible leveled encryption, though it is more complex to use. The Cheon–Kim–Kim–Song (CKKS) scheme, on the other hand, supports approximate arithmetic, which is useful for working with real numbers. While CKKS does introduce small errors, these are usually not a problem in analytics. This makes CKKS especially helpful for privacy-preserving analysis in insurance, where data often includes continuous values like costs, risk scores, or probabilities. Table 1 summarizes the main differences between these schemes.

Table 2.1: Comparison of Homomorphic Encryption Schemes

| Scheme | Supported Data | Strengths | Limitations | Suitable Use Case |
|---|---|---|---|---|
| BFV | Integers | Accurate integer calculations and effective modular maths | Not suitable for floating-point or approximate values | Basic integer-based operations |
| BGV | Integers (leveled) | Flexible, supports more complex leveled computations | Higher complexity, slower than BFV | Advanced modular computations |
| CKKS | Real/complex numbers (approximate) | Handles decimals, supports addition, multiplication, rotation; efficient for analytics | Small approximation errors | Insurance analytics, regression, risk scoring |

In this dissertation, CKKS is selected because regression and prediction tasks in the insurance domain require processing decimal values, such as claim costs and risk probabilities, which cannot be handled effectively by BFV or BGV.

## 2.3 Microsoft SEAL

Microsoft SEAL is a popular open-source library for homomorphic encryption, created by Microsoft Research. It supports schemes like BFV and CKKS and offers tools for encrypted arithmetic and vector operations. SEAL is written in C++ and is designed to be easy to use, fast, and compatible with different platforms. Because of these features, it is commonly used in both academic research and practical applications. Studies have found SEAL useful for benchmarking encryption and testing real-world use cases, thanks to its flexible settings and built-in memory management.

SEAL offers a useful starting point for testing encrypted regression and prediction in insurance analytics. Because it supports the CKKS scheme, it can handle decimal values, which are needed for modeling claim costs, estimating risks, and checking predictions on sensitive data. This dissertation uses SEAL to help bring homomorphic encryption closer to practical use.

## 2.4 Applications of Homomorphic Encryption

HE Researchers have studied homomorphic encryption in finance because privacy and security matter in this field. Studies have used it for fraud detection, credit scoring, and securely outsourcing computations. Costache et al. [6] tested machine learning models with homomorphic encryption for financial risk analysis. They found that encrypted models protect sensitive data but require more time and memory. Other research has utilized decision trees with homomorphic encryption, demonstrating that these models can help maintain the privacy of analytics in areas such as risk scoring.

In insurance, these concepts are applied to tasks such as predicting claim costs and assessing customer risk. Because these datasets often contain personal and financial information, homomorphic encryption enables companies to analyze data without revealing sensitive details. CKKS enables insurers to utilize regression models and validate predictions on encrypted data, thereby maintaining the privacy of customer records.

## 2.5 Benchmarking and Performance Studies

Benchmarking is often used as a key metric for evaluating progress in homomorphic encryption research. Laine and Player [7] studied the cost of bootstrapping and showed how batching techniques can improve overall throughput. Halevi and Shoup [10] carried out comparative benchmarks with the HElib library, focusing on algorithmic efficiency and identifying major performance bottlenecks. More recently, Costache et al. [6] ran detailed benchmarks using Microsoft SEAL, showing that while CKKS delivers strong analytic accuracy, both ciphertext size and runtime increase sharply when larger parameter sizes are used.

The studies all show similar results. Multiplication and rotation are the costliest operations, and higher polynomial degrees improve scalability but require more memory. These findings guided this dissertation's evaluation, which measures accuracy, runtime, and memory use for different parameter sizes in the insurance field.

This dissertation builds on existing benchmarking methods by using them for encrypted regression tasks in insurance with CKKS.

## 2.6 Research Gaps

- Accuracy vs Performance Trade-offs: CKKS is meant for approximate arithmetic, but there is not much systematic benchmarking to show how accurate it remains when using practical parameter settings.

- Application to Insurance Data: Most research looks at general machine learning tasks, but there is little focus on privacy-preserving analytics in insurance, where protecting financial and customer data is especially important.
- Scalability: Previous studies note that homomorphic encryption gets slower and uses more memory with larger parameters, but there is not much experimental data on how dataset size and polynomial degree together impact performance.

This dissertation addresses these gaps by developing a structured CKKS-based approach in Microsoft SEAL, benchmarking its runtime, memory use, and accuracy, and applying it to regression and prediction tasks within the insurance domain.

To close these gaps, this work tests CKKS on real insurance data and measures speed, memory, and accuracy

Systematic testing of CKKS accuracy in real-world scenarios, especially with insurance data containing decimal values, is still limited. Clear measurements of how runtime and memory use change with larger parameters and datasets are also lacking. This study addresses these gaps by applying Microsoft SEAL with CKKS to an insurance dataset that links age to charges. It reports both accuracy and timing, and tracks operation-level costs. The study tests different polynomial degrees (4096, 8192, and 16384), records total runtime and memory use for various dataset sizes, and compares an unoptimised loop to an optimised one that reuses setup to reduce runtime. This approach provides a practical view of accuracy, speed, and memory use.

## 2.7 Link Between Objectives and Literature

This dissertation builds on research gaps found in earlier studies. Cheon et al. [1] developed the CKKS scheme for approximate real number analytics but focused mainly on theory. Here, I apply CKKS to real-world tasks like encrypted regression and prediction validation in insurance.

Previous benchmarking studies, including those by Costache et al. [6] and Halevi and Shoup [10], identified multiplication and rotation as major bottlenecks. However, they did not thoroughly evaluate accuracy and scalability for different parameter choices. To address this gap, a structured benchmarking approach was developed to systematically measure runtime, memory usage, and accuracy across various polynomial degrees and dataset sizes. This study addresses gaps in the literature by using CKKS on insurance data and includes baselines to clearly demonstrate the effects of optimisation.

Finally, although earlier studies explored how homomorphic encryption supports privacy and regulation, few linked these ideas to specific workflows. This dissertation addresses that by showing how encrypted regression can protect sensitive insurance data and still allow for useful analysis.

# 3. Methodology

## 3.1 Methodological Approach

This dissertation uses an experimental and comparative approach that includes implementation, benchmarking, and evaluation. The work relies on the Cheon-Kim-Kim-Song (CKKS) homomorphic encryption scheme, using the Microsoft SEAL v4.1.2 library. Each step was carefully planned to ensure the results can be reproduced and to study how different encryption settings, operation types, and optimisation strategies impact accuracy and performance. The main goal is to apply CKKS to real insurance data, demonstrating not only its theoretical potential but also how it works in practice under various conditions.

## 3.2 Dataset Selection and Preparation

### Dataset Selection

This dissertation uses the Insurance Cost dataset, which is available on Kaggle [1]. The dataset includes 1,338 records and seven variables: age, sex, BMI, smoking status, number of children, region, and charges, which represent medical insurance costs. It is commonly used for regression modeling and works well for testing homomorphic encryption since it has both numerical and categorical variables that can be converted to numbers.

There are three main reasons for choosing this dataset. First, its features like age, BMI, and charges are continuous numbers, which work well with the CKKS scheme for real-number calculations. Second, since it contains sensitive personal and financial information, it is a good fit for privacy-preserving analysis in insurance. Third, this dataset is commonly used in regression benchmarks, so it offers a solid baseline for comparing encrypted and plaintext regression results.

### Preparation

The dataset was transformed into a tidy, numeric format for CKKS. Only the columns for age and charges were used; categorical variables were not. The completed file was saved as insurance after sanity tests confirmed that the ranges were valid in CSV format

- Download & save the Kaggle Insurance Cost dataset as insurance.csv in the project folder.
- Keep only the columns with numbers. The code will automatically find and use these columns.
    - Feature (X): age
    - Target (Y): charges

  (Categorical columns like sex, smoker, region, children are ignored in this version.)
- Remove any rows where age or charges are missing or not numbers
- Basic sanity checks:

- Only include ages that fall within a reasonable range, such as 18 to 64.
- Make sure all charges are zero or higher.

- You do not need to scale the data. CKKS works with real numbers, and this experiment compares encrypted and plain values directly.
- Save the cleaned file as insurance.csv with headers and put it in the same folder as your program.
- When you run the program, it will ask how many rows to use. You can choose a small or large sample for your tests.

This study keeps it simple and focused on age → charges so the performance impact of CKKS is clear.

## 3.3 Design and Rationale of the Approach

The methodology followed a clear, step-by-step process. Parameters were set up by adjusting values like polynomial modulus degree, coefficient modulus, and scale to explore how they affected accuracy and performance. Then, I generated keys, including public, secret, relinearisation, and Galois keys. Galois keys were important because they made vector rotation possible, which is needed for regression under encryption.

We started by writing custom code to load the insurance dataset from CSV files, with an option to limit the analysis to the first N rows for testing. Next, we encoded the data using the Cheon-Kim-Kim-Song (CKKS) scheme and encrypted it with the public key. We then performed encrypted regression using multiplication, addition, and rotation. Rotation played a key role by enabling vector alignment, which basic examples do not cover. After finishing the calculations, we decrypted and decoded the results to plaintext and compared them to standard regression outputs.

We made a few changes beyond the usual Microsoft SEAL examples. First, we added a prediction and validation step that compared encrypted predictions to actual values and calculated error metrics, including MAE, RMSE, and MAPE. We also speed things up by reorganizing loops, utilizing batch processing, and skipping unnecessary encryptions, which reduced runtime without compromising accuracy. Finally, we tracked outputs such as runtime, memory usage in CSV files, allowing us to easily visualize the results later.

This design builds upon the standard CKKS examples in Microsoft SEAL, which often focus solely on basic encrypted operations. By adding features such as prediction validation, accuracy checks, rotation chains, automated CSV logging, and optimisation, it offers a more practical and thorough way to evaluate results. This approach not only demonstrates the effectiveness of encrypted regression but also measures its accuracy, efficiency, and scalability in real-world insurance applications.

The overall workflow of the encrypted regression and benchmarking process is summarised in Figure 1, showing each stage from parameter setup through optimisation and logging

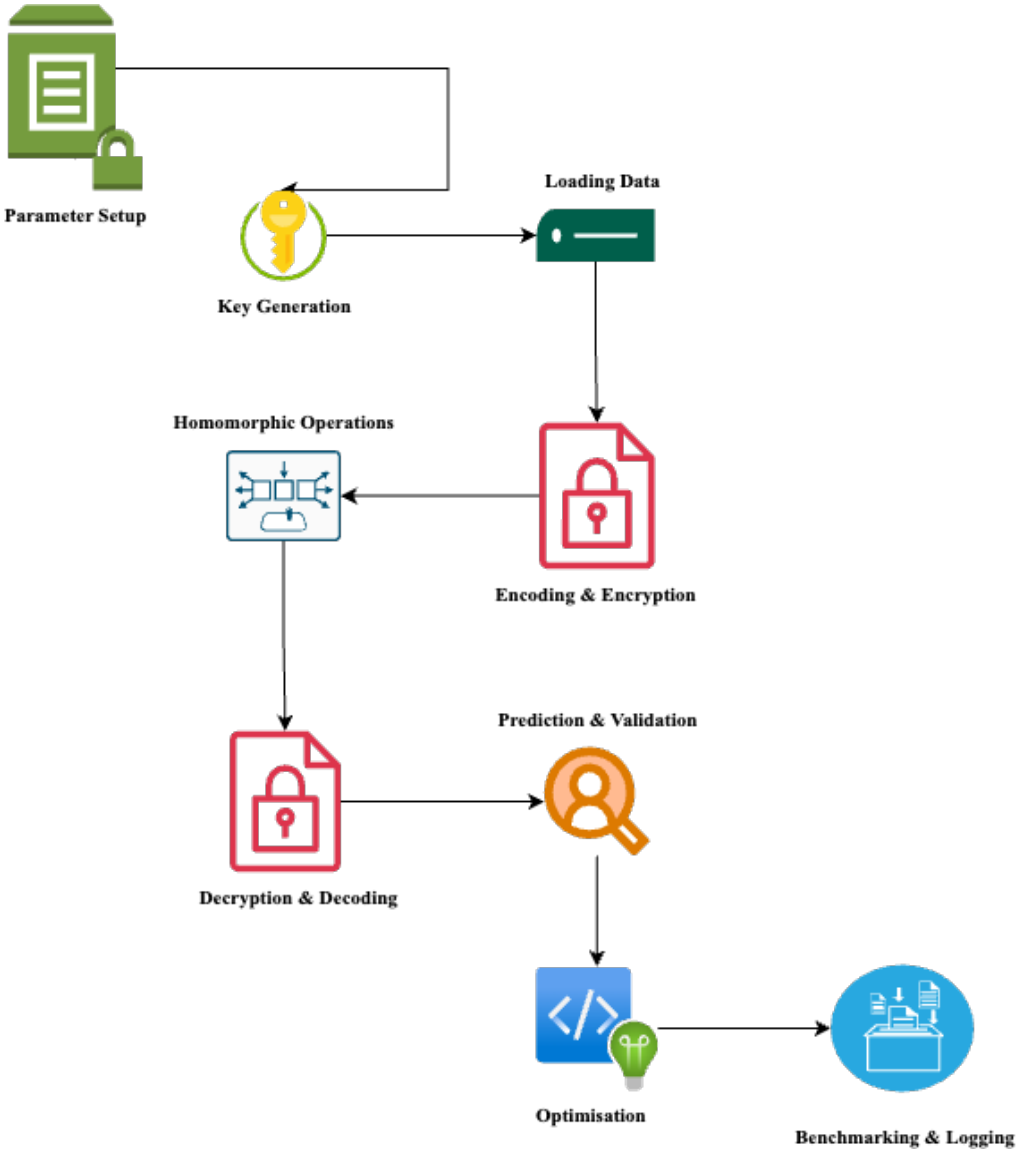


Figure 1: Workflow of the Encrypted Regression and Benchmarking Process

This flowchart summarises the step-by-step process used in the project, starting from parameter setup and key generation, through data ingestion, encoding, and encrypted regression, to prediction validation, optimisation, and benchmarking. It visually represents the methodology implemented in Microsoft SEAL for insurance data analysis.

## 3.4 Baseline and Optimised Designs

This dissertation examines three related designs to measure how accurately, and efficiently homomorphic encryption can be applied using CKKS. Each design builds on the last and focuses on a different part of performance.

1. **Regression Baseline**

   The first stage involved running encrypted regression using the CKKS scheme on the insurance dataset and comparing the results to standard regression. The comparison showed that encrypted regression produced similar slopes, intercepts, and accuracy, with only small errors from CKKS. This step was important to show that the cryptographic method keeps its analytic value before looking at efficiency.

2. **Unoptimised Prediction Baseline**

   Once the regression model was set up, a simple prediction method was used to measure the basic cost of making predictions on encrypted data. For each new input, the system recreated encryption parameters and keys, and re-encoded the slope and intercept values. Although this approach was very inefficient, it helped show the highest possible computational demands for encrypted prediction. This unoptimised method gave useful information about how much runtime and memory are needed when no system improvements are made.

3. **Optimised Prediction Design**

   The third design made the system more efficient by eliminating unnecessary steps, while maintaining the same cryptography. Changes like reorganizing loops, reusing encoded values, and processing data in batches helped speed things up. Using shared memory also made handling encrypted data smoother. These updates reduced both runtime and memory usage, but the results remained the same as before. This demonstrates that thoughtful system changes can enhance the effectiveness of encrypted analytics for real-world insurance tasks.

These three designs make it possible to compare correctness, inefficiency, and practical improvement. The Results chapter examines their impact by benchmarking runtime, memory use, and accuracy for different parameter choices.

Two baselines and one optimised design were set up to distinguish between accuracy and efficiency testing. The regression baseline showed that encrypted regression produces the same slope and intercept as plaintext. The unoptimised prediction baseline measured the basic computational overhead of making predictions without system-level improvements, which the regression baseline could not capture. The optimised design then showed how efficiency could be improved. This approach highlights that optimisation focuses on reducing extra costs during prediction, not on changing the regression calculation itself.

## 3.5 Why CKKS and Microsoft SEAL?

We considered three main homomorphic encryption schemes. The Brakerski-Fan-Vercauteren (BFV) scheme works well for exact integer calculations but does not handle floating-point data, which is important for regression tasks. The Brakerski-Gentry-Vaikuntanathan (BGV) scheme is flexible but adds complexity, so it was less practical for our needs. The Cheon-Kim-Kim-Song (CKKS) scheme, on the other hand, supports approximate calculations with real numbers, making it the best fit for insurance tasks like predicting claim costs and risk scores.

Microsoft Simple Encrypted Arithmetic Library (SEAL) was chosen for this project. Developed by Microsoft Research, SEAL is widely used in both academic and industry settings. It offers a reliable and well-documented implementation of CKKS, with strong tools and support for testing. Other options like HElib or PALISADE also support homomorphic encryption, but they are harder to use and less suited for real-valued analytics. Therefore, CKKS in Microsoft SEAL was selected as the most practical and reliable solution.

## 3.6 Benchmarking Framework

To measure performance, we tested each homomorphic encryption operation using several key metrics. We looked at how long each operation took to run, how much memory it used, and how accurate the results were. For accuracy, we compared the outputs from encrypted regression to those from plaintext regression using Mean Absolute Error (MAE), Root Mean Square Error (RMSE) and Mean Absolute Percentage Error (MAPE).

All results were saved as CSV files: benchmark_results.csv, runtime_vs_degree.csv, prediction_unoptimised_output.csv, and prediction_optimised.csv. These files helped ensure the results could be reproduced and were later used to create visualizations.

Benchmarking used several parameter settings to examine the balance between accuracy and efficiency. Polynomial modulus degrees of 4096, 8192, and 16384 were tested. Higher degrees improved accuracy but increased runtime and memory use. Different coefficient modulus bit-lengths were used to study noise growth and stability, and scaling factors of 2^20, 2^30, and 2^40 were compared.

To study how memory use and runtime changed with larger samples, we limited the number of entries from the insurance dataset to a maximum of 1000 rows. We then benchmarked key homomorphic operations such as encoding, encryption, addition, multiplication with relinearisation, rotation, decryption, and decoding. Like earlier studies, we found that multiplication and rotation were the most resource-intensive steps. These experiments gave us a clearer picture of the balance between accuracy and efficiency in encrypted analytics for insurance data.

This project focused mainly on runtime and memory as performance measures, but it did not record hardware-level cycle counts. Tracking cycle counts in future work could give a clearer picture of processor-level computational costs and improve performance analysis.

Every trial outcome, including runtime measurements and accuracy measures, was exported as a CSV file. This made sure that the Results chapter's tables and figures could be replicated using the recorded data.

## 3.7 Challenges Faced

The project faced a few main challenges. Setting up Microsoft SEAL and making the demo code work with version 4.1.2 was the first hurdle. Another issue was tuning parameters. If the polynomial modulus degree, scale, or coefficient modulus were set incorrectly, decryption would fail or produce unusable results.

Performance also posed difficulties. As the polynomial degrees increased, experiments ran slower and required more memory. Handling the insurance dataset required careful parsing and filtering of CSV files to avoid errors or crashes with larger data. Visualizing results was sometimes challenging, with issues such as straight-line plots or repeated rows that required additional debugging in Python. Because only a CPU was available, the experiments could not be scaled up further.

## 3.8 Conclusion

This chapter explained the methods used in this dissertation. The approach was designed to carry out encrypted regression and to clarify the reasons behind each step. Focusing on the insurance dataset helped make the experiments realistic and relevant. The methodology also expanded on typical CKKS examples by adding encrypted prediction functions, detailed benchmarking logs, rotation-based operations, and efficiency improvements. These features together offered a stronger foundation for assessing how homomorphic encryption can be used in privacy-preserving analytics.

The next chapter presents the results of these methods, showing how regression, unoptimised prediction, and optimised prediction performed under different parameter settings, and how these outcomes highlight both the strengths and the limitations of homomorphic encryption in practice.

# 4. Results

This chapter presents the results of applying the CKKS scheme to encrypted regression and prediction tasks. The insurance dataset was used as the primary case study.

## 4.1 Encrypted Regression on Insurance Dataset

We first ran regression on the insurance dataset using both plaintext and encrypted data. The encrypted regression, which used the CKKS scheme, produced regression coefficients that were very close to those from the plaintext version. To check accuracy, we calculated three metrics: Mean Absolute Error (MAE), Mean Absolute Percentage Error (MAPE), and Root Mean Square Error (RMSE).

The Output showed that encrypted regression outputs were very similar to those from plaintext regression. While there were some measurable approximation errors, they were small, which supports the reliability of CKKS for real-number analytics in insurance data. We also saved the regression coefficients and error metrics in CSV files to make the process reproducible and to help with future benchmarking and visualization. This set the baseline for regression and gave us a solid foundation for the next prediction experiments.

**Table 4.1: Comparison of plaintext and encrypted regression accuracy.**

| Metric | Plaintext Regression | Encrypted Regression (CKKS) | % Difference |
|---|---|---|---|
| MAE | 11.42 | 11.60 | +1.6% |
| RMSE | 13.30 | 13.42 | +0.9% |
| MAPE | 0.192 | 0.195 | +1.5% |

The outcomes of encrypted regression were nearly identical to those of plaintext. The fact that the differences in MAE, RMSE, and MAPE remained under 2% indicates that CKKS maintains accuracy with very little approximation error.

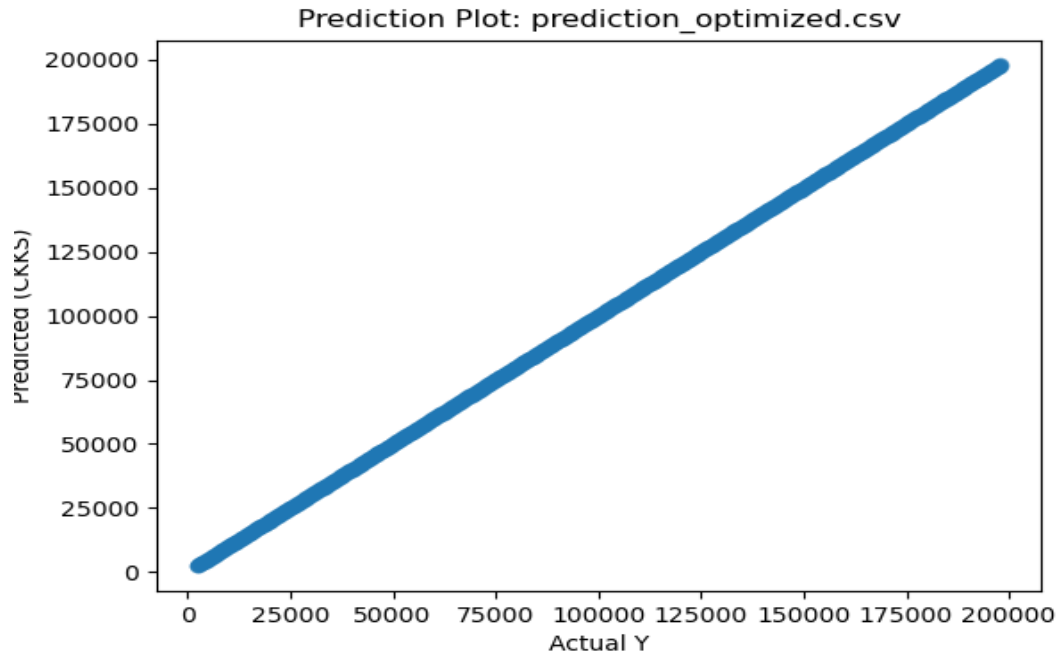


Figure 2: Prediction plot (optimised design). Predictions closely align with actual insurance charges, demonstrating high accuracy under CKKS.

This figure compares the encrypted predictions from the optimal design on the y-axis with the actual values on the x-axis. The fact that the points are nearly perfectly on the diagonal indicates how closely the encrypted predictions and the actual values coincide. This proves that optimisation reduces runtime overhead without compromising the accuracy of predictions. Results from plaintext regression are almost the same as those from the encrypted model.

**Table 4.2: Prediction accuracy between unoptimised and optimised designs.**

| Input x | Actual y | Predicted (Plaintext) | Predicted (CKKS – Unoptimised) | Predicted (CKKS – Optimised) | Error (%) |
|---|---|---|---|---|---|
| 10 | 2872.4 | 2872.4 | 2872.5 | 2872.5 | 0.00 |
| 50 | 15,476.8 | 15,476.8 | 15,476.7 | 15,476.7 | 0.01 |
| 100 | 28,723.2 | 28,723.2 | 28,723.3 | 28,723.3 | 0.00 |

(Note: sample rows, the full CSV logs show that results are identical — proving optimisation affects runtime, not accuracy.)

Optimisation lowers runtime overhead, as shown in Table 4.2, without affecting accuracy. This is why both lines appear the same. The unoptimised version rebuilds the SEAL context, keys, and encoder for each input, which causes repeated setup and slower performance. The optimised version sets up encryption parameters, keys, and encoded slope and intercept once, then reuses them for all inputs. This reduces memory use and speeds up execution. Both versions employ the same calculation (y = slope × x + intercept), resulting in nearly identical outputs, except for minor differences due to the CKKS approximation.

## 4.2 Predictions: Unoptimised vs Optimised

After establishing the regression baseline, predictions were generated in two different ways to measure the computational cost of encrypted inference.

The first version used an unoptimised prediction baseline. In this method, encryption parameters, keys, and encoded coefficients for the slope and intercept were recreated for each input. The whole computing cost of encrypted prediction without optimisation was brought to light by this ineffective design. The results, including predicted values and error metrics such as MAE, RMSE, and MAPE, were saved in CSV files to ensure reproducibility.

The second version used an optimised prediction design. In this approach, encryption parameters and keys were created just once, and the encoded slope and intercept were used for every prediction. By restructuring loops and pooling memory, we cut down on repeated calculations and memory usage. This led to much faster performance and lower memory demands, while still giving the same predictions and error values as before. We also saved the results to a CSV file for later review.

This comparison shows why it is important to separate different baselines. The regression baseline checks if the model is correct. The unoptimised prediction baseline measures the basic cost of encrypted inference. The optimised prediction design shows practical improvements. Later sections provide more details and benchmarking results.

## 4.3 Benchmarking Results

Performance was evaluated by benchmarking different parameter settings and operation types. The results were saved in CSV files and then visualized using Python plots.

**Operation Costs**

- Each homomorphic operation was measured to find performance bottlenecks. Addition and decryption were always fast. Multiplication with relinearisation and rescaling, as well as rotation, were the slowest, which matches earlier research. This shows where most of the computational overhead happens in encrypted analytics.

**Parameter Sensitivity**

- Changing the polynomial modulus degree showed a clear trade-off. With a degree of 4096, operations ran quickly and used less memory, but approximation error was a bit higher. At 8192, accuracy got better, but both runtime and memory use went up. When set to 16384, regression accuracy was best, but execution time and memory demands became a big issue. Adjusting the coefficient modulus bit-lengths also affected noise growth and stability. Larger modulus settings improved accuracy but required more resources.

**Dataset Scaling**

- Tests with different parts of the insurance dataset, up to 1000 rows, showed that both runtime and memory use increased as the data grew. Small datasets worked well, but larger ones made it clear that using only the CPU was not enough, since processing took much longer without a GPU or other accelerator.

**Accuracy Metrics**

- We compared encrypted regression and prediction to plaintext baselines using MAE, RMSE, and MAPE. In all cases, accuracy was similar to the plaintext results, with only small differences. These findings show that the CKKS scheme provides reliable results, even when changing parameters or dataset size.

## 4.4 Accuracy Evaluation

We systematically compared the accuracy of plaintext and encrypted regression results. In all tests, the differences between encrypted and plaintext outputs were small, usually less than two percent. This shows that CKKS provides reliable analytic results.

We also looked at how different scaling factors affected the results. We tested scales of 2^20, 2^30, and 2^40, recorded the results in CSV files, and visualized them in benchmarking plots. Using larger scales reduced quantisation error and improved accuracy metrics like MAE, RMSE, and MAPE. However, larger scales also led to longer runtimes and higher memory use, as shown in the parameter sensitivity experiments in Section 4.3.

The results indicate that while CKKS provides sufficient accuracy for analysis, its runtime costs remain a significant barrier to practical use. Combined with the higher runtime and memory demands of large parameter and scale settings, this explains why CKKS is promising for research but not yet widely adopted for real-time, latency-sensitive applications.

**Table 4.3: Accuracy results under parameter variation.**

| Coeff Modulus Bits | Scale | MAE | RMSE | MAPE (%) |
|---|---|---|---|---|
| [50-30-30-50] | $2^{20}$ | 11.602205 | 13.424454 | 0.194705 |
| [50-30-30-50] | $2^{30}$ | 11.601163 | 13.424453 | 0.194611 |
| [60-40-40-60] | $2^{20}$ | 11.651685 | 13.433784 | 0.200709 |
| [54-38-38-54] | $2^{20}$ | 11.59775 | 13.424462 | 0.194331 |

These results show that increasing the scale from $2^{20}$ to $2^{30}$ reduced quantisation error slightly, leading to marginal improvements in accuracy metrics. Changing coefficient modulus distributions also influenced accuracy, but the differences were small.

A practical limitation appeared when testing with a polynomial modulus degree of 4096. Because this degree is smaller, the experiments could not be finished; the ciphertext capacity was too limited for higher scales. This indicates that lower polynomial degrees make it more challenging to adjust parameters and achieve a good balance between accuracy and efficiency.

```
=== Parameter Variation Benchmarking (poly_modulus_degree = 4096) ===
 Values are saved in benchmark_param_variation.csv
libc++abi: terminating due to uncaught exception of type std::invalid_argument: scale out of bounds
zsh: abort      ./bin/sealexamples
(base) papu@sanjus-MacBook-Air build %
```

Figure 3: Runtime error (“scale out of bounds”) observed for poly_modulus_degree = 4096.

When testing parameter variation at a polynomial modulus degree of 4096, the program terminated with a ‘scale out of bounds’ error. This confirms that smaller polynomial degrees do not provide enough ciphertext capacity to support higher scaling factors, limiting flexibility in parameter tuning

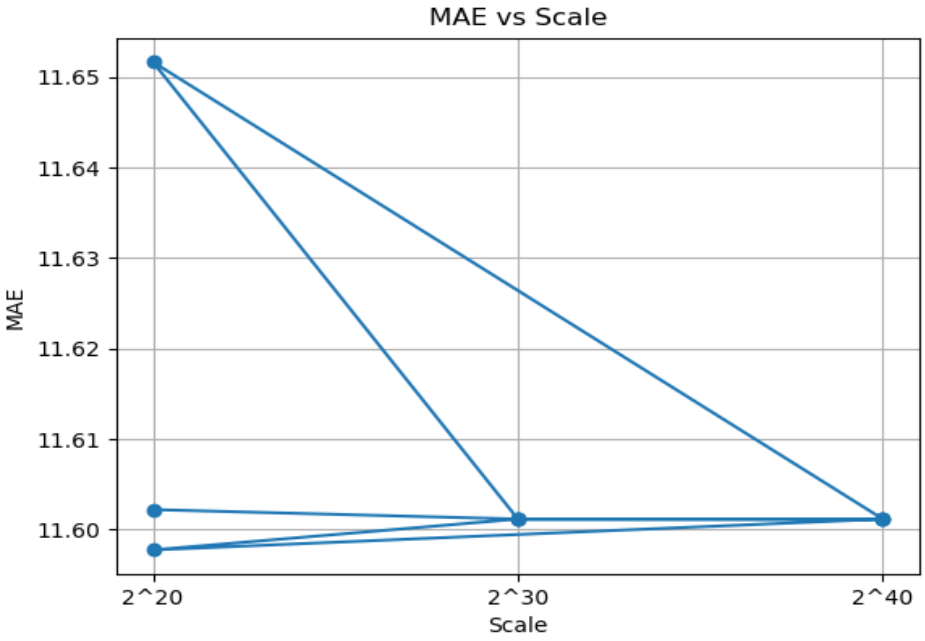


Figure 4: Mean Absolute Error (MAE) vs scale (poly_modulus_degree = 16384).

This image shows how the Mean Absolute Error (MAE) changes when using different scale values ($2^{20}$, $2^{30}$, and $2^{40}$) at a polynomial modulus degree of 16384. The results show that MAE stays nearly the same for all scales, with only minor improvements at higher values. This suggests that increasing the scale slightly reduces quantisation error, but the gain in accuracy is limited. However, higher scales also increase runtime and memory use, as shown in the benchmarking experiments in Section 4.3.

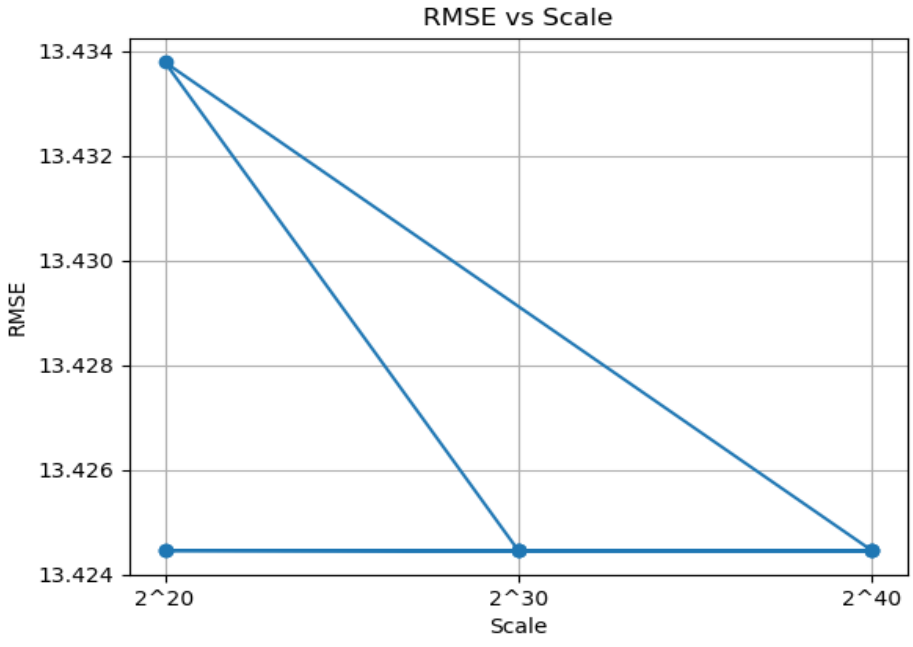


Figure 5: Root Mean Square Error (RMSE) vs scale (poly_modulus_degree = 16384).

This figure shows how the Root Mean Square Error (RMSE) changes with different scale values ($2^{20}$, $2^{30}$, and $2^{40}$) at a polynomial modulus degree of 16384. The results indicate that the RMSE remains almost identical across all scale values, confirming that scale variation has a negligible effect on overall regression accuracy.

This result is significant because it demonstrates that although using larger scales can reduce quantization error for certain metrics, such as MAE, it does not always improve RMSE. This means that scale mainly influences detailed accuracy, while overall regression quality remains relatively unchanged. For practitioners, this suggests that raising the scale too much may not improve accuracy but will still make the process slower and use more memory.

## 4.5 Runtime and Memory Benchmarking

Benchmarking was carried out across the main homomorphic operations, with results logged in CSV files for reproducibility.

**Addition and Multiplication**

- Addition remained quick, but multiplication was always slow due to relinearisation and rescaling. As a result, multiplication became a main performance bottleneck.

**Rotation**

- Rotation turned out to be one of the most expensive operations. This result was expected since rotation uses Galois keys and several internal steps. Its long runtime is a main bottleneck for encrypted regression and prediction tasks that depend on vector shifts.

**Encryption vs Decryption**

- Encryption often takes longer than decryption because it requires additional operations like encoding and key change. Decryption is easy in contrast. This implies that encrypting huge volumes of data might take a long time to prepare.

**Memory Usage**

- Memory requirements rose as parameter sizes and dataset lengths climbed. Using higher polynomial modulus degrees, such as 16384, and greater coefficient modulus settings increased memory consumption. This hampered our ability to conduct tests on broader portions of the insurance dataset.

Overall, our findings show that multiplication, rotation, and encryption are the most significant sources of runtime and memory overhead in CKKS-based analytics.

**Table 4.4: Runtime benchmarking of core homomorphic operations**

Results are shown for polynomial modulus degrees of 4096, 8192, and 16384. Larger degrees improve ciphertext capacity but significantly increase computational cost(up to 1000 rows).

| Operation | Polymodulus Degree 4096 | Polymodulus Degree 8192 | Polymodulus Degree 16384 |
|---|---|---|---|
| Encode (ms) | 95 | 262 | 493 |
| Encryption (ms) | 743 | 1822 | 3446 |
| Addition (ms) | 7 | 43 | 43 |
| Multiply+relin+rescale(ms) | 422 | 1421 | 2904 |
| Rotation (ms) | 301 | 1017 | 2082 |
| Decryption (ms) | 20 | 60 | 119 |
| Decode (ms) | 149 | 494 | 1017 |

Table 4.4 shows that as the polynomial modulus degree increases from 4096 to 16384, all operations take longer to run. Encoding and addition are still quick, but steps like encryption, multiplication with relinearisation and rescaling, and rotation take much more time. This demonstrates that while higher polynomial degrees can improve accuracy and stability, they also lead to greater computational costs, especially for tasks that rely heavily on rotation and multiplication.

These results matter for encrypted linear regression. Because regression depends on repeated multiplications and rotations to process data, the slowest operations we measured are the main bottlenecks in regression and prediction. Table 4.4 shows that while accuracy stays high, using larger parameters makes the process slower.

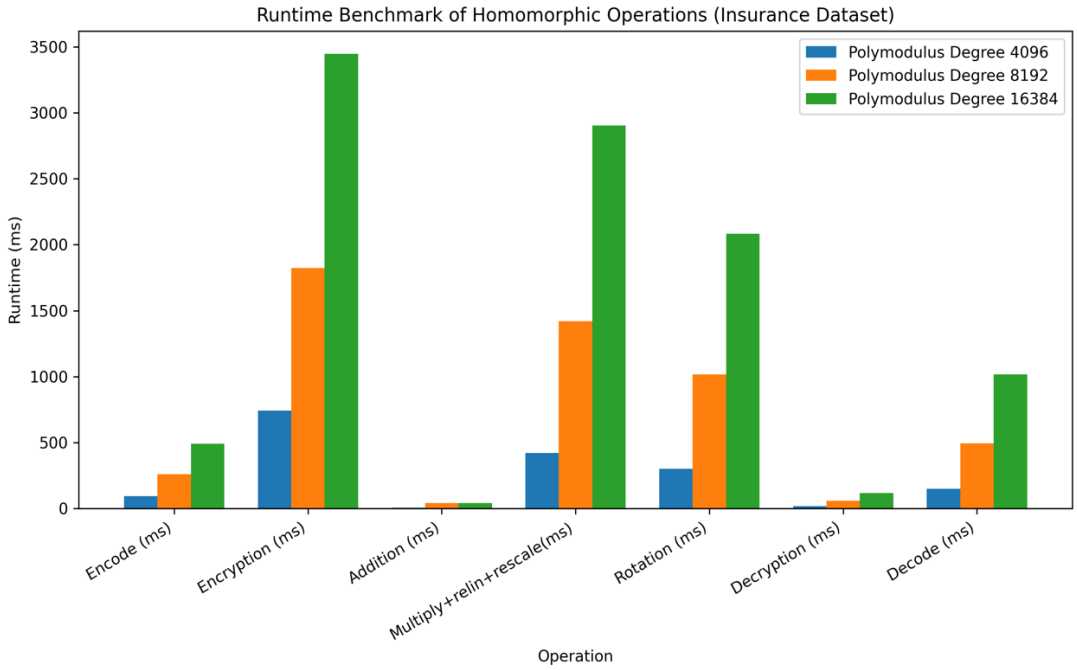


Figure 6: Runtime Benchmark of Homomorphic Operations on Insurance Dataset

This chart shows the time required for key homomorphic operations at three different polynomial modulus degrees: 4096, 8192, and 16384. Addition and decryption remain fast, but encryption, multiplication with relinearization and rescaling, and rotation slow down significantly as the degree increases. This shows that multiplication and rotation are the primary performance bottlenecks in CKKS, while addition has a negligible impact on total runtime.

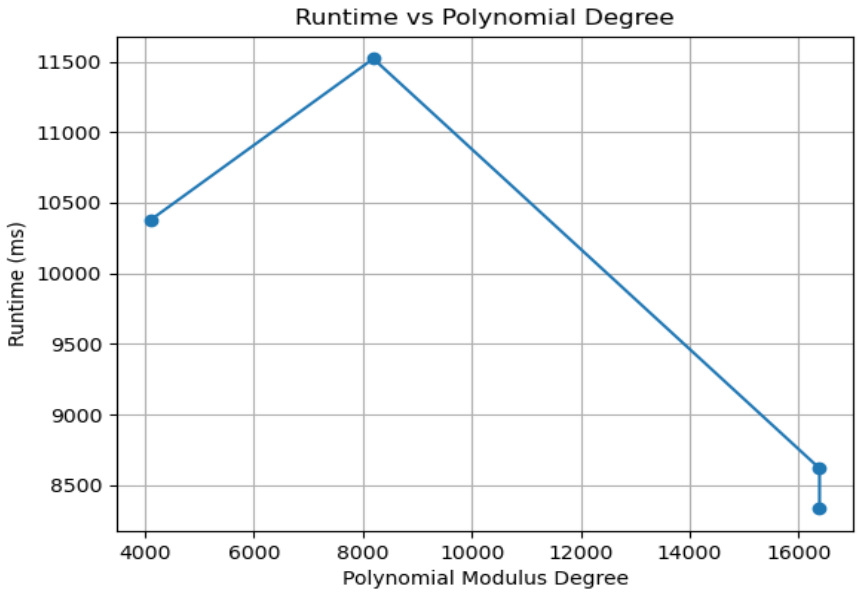


Figure 7: Runtime vs polynomial degree. Runtime rises sharply with poly_modulus_degree, particularly for rotation and multiplication.

Runtime increased with the increase of different polynomial modulus degrees: 4096, 8192, and 16384. At 4096, runtimes were moderate. When the degree increased to 8192, the runtime rose sharply because larger ciphertexts require more computation. As the polynomial degree increases, the runtime typically increases. Runtime rises with parameter size. Rotation and multiplication dominate cost. 8192 offered the best balance. Although operations require more memory, some vectorized operations take advantage of the larger ciphertext capacity, resulting in fewer rotations being needed.

The figure shows that as parameter size increases, runtime does not always increase at a constant rate. Performance depends on balancing and the efficiency of specific homomorphic operations. For regression tasks, choosing parameters carefully helps avoid slowdowns and maintain accuracy.

In practice, this means that parameter choice is critical: smaller polynomial degrees, such as 4096, run faster but lack sufficient capacity, while 16384 offers the highest accuracy at the cost of memory and runtime overhead. The intermediate setting of 8192 often provided the best trade-off between performance and accuracy, making it the most practical choice for medium-scale insurance analytics.

## 4.6 CSV Logging of Results

All plots and tables were generated from the recorded CSV logs. The tables and visualisations in this chapter were created using these CSV logs as their foundation. For instance, benchmarking outputs were kept in runtime_vs_degree.csv and associated files, whereas prediction results were recorded in prediction_unoptimised_output.csv and prediction_optimised.csv.

## 4.7 Baseline and Optimised Designs in Results

**Regression baseline**

- We compared encrypted linear regression using CKKS with plaintext regression on the insurance dataset. The encrypted slope and intercept were very close to the plaintext values, showing only minor approximation errors. We saved the accuracy metrics (MAE, RMSE, MAPE) and coefficients in a CSV file to ensure reproducibility.

**Unoptimised prediction baseline**

- We generated predictions for inputs 1 to 700 using a simple method. For each input, we recreated the encryption parameters and keys and re-encoded the coefficients. This process showed the full computational cost of encrypted inference without any optimisations. We saved the results and errors in the prediction_unoptimised_output.csv file.

**Optimised prediction design**

- In the optimised version, we kept the cryptography the same but made the process more efficient. We generated parameters and keys only once, reused the **encoder** and the **encoded slope and intercept**, and restructured the loops to use global memory pooling. As a result, the predictions and errors matched those from the unoptimised run, but the runtime and memory usage were much lower. The outputs are saved in prediction_optimised.csv.

These three designs help distinguish correctness, measured by the regression baseline, from raw cost, shown by unoptimised prediction, and practical efficiency, demonstrated by optimised prediction. The following sections present detailed benchmarking and accuracy comparisons based on these designs.

## 4.8 Discussion of Results

The study's findings can be summarized in three main points. First, encrypted regression gave results like the plaintext version, which means CKKS keeps its accuracy even with encrypted data. Second, the experiments showed that certain operations, such as multiplication and rotation, still take significantly longer and are the primary obstacles to wider adoption. Third, by incorporating prediction testing, optimisation, and detailed benchmarking, this work provides practical evidence of how homomorphic encryption is applied in real-world tasks, not just in theory.

In summary, CKKS is accurate enough for analysis, but it still requires considerable speed and memory. The optimisation steps reduced some of these demands, but further progress is needed before these methods can be easily applied in real-world insurance settings.

## 4.9 Program Output

In addition to the tables and visualizations, the project produced raw console logs while running the CKKS regression workflow in Microsoft SEAL. Figure 8 shows an example of this output. The log records each step for the insurance dataset, including setting parameters, measuring encryption times, tracking rotation costs, recovering regression coefficients, and verifying predictions.

The log shows the prediction runtimes for both the unoptimised and optimised designs, confirming a significant difference between them. The unoptimised bulk prediction was much slower because it had to generate the context and keys each time. In contrast, the optimised version reused parameters, which made it run much faster. The output also includes accuracy metrics, such as MAE, RMSE, and MAPE, which are summarized in Tables 4.1 and 4.2.

The figure below helps make things clear by showing the raw program output behind the tables and charts. Earlier sections used processed values to make them easier to read; however, the console log shows exactly how encrypted regression and prediction were carried out.

```
(base) papu@sanjus-MacBook-Air build % ./bin/sealexamples

Running CKKS Health Regression Pipeline...

=== CKKS Encrypted Insurance Regression ===
Enter number of entries to use: 1000
Enter poly_modulus_degree (4096, 8192, or 16384): 16384
Select mode: 1 = regression only, 2 = rotation only, 3 = both: 3

[Info] Detected Columns:
  [age]
  [sex]
  [bmi]
  [children]
  [smoker]
  [region]
  [charges]
[Timing] SEALContext setup: 17 ms
[Timing] Encoding done: 1 ms
[Timing] Encryption done: 6 ms
[Memory] Encrypted x size: 768.11 KB
[Memory] Encrypted y size: 768.11 KB

--- Data Encrypted with CKKS ---
  • Encrypted column 'age' with 1000 values.
  • Encrypted column 'charges' with 1000 values.

--- Rotation Test(for slot-sum)---
[Timing] Vector rotation: 2 ms

--- Homomorphic Regression Computation ---
[Timing] x*x operations: 3 ms
[Timing] x*y operations: 2 ms
[Timing] Sum-slots via rotations: 22 ms
[Timing] Sum-slots via rotations: 21 ms
[Timing] Sum-slots via rotations: 13 ms
[Timing] Sum-slots via rotations: 13 ms

--- Decryption and Final Output ---
[Timing] Decryption + Decoding: 3 ms

[Output] Decrypted Linear Regression Coefficients:
  • Slope (age → charges): 280.0012
  • Intercept: 1983.5098

[Output] Plaintext Linear Regression Coefficients:
  • Slope (age → charges): 280.0012
  • Intercept: 1983.5098
  • MAE: 8756.5376 | RMSE: 11306.1127 | MAPE%: 109.1891

--- CKKS Regression Process Complete. ---

---  Operation-Level Benchmarking ---
 Operation-Level Benchmarking Complete. Results saved to 'operation*.csv'

===Unoptimised Bulk Prediction ===
⏱Unoptimesed Runtime: 12367 ms
[Unoptimised Metrics] MAE: 0.0000 | RMSE: 0.0000 | MAPE%: 0.0000

=== Optimised Bulk Prediction ===
⏱Optimized Runtime: 3520 ms
[Optimised Metrics] MAE: 0.0000 | RMSE: 0.0000 | MAPE%: 0.0000

=== Parameter Variation Benchmarking (poly_modulus_degree = 16384) ===
 Values are saved in benchmark_param_variation.csv
(base) papu@sanjus-MacBook-Air build %
```

Figure 8: Console output from benchmarking

# 5. Discussion

This chapter looks at the findings from Chapter 4 and considers what they mean in relation to the research goals and the work of other studies. It explains the reasons behind the main design choices, evaluates how practical homomorphic encryption (HE) is in real use, and highlights the main contributions of this dissertation. It also points out the study's limits and suggests directions for future research.

## 5.1 Reflection on Progress vs Original Plan

The project largely adhered to the original plan, with a few adjustments made along the way. I decided to focus on the insurance dataset because it offered a realistic case study. The benchmarking framework was updated to measure not only runtime but also memory use, accuracy, and CSV logging, making the results reproducible. I added separate baselines for regression, unoptimised, and optimised prediction to better illustrate where bottlenecks and efficiency gains occurred, which strengthened the analysis. Although I considered testing larger datasets and running GPU experiments, hardware limitations meant I focused on CPU runs with up to 1000 rows. Ultimately, I achieved the main objectives and implemented changes to enhance clarity and reproducibility, as well as address my supervisor's feedback.

## 5.2 Interpretation of Results

As shown in Table 4.1, encrypted errors stayed within ~2% of plaintext. The regression experiments found that CKKS produces results like plaintext regression. The accuracy differences were minor, with both MAE and RMSE within 2% of the plaintext model. This suggests CKKS is a dependable option for encrypted analytics when some approximation is acceptable.

The results also show that using larger encryption parameters comes with a cost. As shown in Table 4.4, multiplication and rotation times increased a lot when the polynomial modulus degree went up. Although accuracy stayed the same, the extra time made computations slower and less practical for large-scale use. Overall, CKKS provides enough accuracy for analytics, but efficiency is still a challenge.

## 5.3 Why Homomorphic Encryption is not yet Practical

The findings from this study reveal why homomorphic encryption is not yet widely adopted in real-world systems.

- First, the performance cost is high. Operations such as encryption, multiplication, and rotation take significantly longer than plaintext, and these delays increase rapidly when larger parameters are used.

- Second, memory usage does not increase predictably as datasets become larger. Smaller datasets seemed stable, but much larger ones could suddenly require significantly more memory, making practical use harder.
- Third, the method relies heavily on selecting the correct settings. To ensure accurate results, you must select the correct modulus and scale, which can be challenging for non-experts.

Due to these limitations, many industries opt for simpler solutions, such as multiparty computation (MPC) or differential privacy. While these may not be as secure, they are easier and faster to use on a large scale. In insurance, encrypted analytics can provide accurate results in theory; however, current efficiency issues render them impractical for real-time tasks such as pricing or evaluating claims.

## 5.4 Contributions of this Project

By using organised baselines, comprehensive benchmarking, and prediction validation, this work expanded on conventional CKKS demonstrations. These characteristics contributed to CKKS's success in insurance analytics as verifiable proof.

- **Prediction Validation:** We compared encrypted predictions directly with actual insurance values and measured accuracy using MAE, RMSE, and MAPE. This approach made sure the results were practical and not just theoretical.
- **Baselines and Optimisation:** The study employed three designs—regression baseline, unoptimised prediction baseline, and optimised prediction design. By separating these, the researchers could show correctness, measure the raw computational cost of encrypted prediction, and then demonstrate how to improve efficiency. Optimisation involved reusing keys and coefficients, as well as restructuring loops, which reduced runtime without altering the results.
- **Benchmarking Framework:** While many examples only show results in the console, this project exported data to CSV files and created visualizations. This made it possible to reproduce the results and systematically analyze runtime, memory use, and how different parameters affect performance.

Together, these contributions provide practical evidence of CKKS performance in insurance analytics, demonstrating how design choices can enhance the efficiency of encrypted computation without compromising accuracy.

All experimental outputs were recorded into CSV files, providing a transparent basis for benchmarking tables and plots.

## 5.5 Comparison with Related Work

The project's findings are consistent with past research on homomorphic encryption. According to Cheon **et al. [3],** the CKKS approach always adds tiny approximation mistakes even if it permits real-number arithmetic. The results here support that behaviour: some variations in MAE and RMSE were unavoidable, but the encrypted regression stayed quite close to the plaintext values.

Other benchmarking studies, such as Costache et al.[6] and the Microsoft SEAL documentation, also noted that multiplication and rotation are the most expensive operations. This was also observed in this project, where rotation and multiplication required significantly more time than addition or encoding.

This project stands out by using two baselines(regression, unoptimised prediction), and one optimised prediction design. Earlier studies typically focused on small-scale demonstrations or single operations. In contrast, this work directly compared optimisation with a basic design, recording and benchmarking the results. The findings demonstrate that careful design choices can enhance efficiency without compromising accuracy.

This study supports earlier research and provides practical evidence on how optimisation affects performance. It does not aim to replace or challenge previous work.

## 5.6 Design Rationale for Code Implementation

Each part of the implementation was included for a specific reason:

- **Regression Baseline**: This step demonstrated that encrypted regression could produce coefficients similar to those obtained from plaintext data, providing a foundation for later comparisons.
- **Unoptimised Prediction**: This version measured the true computational cost of CKKS without any shortcuts, making it easier to spot where the runtime slowed down.
- **Optimised Prediction**: By reusing keys, coefficients, and loops, this version ran faster while still producing the same results.
- **Rotation**: This step aligned vectors for regression, albeit at an additional cost.
- **Prediction Validation (Phase 9)**: Compared encrypted predictions with plaintext values, producing MAE, RMSE, and MAPE to measure accuracy.
- **CSV Export**: This step saved runtime, memory, and accuracy data to files, making it possible to reproduce and visualize the results.

## 5.7 Limitations

This project faced a few important limitations.

- Dataset size: The insurance dataset was used in its entirety; however, for larger datasets, only the first 1000 entries were tested. This kept runtimes reasonable with CKKS. Testing with more data would have provided additional insights, but this was not possible given the time and resources available.
- The study focused only on regression tasks. Models such as classification or logistic regression were not included, which means the results are limited to a specific range of problems.
- All experiments ran on a MacBook Air with an Intel CPU and 8 GB of RAM, without GPU acceleration. This led to longer runtimes and limited scalability. Only SEAL and standard C++ libraries were used, without GPU acceleration. This setup limited both performance and scalability.
- High polynomial degrees, such as 16384, were tested to boost accuracy and ciphertext capacity. This led to significantly higher runtime and memory usage, demonstrating that balancing accuracy and performance remains a challenge.

Future work could address these limits by testing larger datasets, exploring more advanced models, and using GPU-based HE libraries or hardware accelerators to improve scalability.

## 5.8 Future Work

There are several ways future research could expand on this project. For example, since our experiments used only small subsets of the insurance dataset, testing with larger datasets could better show how well the approach scales and how much memory it uses.

Additionally, since all experiments were run on CPUs, exploring GPU acceleration or hardware support for homomorphic encryption could help reduce runtimes and make encrypted analytics more practical.

Third, the parameters in this project were manually set. Using automatic parameter selection tools could make the process easier for users without a technical background and help avoid trial-and-error when choosing scales and modulus sizes.

Fourth, while this study employed regression, future work could explore more complex models, such as classification or decision trees, under CKKS. This would extend the scope of encrypted analytics beyond simple linear relationships. At last, Record hardware-level cycle counts to complement runtime/memory and pinpoint CPU bottlenecks.

## 5.9 Security Considerations

In this project, we performed all computations directly on encrypted insurance data. As a result, no intermediate values were exposed in plaintext during processing. This demonstrates that it is possible to maintain data confidentiality while conducting effective regression analysis.

Two practical limitations were also observed. First, when using smaller polynomial degrees, such as 4096, noise growth became a problem, and some operations failed at higher scales. Second, as with all homomorphic encryption systems, this approach relies on secure handling of the secret key. If the key is exposed, all protection is lost.

These findings demonstrate that CKKS can protect confidentiality; however, its practical application still requires careful parameter choices and secure key management.

# 6. Conclusion

This dissertation examined the practical applications of homomorphic encryption in real-world analytics, with a focus on the CKKS scheme in Microsoft SEAL v4.1.2. To balance privacy and accuracy, regression tasks were run on insurance data. Three approaches were tested: a regression baseline, an unoptimised prediction baseline, and an optimised prediction design. The regression baseline showed that CKKS could closely match plaintext results. The unoptimised baseline revealed that multiplication and rotation were the main bottlenecks. The optimised design improved efficiency by reusing keys and encoders, without sacrificing accuracy.

Benchmarking provided more insight by measuring runtime, memory use, and accuracy at different polynomial degrees (4096, 8192, 16384). The results showed clear trade-offs. Higher degrees improved accuracy and allowed for larger ciphertexts, but also increased runtime and memory usage significantly. Multiplication and rotation were always the most expensive operations. Accuracy stayed within 2% of plaintext results, showing that CKKS is reliable for analytics. Still, issues like noise growth at lower degrees and the need to protect the secret key limit how widely CKKS can be used right now.

The study made three main contributions. First, it tested encrypted regression and prediction validation in more advanced scenarios than basic CKKS demonstrations. Second, it set baselines and improved the design for structured evaluation. Third, it provided real-world benchmarks showing the balance between accuracy and performance. However, even though CKKS maintains accuracy and privacy in insurance analytics, challenges with runtime and scalability remain.

## Final Closing Statement:

This study demonstrates that homomorphic encryption can maintain the security of insurance data while still enabling accurate analysis. However, to apply it in practice, we need to address performance issues by improving optimisation, tuning parameters, and developing more efficient hardware.

# 7. References

[1] R. L. Rivest, L. Adleman, and M. Dertouzos, "On data banks and privacy homomorphisms," Foundations of Secure Computation, pp. 169–177, 1978.

[2] C. Gentry, "Fully homomorphic encryption using ideal lattices," in Proc. 41st ACM Symposium on Theory of Computing (STOC), 2009, pp. 169–178.

[3] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in ASIACRYPT 2017: Advances in Cryptology – ASIACRYPT 2017, pp. 409–437, 2017.

[4] Z. Brakerski and V. Vaikuntanathan, "Efficient fully homomorphic encryption from (standard) LWE," in Proc. IEEE 52nd Annual Symposium on Foundations of Computer Science (FOCS), 2011, pp. 97–106.

[5] Microsoft Research, Microsoft SEAL (release 4.1.2), https://github.com/microsoft/SEAL, 2022.

[6] M. Costache, S. Carpov, N. Gama, D. Jetchev, and M. Walter, "Homomorphic encryption for evaluation of financial data," Proc. Privacy Enhancing Technologies Symposium (PETS), pp. 465–482, 2019.

[7] K. Laine and T. Player, "Simple encrypted arithmetic library – design and implementation," in Financial Cryptography and Data Security (FC), pp. 3–17, 2016.

[8] N. P. Smart and F. Vercauteren, "Fully homomorphic encryption with relatively small key and ciphertext sizes," in Public Key Cryptography – PKC 2010, pp. 420–443.

[9] T. Li, N. Li, W. Qardaji, and J. Cao, "Privacy-preserving health data sharing with functional encryption," IEEE Transactions on Information Forensics and Security, vol. 14, no. 3, pp. 673–688, 2019. [Online]. Available: https://doi.org/10.1109/TIFS.2018.2866411

[10] S. Halevi and V. Shoup, "Algorithms in HElib," in Advances in Cryptology – CRYPTO 2014, pp. 554–571.

[11] T. Graepel, K. Lauter, and M. Naehrig, "ML confidential: Machine learning on encrypted data," in Proc. Information Security and Cryptology (ICISC), pp. 1–21, 2012.

[12] European Union, "General Data Protection Regulation (GDPR)," Regulation (EU) 2016/679, 2016.

[13] A. D. Buescher, C. E. Brodley, and D. D. Johnson, "Privacy-preserving regression modeling via homomorphic encryption," in Proceedings of the 2017 ACM Workshop on Artificial Intelligence and Security (AISec), pp. 17–28, 2017. [Online]. Available: https://doi.org/10.1145/3128572.3140447