School of Engineering and Computer Science
Te Kura Mātai Pūkaha, Pūrorohiko

PO Box 600
Wellington
New Zealand

Tel: +64 4 463 5341
Internet: office@ecs.vuw.ac.nz

Implementation of a Privacy-Preserving Machine Learning Model Using Homomorphic Encryption

Author: Thomas Green
Supervisor: Arman Khouzani
2025

Submitted in partial fulfilment of the requirements for
Bachelor of Engineering with Honours

Abstract

As data privacy and security concerns continue to escalate, particularly in domains handling sensitive information such as healthcare, finance, and cloud services, there is a growing need for methods that protect data during computation. Integration of Fully Homomorphic Encryption (FHE) in business would provide measurable economic benefits by allowing lines of business and third parties to perform big data analytics on encrypted data while maintaining privacy and compliance records. This project investigates the feasibility and practicality of implementing a privacy-preserving machine learning (ML) model using FHE, a cryptographic approach that allows direct computation on encrypted data. The core objective is to develop and evaluate an end-to-end ML workflow in which training and inference are performed entirely on encrypted inputs. This will be achieved using modern open-source FHE and ML libraries. Comparative analysis between encrypted and plaintext models will be conducted to assess performance, accuracy, and computational overhead trade-offs. Additionally, hybrid optimization strategies and a user interface may be explored to demonstrate real-world viability and practicality. The project aims to contribute to the growing field of secure AI by showcasing the viability of FHE-powered, privacy-preserving ML systems in modern computing environments.

1 Introduction

1.1 Motivation

As of 2025, the increasing integration of machine learning into critical sectors has brought significant advancements but also raised challenges and concerns regarding data security and user privacy. This is especially true in fields such as healthcare, finance, and cloud services, where sensitive information is routinely processed using AI models hosted on external or untrusted infrastructure.

This highlights the growing risk of exposing sensitive data during computation, particularly in AI systems where raw inputs must often be decrypted before processing, creating a critical vulnerability window where data is exposed. For example, in January 2025, a misconfigured cloud database at AI firm DeepSeek left over a million sensitive entries, including chat logs, API keys, and internal metadata, publicly accessible, raising major concerns about regulatory compliance and data security [1]. Such incidents are particularly problematic in contexts where legal, compliance, and ethical standards require the safeguarding of personally identifiable information (PII), protected health information (PHI), intellectual property (IP) or confidential corporate data.

Integrating Fully Homomorphic Encryption (FHE) into workflows offers a promising solution to these concerns. It enables arbitrary computation to be performed directly on encrypted data [2], eliminating the need to expose sensitive information during training or inference. Its capability aligns with zero-trust principles and modern privacy requirements, especially in AI workflows that operate across distributed or third-party environments. Since its conceptual inception in 1978 [3] and practical breakthrough in 2009 [2], FHE has faced questions about its practicality and feasibility due to high computational overhead. This project investigates whether FHE can be practically and effectively applied to a standard ML workflow, balancing theoretical security with real-world usability. A prototype will be developed using open-source FHE libraries and public datasets to evaluate performance, accuracy, and resource use in end-to-end encrypted machine learning.

1.2 The Problem

As ML becomes increasingly integrated into business operations and decision-making, concerns around privacy and protection are crucial to address [4]. Modern machine learning workflows face a growing tension between data-driven innovation and the privacy rights of individuals and organizations whose data fuels these systems. Whether deployed as public-facing AI products or internal enterprise tools, effective ML models require access to large volumes of data, much of which may include personal, sensitive or proprietary information. However, how this data is collected, shared, or processed frequently raises legal, ethical, and operational challenges. In sectors such as healthcare, finance, and cloud services, these concerns are heightened by strict regulatory frameworks like GDPR [5] and HIPAA [6] and industry-specific compliance standards.

Beyond compliance, organizations increasingly grapple with the risks of poor data governance. Even internally developed models can introduce exposure if training or inference is performed on plaintext inputs within untrusted infrastructure or third-party environments. Many large-scale models are trained on datasets scraped or aggregated from public and private online and internal sources, often without clear permission or informed consent from data owners [7]. This practice has led to class action lawsuits, regulatory scrutiny, and growing concerns over data ownership and transparency [8]. Individuals typically have no visibility or control over how their data is appropriated for AI development, resulting in a "trust deficit" in AI systems, driven by fears that sensitive data may be misused, mishandled, or exploited without transparency or accountability. As a result, there is an urgent need to re-evaluate how sensitive data is treated within AI pipelines and develop built-in privacy safeguards that align innovation with ethical and legal responsibilities. Organizations must reconsider how sensitive data is handled in ML workflows to maintain regulatory compliance, uphold ethical responsibilities, and reduce the risk of data misuse. There is a pressing need for built-in privacy safeguards that enable computation directly on protected data without requiring decryption, especially in high-risk ML workflows, while supporting meaningful model development and prediction.

1.3 Solution and Deliverables

This project aims to implement a privacy-preserving machine learning workflow using Fully Homomorphic Encryption (FHE), enabling encrypted data to be used directly in training and inference without ever being decrypted. The final solution will be a working prototype demonstrating the feasibility of encrypted ML inference on real-world data, with a comparative evaluation against traditional plaintext models.

1.3.1 Functional

The identified functional requirements (FR) and non-functional requirements (NFR) for the solution and deliverables are as follows:

- FR1: A trained ML model (e.g. logistic regression or decision tree) that accepts encrypted input data and produces encrypted predictions. Training will be conducted on plaintext and encrypted data where appropriate, while inference remains fully encrypted.
- FR2: Implementation of encrypted arithmetic operations, i.e. addition, multiplication, using either open-source FHE libraries, e.g. Concrete-ML, PySEAL, TenSEAL, or a valid custom implementation that demonstrates correct fully homomorphic behaviour.
- FR3: Use of Scikit-learn for initial ML model development; optional use of Torch or Tensorflow for experimenting with deeper or quantized models.
- FR4: Benchmarking of plaintext vs. encrypted models based on metrics, such as; execution time, accuracy, memory usage, and encryption overhead.
- FR5: (Optional) A simple user interface to demonstrate how encrypted inputs are processed, and predictions are returned securely. Intended for demonstration to an audience.
- FR6: (Optional) Exploration of optimizations such as batching, approximate functions, federated training, or reduced circuit depth to improve feasibility and efficiency.

1.3.2 Non-Functional Requirements

- NFR1: End-to-end encryption of inputs, model parameters, and outputs using lattice-based schemes. Adherence to zero-trust principles.
- NFR2: Ensure the system design aligns with data protection standards, such as GDPR.
- NFR3: Output must be interpretable and actionable by the user. If a GUI is implemented, it should prioritize clarity and simplicity.
- NFR4: The prototype should be extensible to support more complex models or larger datasets in future iterations.
- NFR5: Code will be modular and documented to support future modifications or library upgrades.

1.4 Environmental and Sustainability

The final product is designed with sustainability in mind by minimizing computational redundancy and optimizing resource usage. Once trained, the encrypted ML model will operate in a low-power mode by caching encrypted inference outputs where applicable, eliminating the need to rerun complex computations for similar inputs. The system supports efficient batching and lazy evaluation strategies, ensuring only essential operations are performed during runtime. Additionally, the architecture avoids reliance on high-energy cloud computing or specialized hardware, instead functioning effectively on standard local infrastructure. The optional GUI is lightweight and optimized for energy-efficient interaction, contributing to the product's minimal environmental footprint. The deliverable prioritizes energy efficiency, reusability, and low-overhead design, supporting sustainable AI deployment.

2 Background Research

Understanding the intersection of privacy, cryptography, and machine learning is essential to evaluating the feasibility and impact of this project. This section synthesizes key academic and industry research on Fully Homomorphic Encryption (FHE), its applications in secure computation, and current challenges in privacy-preserving machine learning.

2.1 Literature Review

2.1.1 Foundations of Secure Computation - 1978

One of the earliest conceptual breakthroughs in privacy-preserving computation was proposed by Rivest, Adleman, and Dertouzos in their 1978 report "On Data Banks and Privacy Homomorphisms" [3]. In this foundational work, they introduced the term "privacy homomorphism" to describe encryption functions that allow computations to be performed directly on ciphertext, producing encrypted results that, when decrypted, match the outcome of operations as if they had been performed on the plaintext.

This idea emerged from a practical small application use case. The idea was a loan company which uses a commercial time-sharing service to store its records (Figure 1) [3]. The loan company's "data bank" contains sensitive information which should be kept private. On the other hand, suppose that the information protection techniques employed by the time-sharing service are not considered adequate by the loan company; specifically, the systems programmers would have access to the information. In response to this, the loan company decides they wanted to encrypt all of their data kept in the "bank" and to maintain a policy of only decrypting data at the home office, meaning the time-shared computer will never decrypt data.

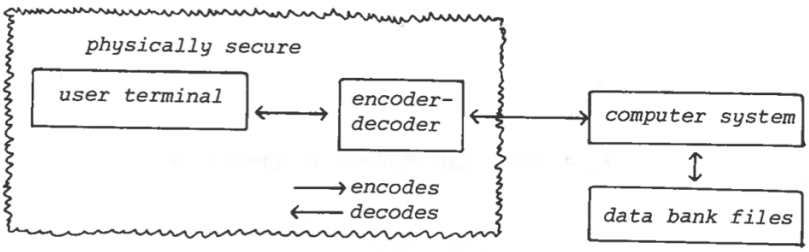

Figure 1: Initial Homomorphic Design [3]

To demonstrate the feasibility of encrypted computation, the authors designed a set of example encryption functions, termed privacy homomorphisms, that allowed basic arithmetic operations to be performed directly on the ciphertext. These were implemented using algebraic systems that preserved operational consistency between plaintext and encrypted data. The authors formalized the mathematical structure of homomorphic encryption using algebraic systems and outlined its potential for secure remote computation while acknowledging inherent limitations, particularly in supporting secure comparisons and defending against chosen-plaintext attacks. Although the specific schemes proposed were later found to be insecure [3], their work laid the theoretical groundwork for decades of subsequent research, including the development of modern FHE.

2.1.2 Fully Homomorphic Encryption Scheme - 2009

The concept of a "Fully Homomorphic Encryption Scheme" [2] was introduced by Craig Gentry in his 2009 PhD thesis, which presented the first construction of an FHE scheme that was provably secure and theoretically sound. A FHE scheme enables arbitrary computations on encrypted data, supporting both addition and multiplication. What makes such a scheme fully homomorphic is the ability to evaluate its own decryption function, achieved through a technique known as bootstrapping. Gentry's model began with developing a somewhat homomorphic "bootstrappable" encryption (SHE) scheme that could perform a bounded number of additions and multiplications on ciphertexts. The innovation came in Gentry's technique of bootstrapping, a recursive process wherein the scheme homomorphically evaluates its decryption circuit to "refresh" ciphertexts and reduce accumulated noise [2]. This allowed the transformation of a SHE into a fully homomorphic one capable of evaluating arbitrary circuits.

The construction was lattice-based, relying on ideal lattices and the hardness of problems such as the Shortest Vector Problem (SVP) [9] and the Ideal Coset Problem, making the scheme resistant to both classical and quantum attacks. Gentry's blueprint also emphasized the importance of keeping the decryption

circuit within the complexity class NC^1 [2], ensuring it could be evaluated homomorphically. While the original construction was computationally impractical due to the overhead introduced by bootstrapping and the depth of the evaluated circuits, it established the foundational paradigm, SHE + bootstrapping = FHE, that modern FHE research continues to build upon. Gentry's work marked a transition from theoretical speculation to practical exploration and remains the cornerstone of all subsequent lattice-based FHE developments. At the core of Gentry's proposal was the notion of evaluating a decryption circuit homomorphically on a ciphertext, thus resetting the noise and enabling further computation. While theoretically well-designed, this process was slow and complex, requiring deep circuit evaluations and large lattice dimensions.

2.1.3 (Leveled) Fully Homomorphic Encryption - 2011

Bootstrapping in FHE is the technique that reduces the noise in a ciphertext by homomorphically evaluating the decryption function on the encrypted data itself [10]. It "refreshes" the ciphertext, allowing more computations to be performed without decryption errors [10]. However, this is what had made the Homomorphic Encryption scheme impractical and infeasible for use in real-world systems due to the bootstrapping process being computationally expensive. Gentry's original scheme required evaluating decryption circuits of depth $O(\lambda^5)$ and performing computations with encrypted secret keys of size $O(\lambda^7)$, leading to an overall complexity of quasi-linear in λ^9 per ciphertext refresh [2].

In this context, a circuit refers to a sequence of arithmetic operations, such as additions and multiplications, performed on encrypted data. Each operation increases the noise within a ciphertext, which is random data added during encryption to preserve security. If the noise grows too large, it prevents correct decryption. Bootstrapping addresses this by "refreshing" the ciphertext, homomorphically evaluating the decryption function to reduce noise and allow further computation.

A key advancement in the path toward practical homomorphic encryption came with Brakerski, Gentry and Vaikuntanathan's 2011 work on "(Leveled) Fully Homomorphic Encryption without Bootstrapping" [11], most commonly known as leveled FHE. This scheme removed the need for expensive bootstrapping operations within bounded-depth computations. Instead of supporting arbitrary-length circuits like earlier FHE schemes, leveled FHE schemes are parameterized by a predefined depth L (number of sequential addition/multiplication operations), allowing encrypted computations on circuits up to L levels without refresh steps [11]. This significantly improves performance and opens new avenues for real-world implementation.

Their scheme is built on the Learning with Errors (LWE) [12] and Ring-LWE [12] assumptions, utilizing modulus switching and relinearisation to manage noise growth without relying on bootstrapping [11]. The technique replaces fully general circuit evaluation with a more practical but bounded-depth model, making it computationally feasible to run meaningful machine learning inferences on encrypted data. Limiting extensive bootstrapping also results in better latency and throughput in settings where the circuit depth is known in advance, a common case for many ML applications.

This leveled approach laid the groundwork for most modern FHE libraries and influenced a wide range of practical applications by making FHE far more efficient and reproducible in constrained environments. A privacy-preserving ML model using leveled FHE is possible as long as the ML model's computations don't exceed the bounded-depth computations of the FHE model.

2.1.3 Federated Learning using FHE

Federated Learning (FL) with Fully Homomorphic Encryption [13] is an approach that enables multiple parties to collaboratively train a shared model without exchanging their raw data. FL addresses privacy concerns by ensuring data never leaves the local device or environment, making it particularly useful in regulated sectors such as healthcare and finance, where data sharing is constrained by compliance frameworks [13]. However, standard FL implementations are vulnerable to inference attacks during model update aggregation, as model gradients can unintentionally leak private information [13]. To address this, IBM has proposed integrating FHE into FL pipelines.

In their implementation [13], model updates are encrypted using a shared key before being sent to an untrusted aggregator, which performs computations directly on ciphertexts and returns an encrypted global model (FR1) [13]. This ensures that individual updates remain private and inaccessible even to the orchestrating server. Notably, because training still occurs on plaintext locally, models can retain their architectural complexity, including arbitrary activation functions, while maintaining end-to-end privacy through FHE at the aggregation step. IBM's work demonstrates that such systems are not only secure but also performant, as their HELayers library leverages vectorized FHE operations (SIMD) to ensure scalability and practical runtime performance (FR6) [13]. Given its balance between privacy and efficiency, this approach presents a promising and computationally feasible design for real-world, reproducible privacy-preserving AI systems.

2.1.4 Related Work

Functional Requirements	PySEAL	TenSEAL	Concrete-ML
FR1	Y	Y	Y
FR2	Y	Y	Y
FR3	N	Y	Y
FR4	Y	Y	Y
FR5	N	N	N
FR6	Y	Y	Y

Table 1: Comparison of Functional Requirements Across Tools

PySEAL is a Python wrapper around Microsoft's Simple Encrypted Arithmetic Library (SEAL) written in C/C++, introduced in March 2018, and is one of the most widely used homomorphic encryption frameworks. SEAL supports three major encryption schemes: BFV and BGV schemes (FR2), which allow modular arithmetic to be performed on encrypted integers [14] and CKKS scheme (FR2), which enables additions and multiplications on encrypted real or complex numbers [14], both of which are somewhat homomorphic schemes that become fully homomorphic through bootstrapping. However, PySEAL and SEAL, by default, operate in a leveled FHE mode, meaning computations are limited to a fixed depth of circuit before decryption is required, thus avoiding the high cost of bootstrapping in most practical scenarios. This makes them especially suitable for inference tasks (FR1) where the depth of computation can be bounded in advance. In PySEAL, encrypted values are stored as polynomials over rings, and computations directly manipulate these encrypted polynomials [15]. The system manages noise growth internally, and the lack of bootstrapping means a carefully chosen encryption parameter set is essential (FR6) to support the desired computation depth [15]. Although PySEAL exposes lower-level functionality compared to other libraries, it enables full customization of FHE workflows and is ideal for benchmarking FHE-based ML operations under controlled conditions, offering both clarity and reproducibility in a research setting (FR4).

TenSEAL, released in April 2021, is designed for machine learning applications that require encrypted inference (FR1) while maintaining ease of integration with popular frameworks like PyTorch (FR3). TenSEAL uses the CKKS (FR2) scheme and provides two primary encrypted tensor types: CKKSVector, which encrypts a vector of real values into a single ciphertext [16], and CKKSTensor, which supports N-dimensional real-valued tensors by organizing them into (N-1)-dimensional tensors of ciphertexts [16]. This enables efficient SIMD-style operations supporting operations over encrypted floating-point numbers. Reliance on CKKS makes it a leveled FHE approach [16], where users must set the multiplicative depth (or levels) during context creation, and once exhausted, no further operations can be performed without re-encryption or bootstrapping (which TenSEAL does not implement by default). This makes TenSEAL especially suitable for ML inference workflows where the number of operations is known and limited (FR4). Its support for encrypted tensors, vector-matrix multiplication, and dot products [17] enables private inference over real-valued vectors with minimal changes. However, the reliance on polynomial approximations for non-linear activations and manual noise management limits its use in deeper or more complex models (FR6) [18]. Despite these constraints, TenSEAL enables practical encrypted inference in low-latency privacy scenarios, offering a compelling mix of cryptographic and engineering accessibility.

Concrete-ML, introduced in July 2022, is a library developed by Zama [19] that leverages TFHE (Fully Homomorphic Encryption + Torus, a structure for encoding real numbers modulo 1) [19] to compile and run full machine learning workflows directly on encrypted data (FR1). Unlike leveled approaches, which rely on CKKS and BGV, TFHE supports programmable bootstrapping, which allows it to evaluate arbitrary functions, including non-linear ones, without switching to plaintext (Figure 2) (FR2) [19].

	BGV	CKKS	TFHE-LIB	TFHE-CONCRETE
Operations	Leveled	Leveled	Bootstrapped	Leveled + Bootstrapped
Non-linear functions	Approximate	Approximate	Exact	Exact or Approximate
Data Types	Integers	Reals	Boolean	Boolean + Integers

Figure 2: Leveled and Bootstrap Approach Cases [19]

Concrete-ML implements a variant of TFHE that supports both leveled and fast bootstrapped operations, as well as approximate/exact evaluation of arbitrary functions [19], where computations are structured in a way to minimize depth (FR6) while preserving correctness. It performs model quantization to convert real-valued models into integer-based equivalents, then compiles them into circuits executable under TFHE constraints [19]. This workflow includes both quantization-aware training and post-training quantization (FR3), depending on the model type, and culminates in FHE circuits that can be deployed in client-server settings [19]. Clients encrypt inputs locally, send them to a server for encrypted evaluation, and then decrypt the results, ensuring full data privacy throughout. While Concrete-ML imposes constraints on model complexity and precision, its transparent compilation stack and automated cryptographic handling make it one of the most production-ready FHE libraries available for practical ML deployment (FR4).

2.2 Tools and Methodology

2.2.1 Languages

For development, there were two primary languages considered: Python and C/C++. The developer has prior experience with both, so the decision was made based on suitability for integrating ML with FHE workflows. Python was ultimately chosen due to its strong ecosystem for both AI and FHE development. Python offers actively maintained libraries such as scikit-learn, PyTorch, and TensorFlow for ML, alongside purpose-built FHE libraries like TenSEAL, Concrete-ML, and PySEAL. These libraries are not only more accessible via Python APIs but also backed by strong documentation and active community support. Python also enables rapid prototyping and testing, which is essential for iterative design in a research-orientated project, especially since the project's goal is to evaluate feasibility. Its dynamic typing and high-level abstractions allow quicker experimentation with model architectures, encryption parameters, and benchmarking setups compared to lower-level libraries.

C/C++ offers better raw performance and is used as the underlying implementation in most FHE libraries; however, it was deemed less suitable for the project's development phase. FHE pipelines in C/C++ require significantly more boilerplate and cryptographic configuration, making experimentation slower and more error-prone. Additionally, many Python-based libraries act as wrappers over performant C/C++ code.

2.2.2 Libraries

To evaluate the feasibility of deploying an FHE-based ML model in a real-world setting, a number of FHE and ML libraries are being reviewed and used. The goal is to assess how well these tools support encrypted ML inference and whether they are mature enough for reproducible and practical deployment.

Scikit-learn is a lightweight and well-documented Python library for classical ML models. It is ideal for prototyping ML workflows and understanding model performance in plaintext. While it lacks built-in

support for encrypted data, it serves as the baseline framework for model development and benchmarking. Many FHE libraries use models initially trained in scikit-learn, converting them to FHE formats.

TensorFlow and PyTorch are deep learning frameworks that support more complex ML architectures, including neural networks and quantization-aware training. TensorFlow provides model quantization and export tools, while PyTorch offers tight integration with TenSEAL. While both are more complex than scikit-learn, they become necessary when experimenting with advanced FHE-compatible models, particularly in scenarios requiring quantized inference or encrypted tensor operations. Their role is exploratory; if the base ML models are insufficient, these frameworks support deeper or more optimized models for homomorphic compilation.

Concrete-ML is the most production-focused FHE-ML library reviewed. Built on the TFHE scheme, it supports both leveled and programmable bootstrapped operations, enabling the encrypted evaluation of arbitrary functions, including non-linear activations. Concrete-ML includes a compiler that transforms trained scikit-learn models into FHE circuits. Its automated quantization, deployment-ready pipeline, and client-server architecture make it a strong candidate for real-world applications. However, its support is mostly limited to classical ML models and quantized approximations, so it may not generalize well to deep learning without trade-offs in precision or complexity.

TenSEAL wraps Microsoft SEAL, and targets encrypted inference using the CKKS scheme. It integrates with PyTorch and supports SIMD-style encrypted tensor operations, such as dot products and matrix multiplications. TenSEAL is highly suitable for real-valued encrypted inference where the model structure is known and bounded in depth. It avoids bootstrapping, relying instead on leveled FHE. This makes it ideal for secure, low-latency inference applications, where data privacy is paramount, though it requires manual tuning of noise budgets and does not support encrypted training.

PySEAL is a low-level Python wrapper over Microsoft SEAL. It exposes the internals of the SEAL library for maximum flexibility in cryptographic configuration. PySEAL supports BFV and CKKS schemes, and while it lacks high-level tensor abstractions, it allows full control over encryption parameters and noise growth. This makes it highly suitable for benchmarking and cryptographic evaluation rather than rapid prototyping or deployment. It is especially useful for validating how different FHE schemes perform under real-world workloads and comparing the precision, ciphertext size, and runtime overhead trade-offs.

2.2.2 Methodology

The project adopts an Agile rapid prototyping methodology approach, organized into iterative development cycles. Each cycle includes initial implementation, testing, benchmarking, and refinement based on observed results. This approach enables quick experimentation with different combinations of ML models and FHE libraries, allowing the project to remain agile while evaluating the feasibility of encrypted inference in real-world scenarios. The methodology suits an individual research project where flexibility, speed and continuous learning are prioritized.

Project management is conducted through GitLab, where features are broken into granular issues linked to specific branches and merge requests. Each feature is developed independently and undergoes a self-review before merging into the main branch to ensure clarity and accountability. Experiments, benchmarks, and encrypted inference pipelines are implemented and documented using Jupyter Notebooks. These notebooks act as documents that combine code, outputs, and commentary, supporting transparency, reproducibility and iterative evaluation. This tooling supports the structured exploration of trade-offs between accuracy, performance, and security in different FHE-ML configurations. This will then be converted into a full Python program, serving as the final model and submitted prototype.

3 Development Progress

3.1 Design

The system is designed to implement end-to-end privacy-preserving ML using FHE. This architecture consists of three primary components: (1) an FHE-based encrypted model pipeline, (2) an encryption and

inference module, and (3) a benchmarking and evaluation layer. All components are initially prototyped using Jupyter Notebooks for rapid experimentation and will be consolidated into a standalone Python application upon completion.

Encrypted Model Pipeline (FR1, FR3, FR6) - unlike conventional machine learning workflows that train on plaintext data and later apply encryption, this project utilizes encrypted inputs directly in the model inference pipeline. This design supports privacy even during computation. Models are developed using libraries like TenSEAL, PySEAL, and scikit-learn, which support encrypted arithmetic options and are compatible with homomorphic evaluation. Model selection prioritizes FHE compatibility (e.g. support for dot products, quantized layers) rather than only plaintext performance/accuracy.

Encryption and Inference Module (FR2, NFR1, NFR2) – handles the core cryptographic functionality: key generation, encryption, homomorphic evaluation, and decryption. Concrete-ML supports TFHE-based evaluation and programmable bootstrapping, while TenSEAL supports CKKS-style SIMD vector operations. PySEAL will also been evaluated for its low-level encryption configuration. Libraries are tested modularly for encrypted arithmetic operations ensuring correctness and performance.

Benchmarking and Logging Layer (FR4, NFR4, NFR5) – compare encrypted models with their plaintext equivalents. Benchmarking scripts measure runtime, memory usage, encryption overhead, and accuracy. This evaluation framework allows iterative testing of multiple configurations to find optimal trade-offs between privacy and performance. Findings from this layer inform design decisions for encryption parameters, model architecture, and library selection.

Optional GUI Considerations (FR5, NFR3) – are being scoped using libraries as potential platforms for visually demonstrating how encrypted data is processed securely through the system. This is not core to the prototype but may be implemented to (1) improve accessibility and understanding for audiences who may not understand the low-level functionality of an ML + FHE implementation and (2) demonstrate if the implementation is viable for practical real-world use.

3.2 Implementation

Implementation to date has focused on developing a functional prototype that meets the foundational requirements of privacy-preserving inference. Development has progressed in Jupyter Notebooks, with several functional requirements already addressed.

Encrypted Model Development (FR1, FR3) – The development of scikit-learn ML models, including logistic regression and linear regression, has been successfully implemented and integrated with encrypted inputs using Concrete-ML. The model accepts ciphertext vectors as input and returns encrypted predictions. The TenSEAL library has also been tested with similar models to access broader model compatibility. Scikit-learn has been used for initial model validation for encrypted and plaintext data to establish a benchmark for encrypted accuracy comparisons.

Homomorphic Arithmetic Operations (FR2) – Homomorphic operations, including encrypted addition and multiplication, have been validated across supported libraries. Concrete-ML's TFHE implementation supports programmable bootstrapping, enabling non-linear operations like activation functions. TenSEAL has been used for testing CKKS-based vectorized operations. The correctness of encrypted arithmetic is verified by comparing decrypted outputs with known plaintext equivalents.

Benchmarking and Performance (FR4, NFR5) – Custom benchmarking code captures performance metrics for plaintext and encrypted inference. Metrics include execution time, memory usage, classification and regression ML evaluation metrics, and prediction accuracy. These benchmarks are used to fine-tune encryption parameters and model configurations. The goal is to demonstrate feasibility and justify using FHE for practical ML use cases.

Security and Privacy Alignment (NFR1, NFR2) – All processing is conducted on scikit-learn datasets generation module, generating data for use in lattice-based schemes (CKKS, TFHE) in alignment with end-to-end encryption goals. In the end program, data will not exist in plaintext format beyond its original input

location, ensuring compatibility with GDPR and other regulations in scope. Zero-trust principles are maintained throughout the prototype.

Extensibility and Documentation (NFR4, NFR5) – Code is modularized by functionality (e.g. encryption routines, model logic, benchmarking) to support future upgrades or integration of new libraries. Documentation is included to assist with onboarding and future expansion. This ensures the system can scale to more complex models or datasets in later phases.

Optional UI (FR5, NFR3) – A basic layout is under consideration for final demonstration and will only be progressed if the final model is feasible and evaluated for real-world use. This UI would accept encrypted input from users and display decrypted prediction outputs. While still under development and optional for project requirements, the intent is to simplify explanation and demonstration for audiences. The application will be basic, possibly as a Streamlit app or Flask web app, and will show a basic GUI interface that will give an understanding of real-world practicality in FHE + ML cases.

The current implementation supports creating, testing, and benchmarking end-to-end ML models. The prototype demonstrates FHE inference and provides a foundation for further research and integration.

4 Future Plan

4.1 Current Deliverables

The project has made significant progress toward building a privacy-preserving machine learning pipeline using FHE. To date, experimentation has been conducted within Jupyter Notebooks, where FHE libraries Concrete-ML and TenSEAL have been tested against different scikit-learn ML models. These experiments focus on enabling encrypted inference directly on ciphertext inputs rather than converting plaintext models. The emphasis has been on evaluating the feasibility and efficiency of homomorphic operations using basic ML models.

We have successfully implemented encrypted addition and multiplication routines using multiple libraries, meeting FR2. Furthermore, a basic encrypted inference pipeline has been established for classical regression and classification ML models, including logistic regression and decision tree, using encrypted inputs, aligning with FR1 and FR3. Benchmarking efforts have commenced, comparing plaintext vs encrypted inference across key metrics, including runtime, ML evaluations, and accuracy, which meet part of FR4. Non-functional requirements, such as modular design (NFR5), interpretability (NFR3), and privacy-by-design (NFR1), have also been actively considered during implementation. The FR5 and FR6 are not yet implemented, as they are reserved for the final development phases.

Challenges encountered include managing the high computational cost of encrypted inference and dealing with limitations in model complexity due to FHE constraints. However, these issues have been met with critical iteration and adaptive experimentation, building on research to make FHE implementations practical. For example, by modifying quantization/bootstrapping levels, adjusting parameter sizes, and simplifying model structure, the pipeline was adapted to operate within feasible performance bounds and will be extended as implementation progresses. The decision to delay GUI implementation and advanced optimization strategies also reflects thoughtful prioritization of core functionality before optional features.

4.2 Planning

With the current implementation, the focus from here will be on continuing to investigate ML and FHE libraries to finalize what the final model libraries will be (Figure 3). From here, the development will be focused on finalizing implementation and meeting all functional and non-functional requirements. We will continue this approach towards the next steps of the project. As we progress, we will focus on evaluating the current functional requirements we are working on and investigating, specifically FR1, FR2, and FR3. We must evaluate these models correctly using ML model evaluation techniques and encryption/decryption evaluation methodologies. Planning is divided into four development sprints. The first sprint is scheduled after the trimester one exam period to ensure consistent progress and allow time for exam study.

The **first sprint** is finalizing library selection and system integration between Jun. 20 – Jul. 10. This will involve confirming and finalizing FHE and ML library selection for production use. Begin transitioning working components from Jupyter Notebooks to Python modules. Completing the implementation of encrypted inference modules (FR1, FR2, FR3). Establishing consistent encryption/decryption APIs and modular architecture (NFR5).

The **second sprint** will involve benchmarking and evaluation between Jul. 11 – Jul. 31. This will include building an automated benchmarking pipeline to compare and evaluate models (FR4). Log metrics include runtime, memory use, accuracy, and ciphertext size. Analyse performance trade-offs and optimize parameter sets. I will also begin to compile results and insights for report writing (NFR4).

The **third sprint** will involve implementing an optional UI and feature expansion between Aug. 01 – Aug. 21. This will include designing and implementing a prototype with a basic GUI for demonstration (FR5). Experimenting with batching, low-precision models, or federated training (FR6). Validating results under constrained computing environments and conducting internal reviews or current outputs for feedback. The starting date for this sprint is flexible as progression with optional requirements will only begin once evaluation and implementation is complete.

The **fourth sprint** will involve system finalization, which will occur between Sep. 01 – Sep. 19. The tool is scheduled to be finalized by the end of Week 9 in Trimester 2. To support this, over two weeks have been allocated before the due date to complete system integration, finalize the encrypted ML workflow, conduct full evaluation and testing, resolve any remaining technical debt, and prepare the implementation for submission. Evaluation activities will be incorporated into the final sprint to ensure adequate time for refinement and validation before delivery.

If performance issues persist, fallback strategies include using lower-depth models, reducing ciphertext precision, or excluding GUI deployment from the final demonstration to prioritize encrypted inference functionality. Non-technical dependencies, such as changes in library APIs or discontinued support, have been considered across libraries being used. In the long term, the architecture is designed to be scalable and adaptable for future use cases. With further optimization and FHE bootstrapping support, the system could evolve into a practical privacy-preserving ML service deployable across untrusted environments.

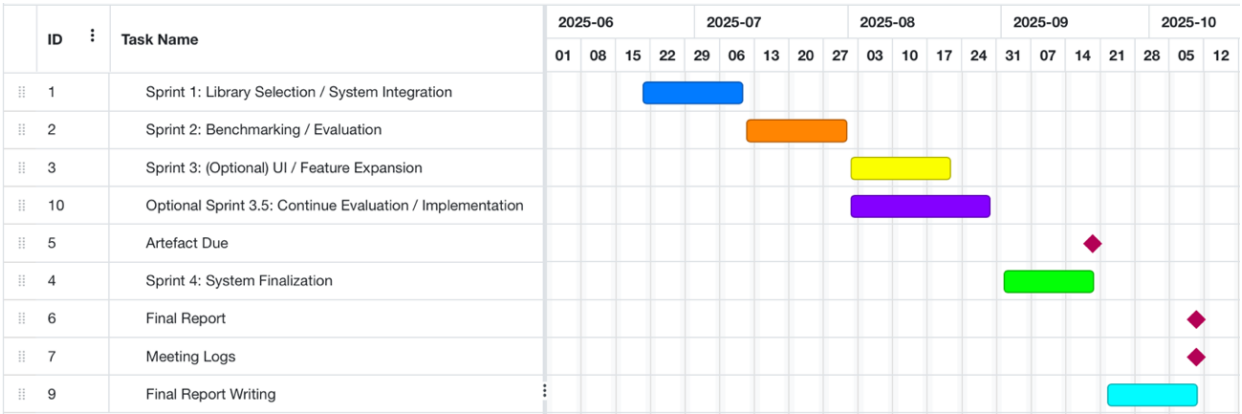

Figure 3: Gantt Chart of Future Project Timeline

References

[1] "DeepSeek Cyber Attack Timeline," Cm-alliance.com, 2025. https://www.cm-alliance.com/deepseek-cyber-attack-timeline (accessed May 30, 2025).

[2] C. Gentry, "A FULLY HOMOMORPHIC ENCRYPTION SCHEME," 2009. Available: https://crypto.stanford.edu/craig/craig-thesis.pdf

[3] R. Rivest, L. Adleman, and M. Dertouzos, "ON DATA BANKS AND PRIVACY HOMOMORPHISMS." Available: https://luca-giuzzi.unibs.it/corsi/Support/papers-cryptography/RAD78.pdf

[4] E. Etheridge, "Cyber security measures: Secure your business with digital signatures," Dataguard.com, Jun. 17, 2024. https://www.dataguard.com/blog/what-is-a-digital-signature-and-how-it-can-secure-your-business/

[5] GDPR, "General Data Protection Regulation (GDPR)," General Data Protection Regulation (GDPR), 2018. https://gdpr-info.eu

[6] CDC, "Health Insurance Portability and Accountability Act of 1996 (HIPAA)," Public Health Law, Sep. 10, 2024. https://www.cdc.gov/phlp/php/resources/health-insurance-portability-and-accountability-act-of-1996-hipaa.html

[7] M. Fazlioglu, "International Association of Privacy Professionals," iapp.org, Nov. 08, 2023. https://iapp.org/news/a/training-ai-on-personal-data-scraped-from-the-web

[8] J. Bock et al., "AI and intellectual property rights," Dentons, Jan. 28, 2025. Available: https://www.dentons.com/en/insights/articles/2025/january/28/ai-and-intellectual-property-rights

[9] S. Khot, "Hardness of approximating the shortest vector problem in lattices," Journal of the ACM, vol. 52, no. 5, pp. 789–808, Sep. 2005, doi: https://doi.org/10.1145/1089023.1089027.

[10] Y. Polyakov, and A. Al. Badawi, "Bootstrapping in Fully Homomorphic Encryption (FHE)," Duality Technologies, Jan. 19, 2023. https://dualitytech.com/blog/bootstrapping-in-fully-homomorphic-encryption-fhe/ (accessed May 30, 2025).

[11] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) Fully Homomorphic Encryption without Bootstrapping," ACM Transactions on Computation Theory, vol. 6, no. 3, pp. 1–36, Jul. 2014, doi: https://doi.org/10.1145/2633600.

[12] V. Lyubashevsky, C. Peikert, and O. Regev, "On Ideal Lattices and Learning with Errors over Rings," Journal of the ACM, vol. 60, no. 6, pp. 1–35, Nov. 2013, doi: https://doi.org/10.1145/2535925.

[13] N. Baracaldo, and H. Shaul, "Federated Learning Meets Homomorphic Encryption," IBM Research Blog, Feb. 09, 2021. https://research.ibm.com/blog/federated-learning-homomorphic-encryption

[14] "Microsoft SEAL," GitHub, Nov. 27, 2022. https://github.com/Microsoft/SEAL

[15] S. Rogers, "PySEAL: Homomorphic encryption in a user-friendly Python package," Medium, May 23, 2018. https://medium.com/bioquest/pyseal-homomorphic-encryption-in-a-user-friendly-python-package-51dd6cb0411c (accessed May 30, 2025).

[16] A. Benaissá et al., "TENSEAL: A LIBRARY FOR ENCRYPTED TENSOR OP- ERATIONS USING HOMOMORPHIC ENCRYPTION." Available: https://dp-ml.github.io/2021-workshop-ICLR/files/18.pdf

[17] "TenSEAL," GitHub, Jul. 24, 2023. https://github.com/OpenMined/TenSEAL

[18] M. Nocker et al., "HE-MAN - Homomorphically Encrypted Machine learning with oNnx models." Available: https://arxiv.org/pdf/2302.08260

[19] "Introducing the Concrete Framework," Zama.ai, 2025. https://www.zama.ai/post/introducing-the-concrete-framework (accessed May 30, 2025).