University of the Philippines Manila
College of Arts and Sciences
Department of Physical Sciences and Mathematics

Fully Homomorphic Encryption-based Machine Learning for Secure Multi-class Tumor Classification

A special problem in partial fulfillment
of the requirements for the degree of
Bachelor of Science in Computer Science
Submitted by:

Gwyneth Rose C. Rosario

June 2023

Permission is given for the following people to have access to this SP:

Available to the general public	Yes
Available only after consultation with author/SP adviser	No
Available only to those bound by confidentiality agreement	No

ACCEPTANCE SHEET

The Special Problem entitled "Fully Homomorphic Encryption-based Machine Learning for Secure Multi-class Tumor Classification" prepared and submitted by Gwyneth Rose C. Rosario in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science has been examined and is recommended for acceptance.

Richard Bryann L. Chua, M.Sc.
Adviser

EXAMINERS:

	Approved	**Disapproved**
1. Avegail D. Carpio, M.Sc.	_____	_____
2. Perlita E. Gasmen, M.Sc. (*cand.*)	_____	_____
3. Ma. Sheila A. Magboo, Ph.D. (*cand.*)	_____	_____
4. Vincent Peter C. Magboo, M.D.	_____	_____
5. Marbert John C. Marasigan, M.Sc. (*cand.*)	_____	_____
6. Geoffrey A. Solano, Ph.D.	_____	_____

Accepted and approved as partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science.

Vio Jianu C. Mojica, M.Sc.
Unit Head
Mathematical and Computing Sciences Unit
Department of Physical Sciences and Mathematics

Marie Josephine M. De Luna, Ph.D.
Chair
Department of Physical Sciences and Mathematics

Maria Constancia O. Carrillo, Ph.D.
Dean
College of Arts and Sciences

Abstract

ML outsourcing is one approach to building AI solutions but involves sharing a client's data with external parties and thus, poses risks on data privacy. Genomic data holds properties that distinguish it from traditional medical data and make it more sensitive with regards to data confidentiality. As patients' information is at risk with ML outsourcing, there is a need to build a solution that would address this problem. That is, a system that would allow for ML outsourcing in the medical field that would ensure data confidentiality. This paper proposes fully homomorphic encryption-based machine learning to achieve a secure multi-class tumor classifier using ConcreteML.
Keywords: Fully Homomorphic Encryption, Tumor Classification, Machine Learning, Privacy, Security, Genomic Data

Contents

List of Figures

List of Tables

I. Introduction

A. Background of the Study

A tumor is a solid mass of abnormal tissue formed in the body when cells divide and grow excessively. It can affect different organs, bones, tissues, and other parts of the human body. Tumors can either be benign or malignant. Benign tumors are non-cancerous and aren't much life-threatening while malignant tumors are cancerous and can spread into the body. Furthermore, tumors may be present in people, regardless of their ages. While not all tumors are cancerous, they can still affect the human body by the pressure they put on organs, bones, tissues, etc. Treatments provided on a tumor also depend on factors such as its type, subtype, and location [2]. In line with this, tumor classification is essential in understanding tumor cells which in return may help in providing new solutions to diagnosis and treatment methods for cancer patients [3].

Diagnosis of a certain disease plays a vital role in the medical system. Not only does it give awareness about the condition of the patient, it is also helpful in finding out suggested care and treatment choices for a patient. Medical diagnosis serves as a guide for medical professionals in their decision making processes. While this process has significantly improved over time with technology, diagnostic errors may occur from misjudgments or misinterpretations and other factors. Such errors may be minimized with machine learning (ML), one of the fastest-growing technical disciplines in the present. Through using this technique in analytics and data science, medical professionals can be provided with more knowledge about a disease to give better care for patients and at the same time minimize the errors [4].

In the area of clinical genomics, pathogenicity - which refers to the probability of a genetic variant being disease-causing - is known to be one of its prime concepts. While there are specific established guidelines for pathogenicity, laboratories

still stumble upon disagreements on classifying a variant's pathogenicity status due to differences in assertions. With machine learning applied to clinical genomics, performance metrics related to prediction quality can be optimized. ML algorithms are also said to be well suited for problems like improving and reframing pathogenicity concepts that involve complex and huge amounts of data. For example, supervised ML approaches improved support vector machine classification of deleterious variants as well as scoring pathogenicity in hypertrophic cardiomyopathy. Thus, the practice of ML is acknowledged vital towards the expanding domain of clinical genomics research [5].

Unlike the traditional symptom-driven approach in medicine, precision medicine addresses the problem that patients may react differently to the same kind of medication. It paved the way for improving healthcare with developing specific treatments based on clinical data associated with a patient and his genomic data. Through clinical genomics, gene-disease relationships are studied and can reveal abnormalities and patient susceptibility depending on the disease and the patient himself. By analyzing such information, precision medicine improves with developing more specific treatment options as well as potential medications for rare diseases. Currently, the integration of different approaches in precision medicine is being considered for further development. This enables a more in-depth understanding of a medical case and can lead to a more accurate and precise treatment for a particular condition [6]. An example of this is the integration of clinical genomics and artificial intelligence (AI), specifically machine learning.

Through the years, countless studies have already been made with regards to the application of ML in different domains. Since machine learning entails data processing, protection of sensitive data especially during analysis must be ensured to avoid information leakage. However, standard practices (e.g. anonymization) aren't enough due to lack of guidance on how to use these available methods as

well as differences in their use cases. For example, k-anonymity is deemed suitable for data publishing while differential privacy is recommended for interactive queries. Both methods also suffer from drawbacks. Inference attacks and homogeneity attacks are still possible with k-anonymity and high dimensionality caused by extra quasi identifiers may lead to increased utility loss. Differential privacy has also received criticisms due to the large information loss that comes with its implementation [7, 8, 9].

In the case of using ML in healthcare, the confidentiality of patients' information is put at high risk. While some treat genomic data similar to traditional health data (those found in medical records), it has properties that distinguish it namely: health/behavior, static, unique, mystique, value, and kinship [10]. With DNA, diagnosing conditions is possible because it has information on an individual's health and behavior. Tests conducted using such data are capable of determining which conditions are likely present in a person. Furthermore, this information is static and has long-lasting value. Genomic data of an individual has little changes over time. Thus, the information it holds can be used for long periods of time unlike traditional medical data such as blood pressure and glucose level whose value tends to decline with time. Genomic data also reveals family relationships since part of it is common with an individual's blood relatives. At the same time, it is unique to a person, which enables its use in forensics. With these properties, it is important to address the increasing concerns in its privacy that goes in parallel with its increasing availability for genomic research.

B. Statement of the Problem

With the wide range and rapid growth of machine learning applications, ML became one of the most promising and common commercial approaches in building AI solutions. ML outsourcing focuses on building ML models that satisfies a client's

specifications or requirements and allows for reduced time and cost [11]. At the same time, this approach does not ensure data confidentiality.

ML outsourcing involves sharing of a company or a client's data with third parties and is a potential risk due to data privacy regulations. When data breach happens, the client may face penalties such as fines and lose credibility due to reputation damage. For patient health information, numerous laws are present in different countries to address data breach. In the US, the Department of Health and Human Services (HHS) issued the Health Insurance Portability and Accountability Act of 1996 (HIPAA), a law that requires creation of national standards that would protect sensitive health information of patients from unauthorized disclosure. It entails a privacy rule with established standards of an individual's rights to understand and control how his health information is used [12]. Australia also implemented a similar law, Privacy Act of 1988, that covers privacy for health service providers (including those who hold health information) and provides protections on handling of health information [13]. The General Data Protection Regulation (GDPR) is a healthcare privacy legislation in the European Union that considers health-related data as a special category of data and provides definition of such data in relation to data protection purposes [14]. In the Philippines, Republic Act No. 10173 also known as Data Privacy Act of 2012 protects an individual's personal information in information and communications systems both in the government and the private sector [15]. This covers implementation of Joint Administrative Order No. 2016-0002, a health privacy code that provides guidelines on health information exchange in the country [16].

In the case of medical practice, patients' information are subjected to risk of data privacy when there is ML outsourcing. Hence, the need for a solution for preserving the privacy of genetic data when it is used in tumor classification using ML techniques.

Homomorphic encryption (HE) is an encryption technique that allows computations to be performed over encrypted data without the need to decrypt them first

[17]. The application of HE in ML is proposed as a solution so that confidential data could be shared with external parties and clients could access computational services securely without the problem of violating data privacy regulations.

C. Objectives of the Study

The study aims to build a system that implements FHE-based logistic regression for multi-class tumor classification using genomic data.

The proposed system had two entities: a client and a server. A medical institution, such as a hospital, that holds genomic information of patients and would outsource ML computation for tumor classification is represented as a client while an ML service provider is represented as the server. The following functionalities were implemented in the proposed system:

1. Allow the medical institution (client) to
    (a) Encrypt genomic data of a patient;
    (b) Send encrypted genomic data to the ML service provider for tumor classification; and
    (c) Receive and decrypt the classification result returned by the ML service provider.
2. Allow the ML service provider (server) to
    (a) Accept encrypted genomic data from the medical institution;
    (b) Perform FHE-based multi-class tumor classification on the encrypted genomic data; and
    (c) Send the encrypted result of FHE inference back to the medical institution.

D. Significance of the Project

Having a secure multi-class tumor classification system is beneficial for both medical professionals and patients. The proposed solution could help medical professionals in their decision making choices with regards to care and treatment that they would suggest to patients. In particular, the use of genomic data for tumor classification would provide development to treatments, such as drugs and therapies, that are tailored to a specific type of tumor and could lead to advancements in precision medicine [18]. In return, this study also aids to improvements in the newly rising medical specialty of genomic medicine, which is linked with precision medicine [19]. Additionally, the integration of HE on the system would ensure confidentiality and privacy preservation of patients' information.

E. Scope and Limitations

1. The study used brain cancer gene expression data from CuMiDa, a curated microarray database for benchmarking and testing of ML approaches in cancer research. Thus, no data collection was conducted.

2. The study used genomic data for the training and testing of the model. Other types of data such as CT Scans and MRIs were not used in the implementation.

3. The study used an existing FHE ML library, specifically ConcreteML, in implementing the proposed system.

4. The study did not cover determination of treatment based on the type of tumor that was classified.

F. Assumptions

1. The client will only input encrypted data to the server.

2. The client will not use other types of data other than genomic data.

II. Review of Related Literature

Machine learning application in healthcare has always been an active topic in the research industry for years up until the present. It is known to have many clinical applications, such as high-risk patient identification, early detection of diseases, diagnosing conditions based on specific data, etc. [20]. Its application in the medical domain helps in suggesting improved healthcare services while being able to minimize errors from misjudgments or misinterpretations [4]. Numerous studies have been conducted on the application of ML in tumor and cancer classification. Various ML algorithms have been implemented, some including a hybrid of multiple algorithms, and various types of datasets have also been used for comparisons.

Some studies involved the use of image-based data, such as magnetic resonance imaging (MRI) and computed tomography (CT) scan. For instance, Tang et al. [21] used MRIs for an image-based classification of tumor type as well as its growth rate prediction. ML models used in the study were decision tree, random forest, and support vector machine (SVM). The results are promising in terms of achieving a noninvasive marker for tumor type classification. Another image-based tumor classification study was performed by Rinesh et al. [22] using hyperspectral imaging (HSI). In this study, tumor localization in the brain was analyzed through performing different operations on hyperspectral images. By using k-nearest neighbors, k-means clustering, and multilayer feedforward neural network (MFNN), they were able to locate the tumor and label the areas of the brain. The proposed method was able to address the brain-molecule optimization process while minimizing error and trial techniques. In 2020, a study by Assiri et al. [23] classified breast tumors using an ensemble machine learning method. The data was obtained from the Wisconsin Breast Cancer Dataset (WBCD) that was based on a digitized image of a fine needle aspirate of a breast mass [24]. The classifiers used in the ensemble classification were simple logistic regression, SVM, and multilayer perceptron network. Hard vot-

ing mechanism was also implemented. Findings showed that the proposed method outperformed state-of-the-art algorithms used in the WBCD, and can be considered as a good starting point for image-based breast tumor classification.

The implementation of machine learning algorithms for tumor and cancer classification is also applicable to genomic data. The use of biological markers is being practiced to determine more specific tumor types and subtypes, and to refine known classifications of tumors, i.e. classifications of malignant tumors developed by World Health Organization (WHO) and Union for International Cancer Control (UICC). By refining these established classifications using molecular-genetic data, their application can provide help in cancer management studies, cancer genomics, and precision medicine [25].

Li et al. [26] performed pan-cancer classification using The Cancer Genome Atlas (TCGA) RNA-seq data. The TCGA dataset contains comprehensive molecular profiles of 31 tumor types for 9,096 patients. Along with this, RNA-seq samples of 602 "normal" tissue adjacent to tumors (of 17 types) were also used for testing. The study aimed to identify sets of genes with expression patterns that can distinguish various tumor types using genetic algorithm (GA) for feature selection and k-nearest neighbors (KNN) for the classification model. The proposed GA/KNN method is capable of identifying gene signatures that separates different classes of samples and reveals subtypes present in a class. The model was evaluated in two different ways. A pan-cancer classification of all tumors regardless of genders of the samples identified many sets of 20 genes that were able to correctly classify 90% of the testing set. Similar results were obtained when the model was evaluated by analyzing 23 non-sex-specific tumor types for females and another separate analysis for males. Li et al. inferred that these identified sets of genes may also become biomarkers for tumor diagnosis and drug development. Comparing the results of evaluation from the tumor samples with the "normal" samples showed that the top-ranked discriminative genes

in actual tumor samples reflect tumor-specific expression differences. They were also able to identify genes that may be incorporated with sexual dimorphism of specific cancer types.

With the emerging DNA microarray technology, molecular level research of tumors continuously rises. Data mining from gene expression data provides a more in-depth understanding of tumors at their genetic level. Furthermore, such data gives comprehensive information about disease pathology of tumors and comparison of genes between normal or abnormal/diseased conditions. A study by Liu et al. [27] classified tumors via gene expression data and sample-expansion based deep learning (DL). With the proposed sample expansion method, the problem of having rare DL-based studies in tumor classification due to inadequate training samples of gene expression data has been addressed. Feature selection was used to reduce the dimensionality of data to be used. Expanded samples were then obtained through randomly cleaning of corrupted input several times. These expanded samples were then used in two deep learning methods, Sample Expansion-Based Stacked Autoencoder (SESAE) and Sample Expansion-Based 1-dimensional Convolutional Neural Network (SE1DCNN). Testing was performed using three tumor datasets: breast cancer (30,006 genes on 20 samples), leukemia (12,600 genes on 60 samples), and colon cancer (2000 genes on 62 samples). They showed that both methods were effective in tumor classification as they achieved high classification accuracies on all three testing sets. This implied that the corrupted samples from the expansion method were deemed useful. Moreover, the expanded samples can generate a variety of gene combinations able to determine biological processes and represent the classes more effectively.

Upon examining these existing studies, one can see advantages brought by using genomic data in the field of tumor classification. Previous image-based studies achieved high performance measures on classifying tumors. However, they were not able to reveal in-depth differences between known tumor types and were not able to

identify other possible types of tumors. Genetic data on the other hand, has the potential to uncover these things. In addition to the goal of the studies to classify different tumor types, genetic data were able to reveal even more specific tumor subtypes as well as potential gene expression markers, gene combinations, and tumor-specific expressions that in return can be used as starting points for further studies of genetic interactions and processes.

e-Healthcare is a newly rising concept in the medical and healthcare sciences. This entails the integration of healthcare systems with information and communication technology (ICT). One characteristic of an ideal e-Healthcare system is that it allows for complete patient privacy. This covers provision of better data management to avoid issues like information leakage, data breach, and the like. While e-Healthcare offers advantages better than traditional healthcare methods, security and privacy concerns, especially in the case of patients, are among the major reasons that hold back its progress for practical applications [28].

Despite the progress of machine learning in tumor classification, one major problem that continues to be a concern is ensuring privacy-preserving computations of data, especially in the case of ML outsourcing. As observed in the aforementioned studies, they are greatly concerned with the performance of the classification models. However, there are no in-depth discussions that mention addressing potential violation of privacy rights of patients when their data is shared with external parties for computation processes.

In the area of machine learning, data collection and transfer to a central point is necessary in training a model. This poses a risk of information leakage for data with confidential and sensitive contents [29]. For instance, using genomic data holds confidential genetic information of patients which may be disclosed during the process of implementing ML techniques [30]. In return, this exhibits non-compliance with the first among the three security goals that must be maintained in applications of big

data - confidentiality, integrity, and availability (CIA) [31].

One possible solution to address data privacy problems is through the use of fully homomorphic encryption (FHE). FHE is a developing technology that supports arbitrary computations on encrypted data. Private machine learning has been gaining interest and FHE allows the ideal learning from data while keeping itself private [32]. In general, FHE is a type of homomorphic encryption (HE). HE is an encryption method that allows computations to be performed on encrypted data without the need of decryption. Similarly, the results returned by HE computations are encrypted and can only be decrypted by the private key owner. FHE to be more specific, supports any mathematical operation for an unlimited number of times [17]. With HE, the privacy of data is preserved because no disclosure of information is involved [30]. HE was also acknowledged as one of the central concepts for privacy-preserving computations in Templ and Sariyar's overview on methods for protecting sensitive data [7]. They identified HE to be among the backbones of federated analysis applications including banking, insurance, healthcare and medicine, retailing, and recommendation systems. With its low disclosure risk, HE is a promising approach for ML predictions.

Before the advancements in FHE, there are other state-of-the-art privacy-preserving techniques that are being used to address data privacy concerns as discussed by Mondal et al. [33]. k-Anonymization avoids the data of an individual to be identifiable by hiding within k similar records. While it prevents linkage attacks, sensitive information can still be revealed through homogeneity attacks and background knowledge attacks. It is also discouraged in precision medicine, which involves longitudinal follow-up studies, since anonymization prevents re-identification [34]. Meanwhile, differential privacy (DP) provides data privacy by adding calibrated random noise to data and thus, preventing reverse engineering on the data. With DP, one cannot determine if an individual's data is included in the original dataset because its presence hardly affects the output. However, drawbacks of DP include a limit for the

number of queries. Once the limit is reached, privacy is no longer guaranteed. It is also open to side channel attacks and like anonymization, sensitive data disclosure is still possible. In genomics, disadvantages of DP application include trade-off between privacy and utility as well as the pre-defined queries [7, 33, 34]. In particular, DP-based approaches are deemed inefficient because the amount of noise to be injected to genomic data containing millions of single nucleotide polymorphisms (SNPs) would greatly degrade the information's utility [35].

FHE can be used in a wide range of areas. As commercial-grade FHE is being pushed to be more available, opportunities centered around efficient data protection and handling can be seen. In his article, Creeger [36] mentioned about the different use cases of FHE. Implementing privacy-preserving computations on encrypted data ensures that data together with its results are protected against breach throughout its life cycle even on untrusted environments. FHE also secures data from quantum computing attack with its lattice-based post-quantum cryptography (PQC). Unauthorized disclosure of data and analytic results and models can be addressed with FHE-secure services. Even on combined datasets of multiple organizations, analysis can be performed securely (i.e. aggregating data from different vendors for facilitating pharmaceutical drug trials). Network traffic analysis is another domain that can be improved when FHE is applied. One can detect a threat actor without revealing threat signature, hence proving its potential use in computer network security and anti-money laundering. It is also possible for FHE to query if particular data exists in a larger data store without giving away the contents of the query or the data store, otherwise known as private set intersection. In blockchain, FHE together with Zero-Knowledge Proofs enables private transactions and can prove that they occurred even without data disclosure. Lastly, establishing confidential data resources using FHE can produce revenue streams for allowing them to be used by untrusted platforms.

Weng et al. [37] demonstrated the use of FHE in an artificial pancreas device

(APD) system that aids in monitoring and regulating a patient's blood glucose level. The problem lies with implementing the system in a cloud computing environment as it is highly concerned with data privacy risk of diabetes patients. Compromised data may lead to potentially life-threatening conditions of these patients or any other negative impact on their health status. Weng et al. proposed a secure APD system based on an FHE scheme, Cheon-Kim-Kim-Song (CKKS), with a Proportional-Integral-Derivative (PID) controller. They used Microsoft SEAL library that already implements their chosen scheme but there were limitations such as lack of support for division and comparison operations, need for polynomial approximation to evaluate more general functions, and more time and memory required for the computations. Hence, the rationale for using a PID-based controller since its algorithm can address these issues. To evaluate the proposed privacy-preserving APD system, it was compared with a plaintext APD system. There was no significant difference found between the two systems.

In recent years, there has been an increase in the number of studies related to integration of homomorphic encryption with artificial intelligence due to its promising approach to data privacy protection. Machine learning and deep learning studies [38, 39, 40] have been conducted and are still being developed with HE for different classification problems such as text classification [41], tumor classification [30], viral strain classification [42], etc. These wide range of HE applications with ML continue to expand with the help of existing libraries such as Microsoft SEAL, nGraph-HE, nGraph-HE2, IBM HElayers, TenSEAL, and ConcreteML implemented in varying programming languages like C++ and Python [43, 44, 45, 46, 47, 48].

Some of these libraries, namely Microsoft SEAL, HElayers, and TenSEAL, are generally used to perform homomorphic encryption operations. ConcreteML and nGraph-HE distinguish themselves from the others as they focus on integrating HE into ML. nGraph-HE allows for HE-based deep learning through Intel's graph com-

piler for neural networks [44, 45]. However, this project was declared to be only a proof-of-concept not intended for production and was recently announced to be discontinued and no longer maintained by Intel. ConcreteML provides an open-source set of tools for privacy-preserving ML inference framework based on FHE. It has ready-to-use built-in FHE-friendly ML models that are similar to the Scikit-learn ML library in Python and are compatible with its main workflows [48, 49].

Among these existing HE libraries, ConcreteML is more promising in ML applications for a number of reasons. Apart from its built-in FHE-based ML models, ConcreteML supports division operation that is not implemented in the other libraries [50]. It also allows for comparison operation that is generally hard to implement in FHE, and this is evident in the tree-based models that it offers [51]. To add, ConcreteML uses the Torus FHE (TFHE) scheme that supports programmable bootstrapping to evaluate homomorphic non-linear functions efficiently while reducing noise growth [52]. The parameter setting is also eased as it is handled in the library's compiler. Lastly, ConcreteML is implemented with Python, a high-level language, and makes it a user-friendly library. With these features, ConcreteML can be used even without extensive knowledge of cryptography.

In ConcreteML, the model is trained in plaintext and then compiled into its FHE equivalent. However, quantization of the model is required before compilation. It limits the scope of an input from a continuous set of values (real numbers) to a discrete set (integers). With quantization, the model becomes compatible with FHE, which is limited to 16-bit integers. Quantization method also varies depending on the ML model that will be used. Current limitations of the library include required quantization of the models and lack of support for pre-processing and post-processing of inputs and outputs. Improvements are expected in the coming months as these issues are currently being addressed [48, 49, 53].

In 2020, the iDASH privacy and security workshop held a competition for secure

genome analysis [3]. On the first track of the competition, participants were tasked to build a secure multi-label tumor classification using homomorphic encryption. A study that was awarded co-first place in this competition is the "Secure tumor classification by shallow neural network using homomorphic encryption" by Hong et al. [30]. The dataset used in the study was obtained from two publicly available datasets of TCGA samples, one containing somatic Somatic Nucleotide Variation (SNV) mutations and the other containing gene-level Copy Number Variation (CNV), for different tumor locations. Specifically, the dataset contains 3,622 instances from 11 cancer types and genetic features from 25,128 genes. They proposed a privacy-preserving solution to the problem by using a shallow neural network with softmax activation based on HE through an approximation method. CKKS scheme was implemented through the HEaaN library to perform inference step over encrypted data. The study addressed both problems of reducing the size of large-scale genetic data and difficulty in softmax function due to exponential and inverse functions. Findings of the study proved that their introduced CN and Variants data filtering methods during preprocessing was able to reduce the number of genes from 25,128 to 4,096 or less and attained a microAUC value of 0.9882 with a 1-layer shallow neural network. Overall, the proposed model was able to accomplish tumor classification inference on the testing set in a duration of only 3.75 minutes. In addition, this study was considered to be the first implementation of a neural network model with softmax activation using HE.

Another solution that won first place in the said competition was that of Inpher's team. Carpov et al. [54] built a framework, GenoPPML, for a privacy-preserving genomics ML pipeline. There were three actors involved in the framework: genomic database owners, data analyst, and final users. GenoPPML architecture was also divided into two phases: learning phase and prediction phase. In the learning phase, a joint logistic regression model is trained on datasets from the genomic database owners

through multiparty computation (MPC). To ensure privacy against model inversion attacks, differential privacy is implemented before revealing the trained model to the data analyst. Prediction phase commences when the analyst gets a hold of the differential private model and private prediction services are provided. Specifically, FHE inference is performed over the users' genomic data. Apart from being one of the most basic ML models applied in genomics, Carpov et al. chose logistic regression since it doesn't require any assumption on the hardware when implementing computations securely on medical data and thus, making it a promising approach in outsourced medical computations. Tools used in the study were the Inpher XOR library (for learning phase) and the open-source TFHE library (for prediction phase). The solution was benchmarked with three datasets from the TCGA database namely: (1) BC-TCGA, (2) GSE2034, and (3) BC12-TCGA. The first dataset consists of 17,814 genes and 590 samples in which 61 are normal tissues and the remaining 529 are breast cancer tissues. The second dataset has a total of 12,635 genes and 286 samples where 107 are recurrence tumor samples and 179 are no recurrence samples. The third and last dataset used in the solution is the same as the dataset used in the study of Hong et al. Overall, Inpher's HE for model prediction achieved one of the highest accuracies, 97.05%, and the lowest prediction time of 0.75 seconds in the iDASH 2020 track I competition. It was also awarded first place in track III of the same competition with the task on differentially private federated learning for cancer prediction [54, 55, 56]. Other teams that were part of the four-way tie for first place in iDASH's competition were Samsung SDS and Desilo. Samsung SDS also proved their HE technology in the competition by attaining high scores for both speed and accuracy of their analysis [57].

With regards to application of FHE-based machine learning in tumor classification, the amount of studies conducted over the years are not as many as compared to that of implementing traditional ML models. Thus, supporting the idea of the need

for exploration of FHE in this particular field of study. By doing so, it is not only the data privacy issues in ML that will be addressed but also the lack of practice of private machine learning. Additionally, combining this with the use of genomic data can provide a gateway for a secure research environment in the domain of precision medicine and genomic medicine.

III. Theoretical Framework

A. Homomorphic Encryption

Homomorphic encryption is an emerging technology - introduced in 1978 by Ronald Rivest, Len Adelman, and Michael Dertouzos - that lets direct computations be executed on encrypted data [36]. They recognized privacy homomorphism properties on encryption schemes wherein computations performed on plaintext data have the same result as the decrypted result of computations performed on their encrypted forms. In their observation of the Rivest-Shamir-Adleman (RSA) encryption, the product of encrypted numbers gave the same product as that of the plaintext numbers encrypted with the same key. With HE, it was found that it is possible to separate data access from data processing as it ensures data confidentiality even in untrusted environments and returns the result of the operations in its encrypted form that can only be decrypted by its corresponding private key.

Over time, three main types of HE were established: partially homomorphic encryption (PHE), somewhat homomorphic encryption (SHE), and fully homomorphic encryption (FHE). PHE allows only one operation between addition and multiplication to be performed for an unlimited number of times on ciphertext. SHE allows for both addition and multiplication on encrypted data but with a limit on the number of times they can be performed. Meanwhile, FHE encompasses both limitations of PHE and SHE since it allows for unrestricted homomorphic computations - both operations can be performed for an unlimited number of times [58, 59].

B. Fully Homomorphic Encryption

In 2009, Craig Gentry proposed the first FHE scheme. Originally, noise aggregated during HE computation and this limited the application of HE. In Gentry's paper, he was able to address this problem and showed that unrestricted computation on

encrypted data is possible through his noise reset process called bootstrapping [36]. It goes with decrypting and re-encrypting a ciphertext through a homomorphic computation that uses an encrypted secret key and a public key. However, two significant limitations were noted in his work. Bootstrapping computation exceeded the performance capabilities of available hardware platforms and conditionals that allow for both programmatic comparison and selection operations lacked efficient implementation. With continued research, four generations of FHE have been established and a variety of advancements has been made to improve the original Gentry scheme.

The first generation of FHE consisted of Gentry's 2009 scheme and the Dijk-Gentry-Halevi-Vaikutanathan (DGHV) scheme that changed the SHE portion of Gentry's, with an integer-based scheme, due to its slow computation. In 2011 and 2012, Brakerski-Gentry-Vaikuntanathan (BGV) and Brakerski/Fan-Vercauteren (BFV) schemes that make up the second generation introduced the concepts of learning with errors (LWE) and ring learning with errors (RLWE) security models. These models are similar to solving the closest vector problem in lattice mathematics and are based on the inability of determining coefficients, with each equation having a small and random additive error. LWE uses a system of linear equations while RLWE uses polynomial rings over finite fields. BGV and BFV also introduced leveling which enabled execution of a logic-gate circuit with a preset depth before bootstrapping. Relinearization reduces the length of ciphertext in order to minimize computational cost and storage burden but is a computationally expensive method itself. With Gentry-Sahai-Waters (GSW) scheme, relinearization in homomorphic multiplication was avoided and slow noise growth was observed. Bootstrapping was then simplified and optimized by the Ducas-Micciancio-"Fastest Homomorphic Encryption in the West" (FHEW) scheme. It also allowed for more efficient ring variants. These two schemes account for FHE's third generation. Lastly, the fourth generation of FHE consists of the Cheon-Kim-Kim-Song (CKKS) and torus fully homomorphic

encryption (TFHE) schemes. In 2016, the number of bootstraps in a logic circuit were lessened using the programmable bootstrapping (PBS) technique introduced by TFHE. This development was further extended by CKKS as it was able to control noise growth in HE multiplication through rounding operations for encrypted data [36].

Apart from their specific advancements over FHE, the aforementioned schemes differ in terms of their computation model. Homomorphic computation has three models: boolean circuits, exact/modular arithmetic, and approximate number arithmetic [36, 60]. In the boolean circuits approach, bits are used to represent plaintext while arbitrary boolean logic-gate circuits express the computations. It supports fast number comparison and bootstrapping and is implemented in GSW, FHEW, and TFHE. BGV and BFV, on the other hand, implement a modular arithmetic model that represents plaintext as integers modulo a plaintext modulus t and expresses its computations as integer arithmetic circuits mod t. Its features include fast and high-precision scalar multiplication and integer arithmetic, efficient single-instruction multiple data (SIMD) computations on vectors of integers, and a leveled design that can be used even without bootstrapping. The approximate number arithmetic implemented in CKKS makes use of computations that are similar with floating-point arithmetic and expresses plaintext as real numbers, including complex numbers. In addition to modular arithmetic's efficient SIMD computations and leveling, approximate number arithmetic can also support fast polynomial approximation and deep approximate computations. It also allows for relatively fast multiplicative inverse and discrete Fourier transform.

C. Workflow of ConcreteML

The lifecycle of a Concrete-ML model can be divided into two major parts: (1) model development and (2) model deployment [61].

Generally, there are four steps in the model development process of ConcreteML. First is the model training. Since ConcreteML only supports FHE inference as of the moment, the model can only be trained using plaintext data. The second step is quantization which can be performed either during training or post-training, depending on the ML model that is being developed. Quantization converts the model into an integer equivalent and makes it compatible with FHE. In ConcreteML, the number of bits, *n_bits*, parameter is used for this process. For linear models, *n_bits* can use a single integer value dependent on the number of attributes where it is recommended to use a lower value if the number of attributes is high. However, *n_bits* can also use different values through a dictionary passed to this parameter. For tree-based models, the value of *n_bits* must be less than 8 since such models need *n_bits*+1 bits for its maximum accumulator bit-width. Additionally, Concrete framework can only support up to 8-bit integers. The quantized model then undergoes compilation using Concrete's FHE compiler to obtain the final FHE ML model. This step is done entirely by the Concrete-Numpy backend of ConcreteML and complexity is hidden since execution is done simply through the `compile()` function. Lastly, FHE inference is done on encrypted data [53, 61, 62].

ConcreteML also allows the developed FHE ML model to be deployed in a client/server setting. Upon saving the model, three different files are generated - **client.zip**, **server.zip**, and **versions.json**. The client file consists of cryptographic parameters that will be used to generate both private and public evaluation keys. It also has the **serialized_processing.json** file which contains metadata about pre- and post-processing (e.g. quantization parameters). In ConcreteML, the user does not have to worry on security parameters since they are handled by the backend. It only uses a 128-bit security level and the optimal cryptographic parameters are automatically computed and included in the `compile()` method of the library that is handled by its Concrete-Numpy backend [63]. The server file consists of the compiled FHE model

and is used to run the model on a server. The last file holds information on which versions of ConcreteML and Python are used to compile the model. To allow for this process, ConcreteML uses its `concrete.ml.deployment.fhe_client_server` module that contains several APIs for FHE deployment [61, 64, 65].

To save the model and export all files for the client-server system, the `save()` method from FHEModelDev API is used. FHEModelServer API loads the FHE circuit and runs the model on the server via its `load()` and `run()` methods. FHEModelClient is the client API concerned with data encryption and decryption. The client uses `generate_private_and_evaluation_keys()` method to produce both private and evaluation keys. Then, the serialized evaluation keys are obtained using `get_serialized_evaluation_keys()` method. The input is quantized, encrypted, and serialized by calling `quantize_encrypt_serialize()`. Together with the evaluation keys, the encrypted input are sent to the server and prediction can now be performed. After FHE inference and sending the result back to the client, `deserialize_decrypt_dequantize()` is called to decrypt the prediction [65, 66].

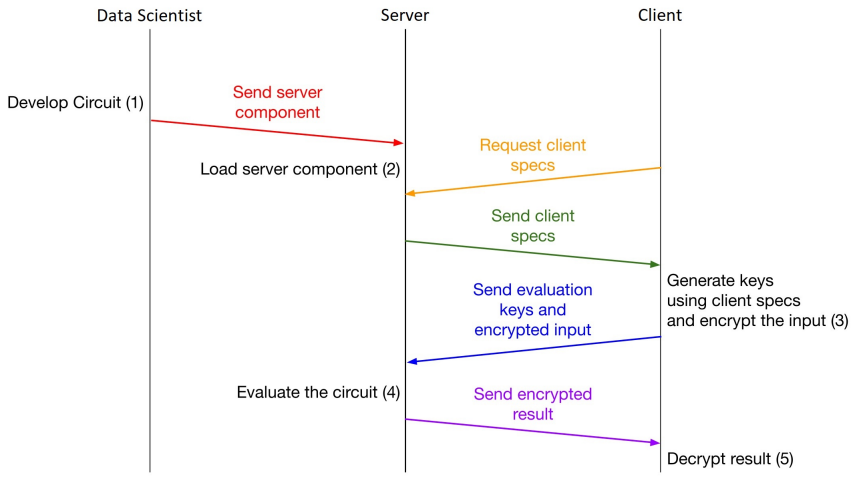


Figure 1: Overall communications protocol to allow cloud deployment of Concrete ML model [1].

D. Machine Learning Models in ConcreteML

ConcreteML provides a number of regression and classification ML models categorized into linear models, tree-based models, and neural networks. For linear models, it supports linear regression, logistic regression, linear support vector classification, linear support vector regression, poisson regression, tweedie regression, gamma regression, least absolute shrinkage and selection operator (LASSO), ridge regression, and elastic net [67]. Decision tree classifier, decision tree regression, random forest classification, and random forest regression compose the built-in tree-based models of ConcreteML. Extreme gradient boosting classification and extreme gradient boosting regression are also included in this category [51]. Neural network models include neural net classification and neural net regression which use Torch as a scikit-learn estimator and Skorch for scikit-learn compatibility [68, 69]. All models supported in ConcreteML are also supported in scikit-learn and are compatible with its main workflows.

E. Logistic Regression

Logistic regression is a widely used statistical model for classification problems and predictive analytics. While it is often used for binary classification, it can also be extended for a multi-class classification problem using multinomial logistic regression in which the dependent variable has three or more possible outcomes [70, 71]. Implementing this model is also simple and relatively faster than other supervised classification algorithms [72, 73]

In machine learning, logistic regression is often used in disease prediction [71] and is considered one of the most basic ML algorithms applied in genomic studies. In fact, previous studies such as that of Assiri et al. [23] and Carpov et al. [54] incorporated this model in tumor classification and demonstrated high accuracy and low computational time. In Assiri et al.'s study, simple logistic regression was one

of the top three among eight state-of-the-art ML models they evaluated on breast cancer dataset and used it to propose an ensemble classification for breast tumor classification. Carpov et al. enumerated two possible approaches to implement secure computations on medical data. The first is a distributed setting where multiple parties compute a function such as logistic regression and the second is assigning the computation to a third party in a non-interactive setting where trust is required on the hardware. As such, they chose the first approach as a promising way to outsource medical computation since it has no requirements of any assumption on the hardware.

F. Special Features of Genomic Data

With the rapid pace of advancements in genome sequencing technology, genomic data became widely available for collection, storage, processing, and sharing to support research in expanding and exploring its applications. However, its increasing availability also led to security and privacy concerns. The use of patients' information in research puts their confidentiality at high risk and this is more true for genomic data due to the complexity of information it holds [10].

According to Naveed et al. [10], genomic data has six properties that distinguish it from traditional health data: behavior, static, unique, mystique, value, and kinship. This type of data holds information about the health and behavior of an individual which allows for diagnosing medical conditions with high chances of being present in a person. Since genomic data doesn't change much over time, it can be used for long periods of time and has a highly important value that is likely to increase with time unlike traditional ones whose value often decreases with time. While it also contains information on an individual's blood relatives and reveals family relationships, genomic data is still unique for every person. There is also a public perception of mystery established about genomic data due to the fear of the unknown as everything about the genome is yet to be discovered and more studies are being conducted.

IV. Design and Implementation

A. Dataset Description

The study used brain cancer gene expression dataset found in Kaggle [74]. This dataset is part of a curated microarray database called CuMiDa[75]. Data consists of 130 samples, each having a sample ID and 54,675 genes as features.

A total of 5 classes were included in the dataset. Among these classes, four are different kinds of brain tumors namely: ependymoma, glioblastoma, medulloblastoma, and pilocytic astrocytoma. The remaining one class consists of normal samples. The brain tumor classes differ in which cell or part of the brain they start to grow from. Ependymoma starts from ependymal cells that can be found both in the brain and in the spinal cord [76]. Glioblastoma begins from astrocytes which are cells that gives nourishment and support to neurons [77]. Medulloblastoma usually forms in the bottom part of the brain, cerebellum [78]. Similar to glioblastoma, pilocytic astrocytoma also begins to form from astrocytes. Both brain tumors belong under astrocytoma tumors. However, glioblastoma is cancerous while pilocytic astrocytoma is considered non-cancerous [79].

Data imbalance is observed in the brain cancer dataset since there is a huge difference between the numbers of samples for each classes. Majority of the samples, 46, are under ependymoma and takes up 35% of the data. This is followed by glioblastoma with 34 samples or 26% of the data. Medulloblastoma, pilocytic astrocytoma, and normal samples make up 17% (22), 12% (15), and 10% (10) of the dataset respectively.

B. Preprocessing Techniques

The dataset was queried for checking of null values and it was determined that no missing values were found. Since the dataset was already normalized, as stated in CuMiDa's work [75], no more data normalization was performed. All genetic fea-

tures in the dataset were floats while only the class column contained string values. Label encoding was performed via the `LabelEncoder` module of scikit-learn's `preprocessing` package to convert the classes into integers that are compatible with the ML models to be used. To address the high dimensionality of the dataset, feature selection was implemented via scikit-learn's `feature_selection` module. A univariate feature selection method was performed through the `SelectKBest` module to select k highest scoring features based on univariate statistical tests [80, 81]. With k set to 20, the original 54,676 features in the dataset were reduced to the top 20 features.

C. Machine Learning Model

Several ML methods have been used in past studies of tumor classification. Integration of a few ML models have also been conducted. From these, models that have achieved high performance include k-nearest neighbors and k-means clustering combined with MFNN [22], ensemble classifier based on simple logistic regression, SVM, and multilayer perceptron network [23], 1-dimensional CNN [26], 1-layer shallow neural network [30], and logistic regression [54]. ML methods were also applied on datasets of CuMiDa [75] with SVM and random forest being the top performing algorithms. Among these models, only logistic regression, SVC, and random forest are supported by ConcreteML [67].

D. Performance Metrics

To compare the performance of the models, different metrics such as accuracy, balanced accuracy, and F1 score were obtained. Accuracy can be used to determine how often the model classifies a sample correctly. It refers to the fraction of the number of correct predictions out of the total number of predictions made by the model [82]. Balanced accuracy is the average of recall obtained on each class. F1 score is the

harmonic mean of the precision and recall [80].

While accuracy and F1 score are common metrics used in classification problems, it must be noted that the dataset is imbalanced. Between these two metrics, F1 score is considered a good metric for data imbalance since it keeps balance between the precision and recall of the model [83]. The *average* parameter on F1 score was set to the value *'weighted'* to account for imbalance. Balanced accuracy is an adjusted version of the standard accuracy metric to perform better on imbalanced datasets [84]. Computation of all three metrics were done for all models (scikit-learn, quantized plaintext, and FHE) through scikit-learn's `metrics` package.

E. ML Model Training Workflow

In the model training proper of the study, there were three kinds of models trained for each of the three ML algorithms trained:

1. **Scikit-learn model.** This is the standard plaintext ML model using scikit-learn library for which the models trained from ConcreteML will be compared to.

2. **Quantized plaintext model.** This is the plaintext ML model trained via ConcreteML's library. What makes it different from the scikit-learn plaintext model is the use of quantization process in training the model. Quantization is the process of converting the ML model from real numbers to integers. The conversion of floating point values in the model implies that there is trade-off between the accuracy and the representation of the values and thus, possibly leading to a degradation of the model [53]. However, this process is required to make the model compatible with FHE since the FHE model obtained in ConcreteML is a result of compiling the quantized plaintext model. Hence, this model is necessary to show the effect of quantization on the performance of the

trained models.

3. **FHE model.** This is the final FHE model that was obtained through compilation of the quantized plaintext model and will be used in the client-server system.

Each model was trained using the same dataset, preprocessing techniques, and train-test split. Training of these models were necessary to compare the performance of the standard ML implementation and FHE implementation. The training workflow for each algorithm is described with the following steps:

1. The Comma Separated Values (CSV) file of brain cancer gene expression dataset of CuMiDa [74] is read into a DataFrame using pandas' `read_csv()`.

2. Using scikit-learn's `preprocessing.LabelEncoder`, classes in the dataset are transformed from string into integers.

3. Feature selection is performed through scikit-learn's `feature_selection.SelectKBest` module. The value of k is set to 20 which reduces the number of features from 54,676 to 20.

4. The dataset is split into 90% training data and 10% testing data using scikit-learn's `model_selection.train_test_split` function.

5. The training data is used to fit the model. For quantized plaintext model, the `compile()` function of ConcreteML has to be called, after fitting, to obtain its corresponding FHE model.

6. Performance metrics are obtained using scikit-learn's `metrics` module.

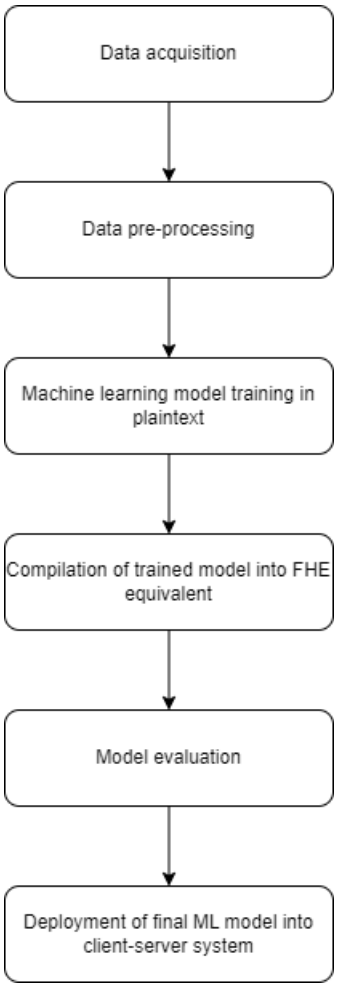


Figure 2: General workflow of the ML model development.

F. Input File Structure

The input file used in training the model and running the client-server system is a CSV file consisting of cancer gene expression data that has the following structure:

1. The input consists of a column for the sample's name or ID number. Value in this column may be a string (if name is used) or an integer (if ID is used).

2. The input has 54,675 columns representing genes. Value in these columns must be floats.

For the training input, an additional column is included in the CSV file consisting

of the tumor "type" (class) of the sample. Values in this column are string objects. For the client input, the columns for genetic features must follow the same genes used in the training dataset.

G. Use Case

A client-server system, representing a medical institution and an ML service provider respectively, was implemented to perform FHE inference on multi-class tumor classification. The medical institution encrypts genomic data and sends this to the ML service provider as an input for which FHE inference will be performed on. The ML service provider then returns the encrypted tumor classification result back to the medical institution for them to decrypt the result and to see the classification. Figure 3 shows the use case diagram of the system.

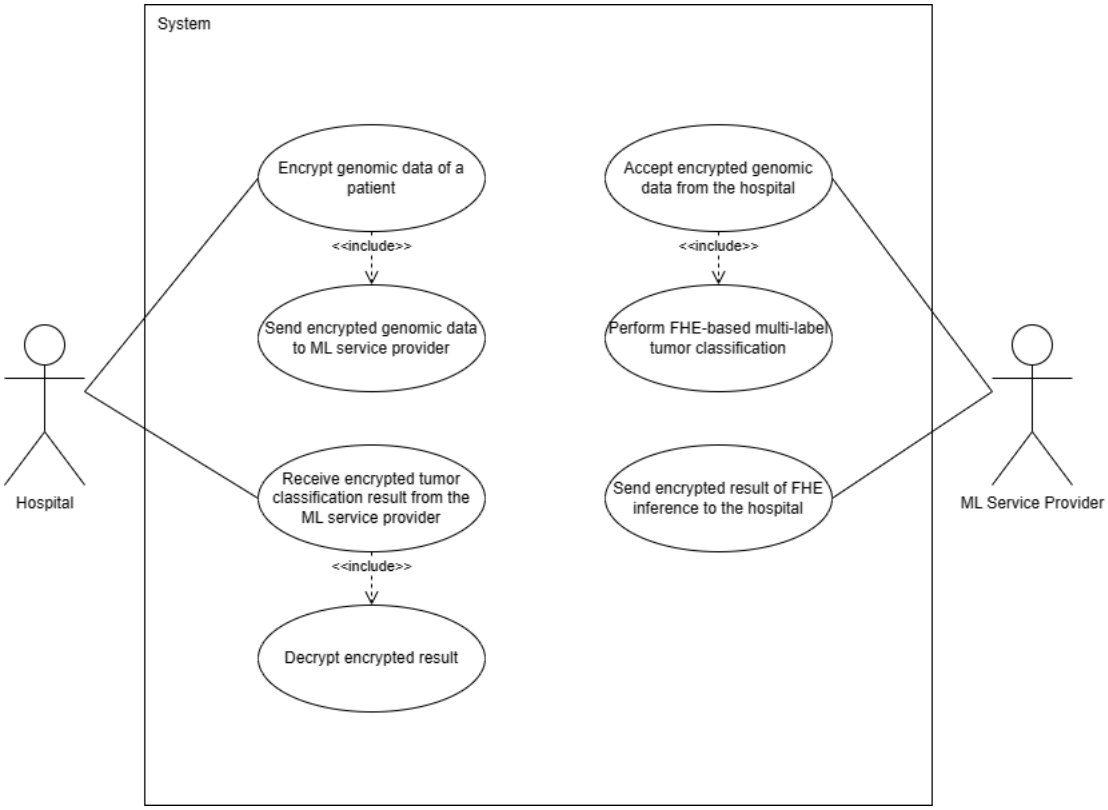


Figure 3: Use case diagram of the client-server system.

H. Threat Model

Consider the following threat models. We assume the system to be semi-honest, meaning the server follows the system's protocol honestly. We also assume that all communication channels between the client and the server are private and secure. If the security of the FHE scheme can be guaranteed, then there is no information leakage of the client's encrypted data even in malicious settings. We also assume that the client and the server does not collude with each other.

The security goals are as follows:

1. The server should not obtain information on the client's encrypted inputs.
2. The server should not obtain information on the encrypted inference results.

I. System Architecture

Before the actual system development, model training using logistic regression, random forest, and support vector classifier was performed to determine the best performing model. The following tools were used in implementing this part of the study:

- **pandas** - open-source library that provides data structures and analysis tools [85].
- **scikit-learn** - an open-source library that provides built-in ML models and tools for predictive data analysis [80].
- **ConcreteML** - open-source library that supports privacy-preserving ML inference based on FHE [48].
- **Google Colab** - a hosted Jupyter Notebook service [86].

Model training and development was mainly performed using pandas, scikit-learn, and ConcreteML in a Jupyter notebook via Google Colab. To read the dataset's CSV

file and create the dataframe, pandas library was used. The dataframe then went through preprocessing using packages from the scikit-learn library. These includes label encoding, feature selection, and splitting of the dataset for training and testing. Both scikit-learn and ConcreteML were used for fitting of the different models. To get the performance metrics of the trained models, the `metrics` package of scikit-learn was used. After selecting and saving the best performing model, the actual system development followed.

The system consists of two sides: a client and a server. The client-side consists of a graphical user interface (GUI) application and the server-side is a Django web application.The entire client-server system was implemented using the following technologies:

- **WSL2** - a tool that allows running of Linux environment on Windows [87].
- **Tkinter** - package used as the standard Python interface to the Tk GUI toolkit [88].
- **CustomTkinter** - a modern UI-library that was developed based on Tkinter [89].
- **ConcreteML**
- **pandas**
- **Django Web Framework** - a Python web framework mainly used for rapid web development [90].
- **Bootstrap** - a popular CSS framework for web and mobile applications [91].

Tkinter and CustomTkinter were the main tools used in developing the client GUI application. To define the methods, pandas and ConcreteML were used. The pandas library was used to implement the feature selection on the client's input data.

ConcreteML was then used to define the methods in the GUI application that would handle data encryption and prediction decryption. The GUI application sends the encrypted input to the server for prediction and decrypts the prediction result it receives from the server.

The server that holds the FHE model for classification was built using Django web framework. ConcreteML was used for the functionalities of FHE prediction. Bootstrap was used for the front-end development of the server. When the server receives the input sent by the client through the GUI application, it performs FHE prediction and sends back the prediction to the client.

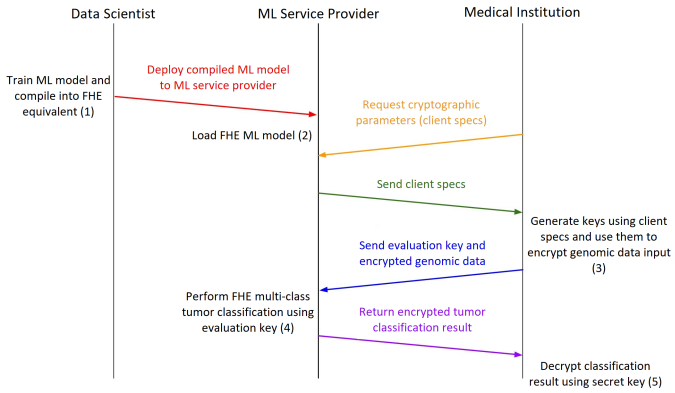


Figure 4: Detailed workflow of the client-server system.

Figure 4 shows the general workflow of the system. The data scientist is assigned with model development. This includes training of the models, model evaluation, compilation of the final model into FHE, and saving the compiled model. ConcreteML provides the FHEModelDev API that is used to save the final FHE model along with all files required to deploy it into a client-server system. This step can be done by simply calling the `save()` command of the said API. Saving the model generates three files (**client.zip**, **server.zip**, and **versions.json**) that are then stored in the **FHE-Compiled-Model** found in the Django project directory for the server. The **features_and_classes.txt** file containing the set of features selected in the dataset

preprocessing and the original class labels of the dataset before label encoding is also stored in the same folder. Using Python's os library to define filepaths, the server accesses the files needed in the different functionalities of the system. The **client.zip** stored in the folder is accessed to be sent to the client upon request and the **server.zip** is accessed whenever FHE inference is performed.

The server in the system represents ML service providers while a client represents medical institutions that will outsource ML computation. Upon launching the client application, the client requests for the **client.zip** file containing the cryptographic parameters for generating private and evaluation keys. The **features_and_classes.txt** file to be used for feature selection class translation is also requested. Once these files are received, the client can now use the application to submit input genomic data for preprocessing (feature selection). The output for preprocessing is then automatically subjected to encryption. ConcreteML uses commands from its FHEModelClient API to perform key generation and encryption. The following are the steps taken to encrypt the preprocessed input of the client:

1. Using the `generate_private_and_evaluation_keys()` method, private and evaluations keys are produced in the client side.

2. Serialized evaluation keys are accessed with the `get_serialized_evaluation_keys()` method.

3. The preprocessed input is subjected to quantization and encryption through the `quantize_encrypt_serialize()` method.

4. The final encrypted input is saved along with the evaluation keys.

After key generation and encryption, the client application sends the encrypted genomic data and evaluation key to the server. Interaction between the client and the server is implemented using Python's requests library. The client sends the

files to `http://localhost:8000/start_classification` where `localhost:8000` is the Django-based server ran in the local machine. When the server receives these files, FHE multi-class brain tumor classification is performed by running the `run()` method in ConcreteML's FHEModelServer API. The encrypted prediction result is compressed into a ZIP file before being sent back automatically to the client. Once the client application receives the result sent by the server, it accesses the ZIP file of the encrypted prediction and performs decryption to determine the plaintext classification result. The client application calls the `deserialize_decrypt_dequantize()` method from ConcreteML's FHEModelClient API to perform decryption on the prediction ZIP file using the private keys that were generated before encryption. The final decrypted inference of the client input is then displayed in the output window of the client application and is also saved into a CSV file.

J. Technical Architecture

The following are the requirements to run the developed client-server system:

- **Operating System:** Linux or Windows Subsystem for Linux 2
- **Storage Space:** 4.5GB free disk space

Linux is required to run the system. If using Windows, a Linux environment can be ran using the Windows Susbsytem for Linux. At least 4.5GB free disk space is needed considering the packages and dependencies used to run the system. Internet connection is also required since the client downloads required files, the **features_and_classes.txt** file and the **client.zip** file, upon launching of the client GUI application.

V. Results

A. Performance of the Machine Learning Models

A..1 Test Machine Specifications

The performance of the machine learning models trained was evaluated on a machine with the following specifications.

Test Machine Specifications	
Operating System	Windows 11 Home Single Language ver. 22H2
Processor	11th Gen Intel(R) Core(TM) i3-1115G4 @ 3.00GHz
Installed RAM	12.0 GB
System Type	64-bit operating system, x64-based processor

Table 1: Device and system specifications of the machine used for performance evaluation of the ML models.

A..2 Model Accuracy and F1 Score

The models used in the study were logistic regression, random forest, and support vector classifier. Model development was conducted in three runs using the same dataset and preprocessing methods. For each algorithm, three types of models were developed: scikit-learn plaintext, ConcreteML plaintext (quantized), and ConcreteML FHE models. To determine the performance of the models, three metrics were used: accuracy, balanced accuracy, and F-1 score.

	Accuracy			Balanced Accuracy			F1 Score		
	Sklearn	Quantized	FHE	Sklearn	Quantized	FHE	Sklearn	Quantized	FHE
Run 1	84.6154%	84.6154%	84.6154%	85.0000%	85.0000%	85.0000%	84.6154%	84.6154%	84.6154%
Run 2	84.6154%	84.6154%	84.6154%	87.1429%	87.1429%	87.1429%	85.0888%	85.0888%	85.0888%
Run 3	92.3077%	92.3077%	92.3077%	95.0000%	95.0000%	95.0000%	95.7265%	95.7265%	95.7265%

Table 2: Performance metrics of each LR model in each run.

Table 2 consists of the computed metrics for each kind of logistic regression model in all three runs. As observed here, there is no difference between the performances of the scikit-learn plaintext, quantized plaintext, and FHE models. Additionally, the values for each metric are all above 80%.

	Accuracy			Balanced Accuracy			F1 Score		
	Sklearn	Quantized	FHE	Sklearn	Quantized	FHE	Sklearn	Quantized	FHE
Run 1	92.3077%	92.3077%	92.3077%	90.0000%	90.0000%	90.0000%	91.7949%	91.7949%	91.7949%
Run 2	69.2308%	53.8462%	53.8462%	64.4444%	55.5556%	55.5556%	67.6627%	59.1270%	59.1270%
Run 3	84.6154%	76.9231%	76.9231%	90.0000%	86.6667%	86.6667%	84.6154%	79.2308%	79.2308%

Table 3: Performance metrics of each RF model in each run.

For random forest models, performance metrics are shown in table 3. A similar observation with LR models was observed in the first run wherein all kinds of models has similar performances. However, in the second and third runs, there was a decrease in the performance metrics of the quantized plaintext and FHE models. The range of values of the metrics is also wider as compared to LR models. While some reached as high as 92%, metrics also went as low as almost 54%.

	Accuracy			Balanced Accuracy			F1 Score		
	Sklearn	Quantized	FHE	Sklearn	Quantized	FHE	Sklearn	Quantized	FHE
Run 1	76.9231%	76.9231%	76.9231%	80.4167%	80.4167%	80.4167%	80.2198%	80.2198%	80.2198%
Run 2	61.5385%	61.5385%	61.5385%	68.0000%	68.0000%	68.0000%	56.5934%	56.5934%	56.5934%
Run 3	76.9231%	76.9231%	76.9231%	73.3333%	73.3333%	73.3333%	77.7855%	77.7855%	77.7855%

Table 4: Performance metrics of each SVC model in each run.

Table 4 shows the performance of all SVC models in all runs. Like the LR models, all kinds of models for SVC have similar performances. There was no observed decrease or increase in each metric for all three kinds of SVC models and for all three runs done.

	Accuracy			Balanced Accuracy			F1 Score		
	Sklearn	Quantized	FHE	Sklearn	Quantized	FHE	Sklearn	Quantized	FHE
LR	87.1795%	87.1795%	87.1795%	89.0476%	89.0476%	89.0476%	88.4769%	88.4769%	88.4769%
RF	82.0513%	74.3590%	74.3590%	81.4815%	77.4074%	77.4074%	81.3577%	76.7176%	76.7176%
SVC	71.7949%	71.7949%	71.7949%	73.9167%	73.9167%	73.9167%	71.5329%	71.5329%	71.5329%

Table 5: Average of performance metrics of each model for all runs.

To compare the overall performance of the different ML algorithms, the average of each model for each metric was computed. Results are displayed in table 5. Looking at the values, logistic regression outperformed both random forest and support vector classifier in all metrics.

The behavior of the similar performance between the scikit-learn, quantized, and FHE models can be attributed to the quantization bits used in training. The training of the ConcreteML models used the default value for quantization bits which is 8. In Li's article covering the application of ConcreteML, an example using 8 bits for quantization achieved the same results where ConcreteML models have the same performance as the scikit-learn model [92]. Li's results also showed that the quantized plaintext and FHE models achieved the same performance even on cases where scikit-learn performed best. Additionally, linear models in ConcreteML tend to achieve the same performance as their FP32 counterparts since they only require very little quantization. Meanwhile, tree-based classifiers in ConcreteML are expected to achieve good performance but not exactly identical to their FP32 counterparts [93]. Thus, possibly explaining the difference between the scikit-learn and the ConcreteML models using random forest.

B. Client-Server System

The system consists of two main parts: client and server. The client refers to medical institutions, such as hospitals, that holds genetic information of patients and

outsources machine learning for tumor classification. The server is an ML service provider for multi-class tumor classification.

B..1 Client-Side

For the client, a graphical user interface was built such that it contains all functionalities necessary to process the data for feature selection, encryption, and sending to server for prediction. It also has a function for decrypting the prediction returned by the server.

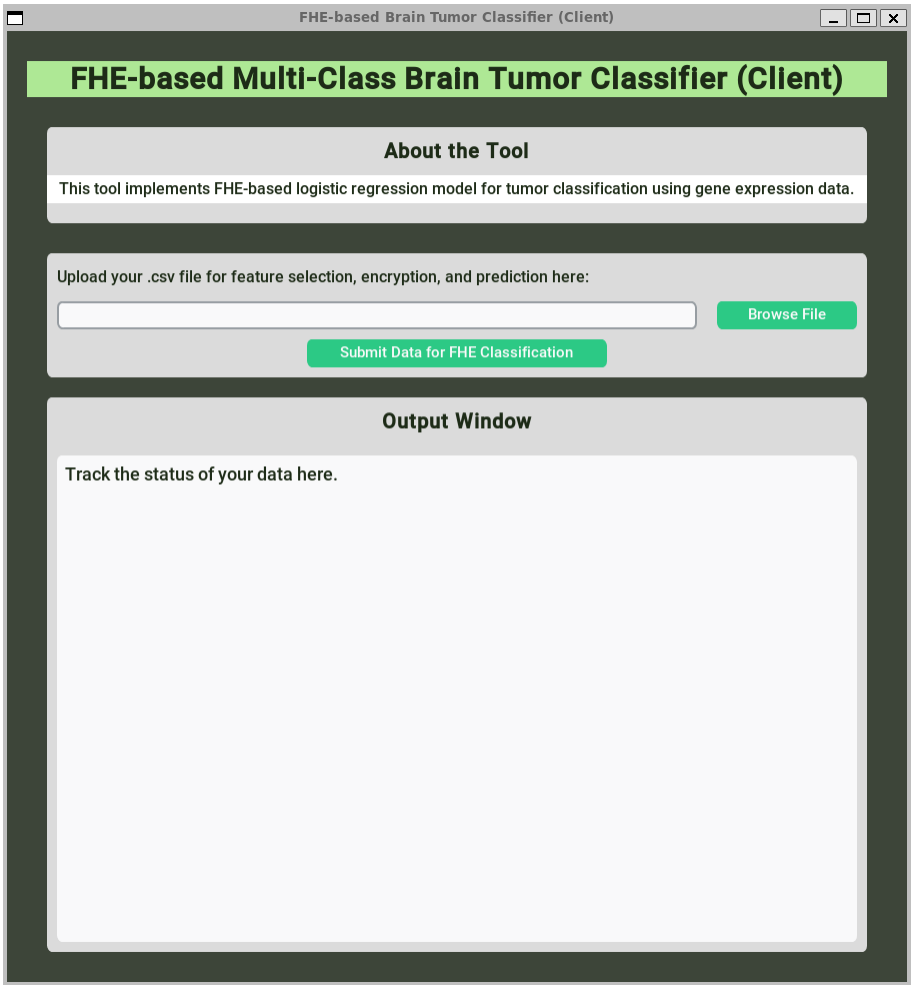


Figure 5: Client GUI upon launching.

Figure 5 shows how the client GUI would look like upon launching of the application. A short description about the main function of tool is provided. A separate

section for the client input is also seen. This is where the client would input the filepath or browse for the input file before submission. The output window is where the status of the whole process from preprocessing to prediction decryption is displayed.

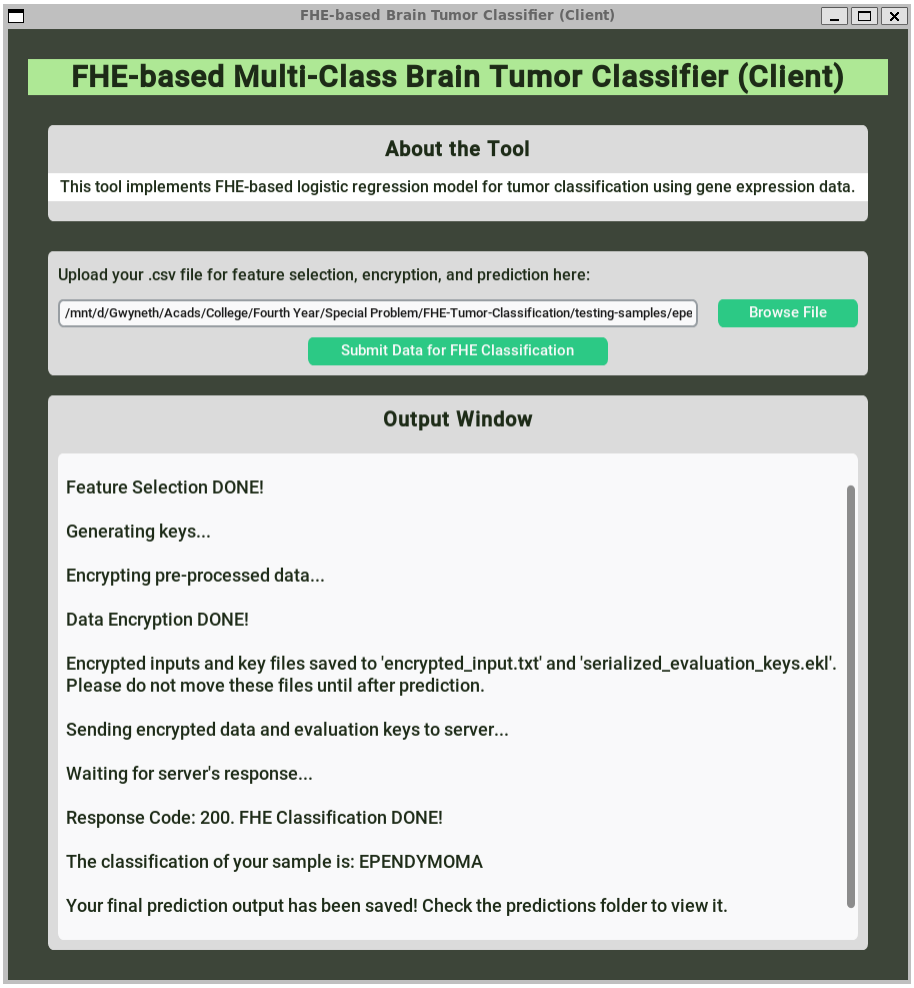


Figure 6: Client GUI with status outputs after FHE prediction.

Figure 6 demonstrates how the output window would look like when the client has finally entered an input file. First, the input file undergoes feature selection. Next, keys are generated to be used for the encryption and decryption processes. Once data encryption is done, the client GUI sends the encrypted data and evaluation keys to the server for FHE tumor classification. The server's role stops at sending back the result to the client. In an FHE ML model, the result of the prediction is also encrypted.

Thus, the client needs to decrypt it to be able to know the final classification output. Once the client receives the prediction result sent by the server, the GUI automatically starts the decryption process and displays the decrypted classification result. It also saves the prediction in a .csv file containing the ID of the sample as well as its FHE tumor classification.

B..2 Server-Side

The server side of the system is a simple home page that holds the model, as seen in figure 7. Since the client GUI automatically processes the client data and sends it to the server once encrypted, the server no longer asks for a file upload of the encrypted input as well as the evaluation keys to be used for FHE prediction. It also doesn't include an output window since all processes can already be tracked in the output window of the client GUI. Apart from this, the server home page can redirect the client to the Github page of the whole system. The Github of the system is a public repository that contains all source codes and documentations.

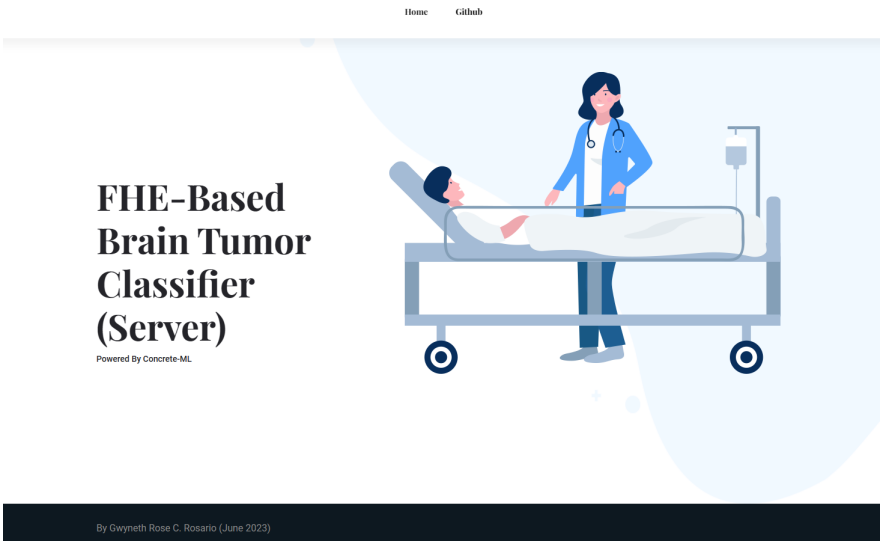


Figure 7: Server home page.

VI. Discussions

The FHE-based Multi-class Tumor Classification Client-Server system is a simple system that aims to implement a privacy-preserving classification tool that uses genomic data as its input.

The dataset used for training of the models is publicly available in Kaggle [74]. Before the training proper, preprocessing techniques were performed in the data. Since no missing values were found and the data was already normalized [75], only label encoding of the classes and feature selection were performed.

A train test split of 90-10 was used, meaning 90% (117) of the samples were used in training the model while 10% (13) of the samples were used in testing. Since the dataset is imbalanced, the *class_weights* parameter in fitting the models were set equal to *'balanced'* to address this problem.

The results of the ML model development showed that logistic regression yielded the best performance among the three algorithms tested. For all runs, SVC achieved the lowest metrics. While random forest also achieved high metrics, results ranged from 69.2308% to 92.3077% for scikit-learn and 53.8462% to 92.3077% for ConcreteML. For logistic regression, metrics for all models, scikit-learn and ConcreteML, performed similarly and ranged from 84.6154% to 95.7265%. The average of performance metrics of all models for all runs also showed that logistic regression indeed had the highest accuracy, balanced accuracy, and F1 score. Hence, logistic regression was used as the final model in building the proposed client-server system.

The logistic regression model used in run 3 achieved the highest metrics among all runs of the said model. Its accuracy, balanced accuracy, and F1 score are all over 90% which are considered to be excellent values [94, 95, 84]. Specifically, the accuracy of the model is 92.3077%, balanced accuracy is 95.0000%, and F1 score is 95.7265%. Its FHE model was used in the implementation of the client-server system.

Logistic regression algorithm, as applied in scikit-learn, quantized plaintext, and

FHE models, was further evaluated by testing its performance in terms of running time and error analysis. The FHE model had an additional evaluation on its key size and ciphertext size since this model includes encryption and decryption.

A. ConcreteML Performance

A..1 Running Time Analysis

A runtime analysis was performed by comparing the difference between the scikit-learn plaintext, quantized plaintext, and FHE. This includes the training time and prediction time for all models and compilation time for the FHE models.

	Sklearn	Quantized	Quantization Time
Run 1	34.4493	81.3487	46.8993
Run 2	60.7603	128.1841	67.4238
Run 3	27.0460	88.1531	61.1072
Average	40.7519	99.2286	58.4768

Table 6: Training time of plaintext models and quantization time of quantized model (in milliseconds).

	Sklearn vs Quantized
Run 1	136.1400%
Run 2	110.9670%
Run 3	225.9382%
Average	157.6817%

Table 7: Increase in training time of the models.

A total of three runs were performed to check the running time of the LR models in terms of training. As seen in table 6, the training times recorded are only in the two plaintext models since there is no actual training performed for FHE. In ConcreteML,

the trained quantized plaintext model is simply compiled to obtain its equivalent FHE model.

For all runs, the training time of quantized plaintext are higher than that of scikit-learn's. The increase in the training time of quantized plaintext is also computed and displayed in table 7. The average training for quantized plaintext and FHE is 99.2286 milliseconds which is 157.6817% higher than scikit-learn's average training time of 40.7519 milliseconds. This is due to quantization performed to obtain these models. In ConcreteML, quantization is performed post-training for linear models such as logistic regression[53]. Since the function for fitting the model already includes the quantization process, the time it takes to quantize the model adds up to the total training time measured. To compute the quantization time, the training time in scikit-learn was subtracted from the total running time of quantized plaintext model training and quantization. An average of 58.4768 milliseconds adds up to the running time of quantized plaintext model due to the added quantization task.

	Compilation Time (milliseconds)
Run 1	174.7668
Run 2	178.0674
Run 3	376.7128
Average	243.1823

Table 8: Compilation time of quantized plaintext models into FHE (in milliseconds).

The running time of compiling the quantized plaintext models into FHE was also recorded in table 8. Over three runs, the average compilation time was 243.1823 milliseconds.

Prediction time of the models was also tested over three runs. Two categories for prediction time were checked: one for the entire test set and the other one is per sample. For the FHE model, prediction time only includes the actual inference step

and not the decryption step. The results are displayed in the following tables.

	Sklearn	Quantized	FHE
Run 1	0.3514	0.6166	53.8583
Run 2	0.4718	0.5169	84.7387
Run 3	0.3741	0.4826	46.1643
Average	0.3991	0.5387	61.5871

Table 9: Prediction time of models (in milliseconds) for the entire test set.

	Sklearn vs Quantized	Sklearn vs FHE	Quantized vs FHE
Run 1	75.4410%	15225.5088%	8635.4215%
Run 2	9.5503%	13644.0062%	16293.9114%
Run 3	28.9994%	12240.7903%	9466.5514%
Average	37.9969%	13703.4351%	11465.2948%

Table 10: Increase in prediction time for the entire test set of the models.

The entire test set consists of 13 samples, equivalent to 10% of the whole dataset. The results in tables 9 and 10 hold the actual average prediction time for the entire test set in milliseconds and the average increase in prediction time of the quantized plaintext and FHE model compared to scikit-learn and the FHE model compared to the quantized plaintext. Scikit-learn has the shortest average prediction time of 0.3991 milliseconds followed by quantized plaintext with 0.5387 milliseconds. Compared to these two, the increase in FHE is quite large, resulting to an average prediction time of 61.5871 milliseconds. In table 10, results show that the least average increase in prediction time is scikit-learn versus quantized plaintext, followed by quantized plaintext versus FHE, then scikit-learn versus FHE.

In computing the prediction time per sample, one sample for each of the 5 classes were used. Hence, a total of 5 samples were predicted in each run and the running

time in predicting each of them was averaged. The mean running time of prediction per sample in each run is recorded in table 11.

	Sklearn	Quantized	FHE
Run 1	0.2785	0.5038	5.0600
Run 2	0.2174	0.4469	7.5726
Run 3	0.3989	0.7658	4.9873
Average	0.2983	0.5722	5.8733

Table 11: Prediction time of models (in milliseconds) per sample.

	Sklearn vs Quantized	Sklearn vs FHE	Quantized vs FHE
Run 1	80.8937%	1716.7437%	904.3157%
Run 2	105.5714%	3383.3955%	1594.4942%
Run 3	91.9555%	1150.1913%	551.2921%
Average	92.8069%	2083.4435%	1016.7007%

Table 12: Increase in prediction time per sample of the models.

The results of the prediction time per sample in tables 11 and 12 show a similar behavior with the results in the prediction time for the entire test set. Scikit-learn has the shortest prediction time while FHE has the highest prediction time. The increase in prediction time per sample also shows that the least increase can be seen in scikit-learn versus quantized and the greatest increase is in scikit-learn versus FHE.

	Entire Test Set	Per Sample
Sklearn vs Quantized	0.35x slower	0.92x slower
Quantized vs FHE	113.33x slower	9.27x slower
Sklearn vs FHE	153.31x slower	18.69x slower

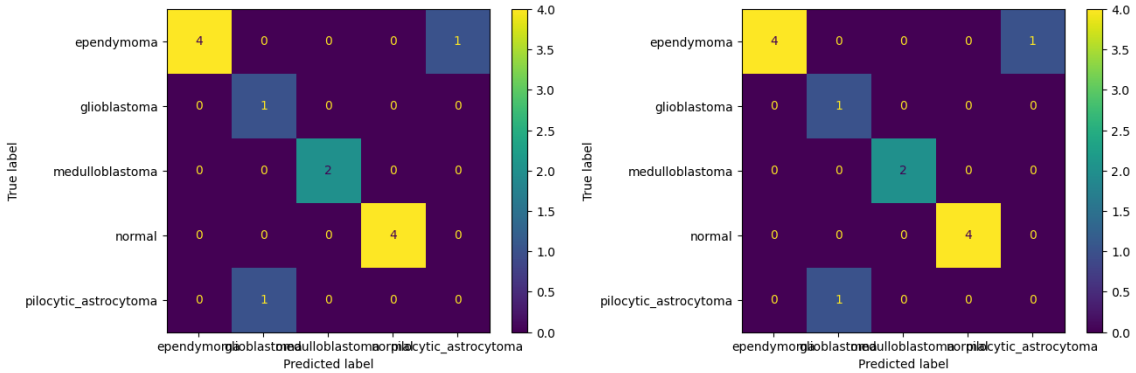
Table 13: Average slow down in prediction time of the models for the entire test set and per sample.

The average slow down in prediction time for the entire test set and per sample were also computed and results are shown in table 13. The greatest slow down is observed on scikit-learn versus FHE and the least slow down is on scikit-learn versus quantized plaintext. These results go in parallel with the increase in prediction time shown in tables 10 and 12.

Combining the results from tables 9 to 13, there is an increase in the prediction time of both quantized plaintext and FHE models which can be attributed to the quantization applied as part of the model development in ConcreteML. Performing matrix multiplications or convolutions with quantized values lead to more complex computing equations [53]. While most ML operations use integer arithmetics that are compatible with quantized values, a model cannot simply replace floating point operations with an equivalent integer operation. Important quantization parameters such as scale factor and zero point are used in operations over quantized values to ensure that the final output of the operation is also quantized [96, 97]. Hence, making the operation more complex and in return may affect the running time of the task.

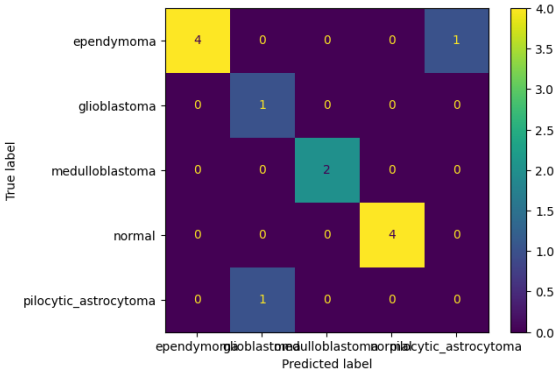
A..2 Error Analysis

To further evaluate the performance logistic regression, error analysis was also performed. The performances of scikit-learn, quantized plaintext, and FHE models were compared in terms of misclassifications.



(a) Scikit-learn

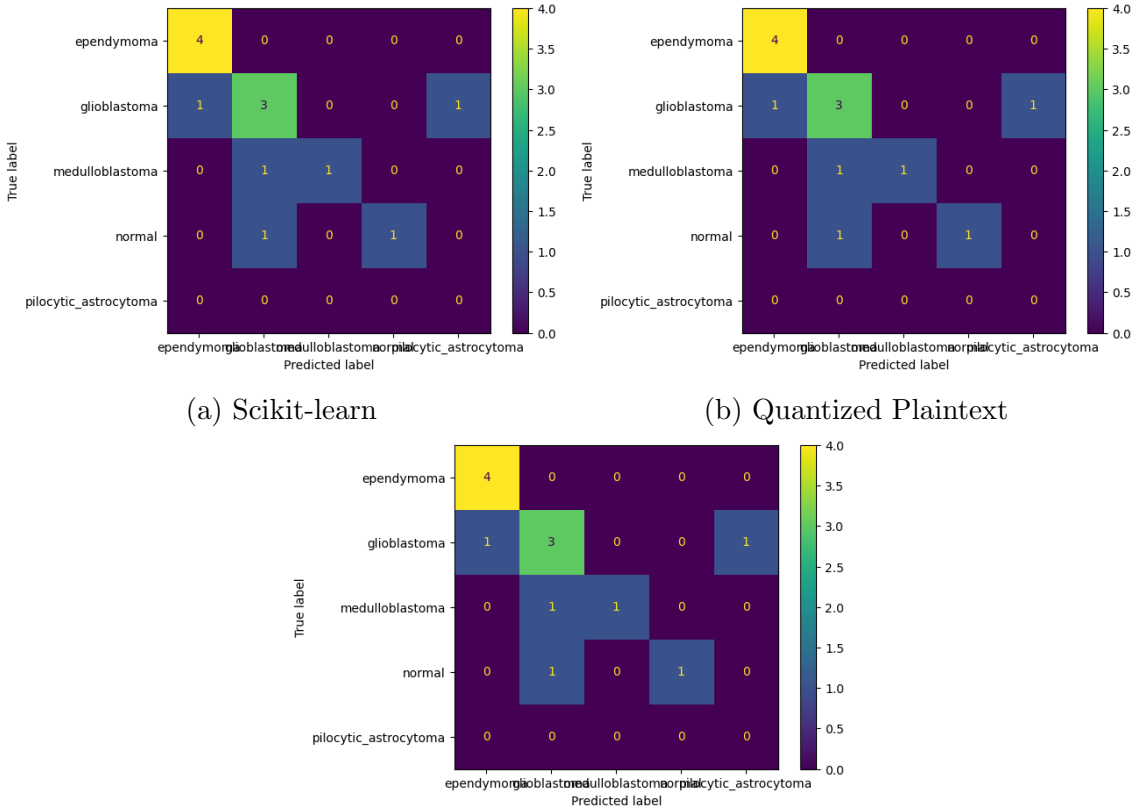
(b) Quantized Plaintext



(c) FHE

Figure 8: Confusion matrices of the models in run 1.

Figure 8 displays the confusion matrices of the three LR models trained in the first run. It is observed that the performance of the three models do not differ from each other due to the same number of correctly classified samples as well as the same misclassified samples.



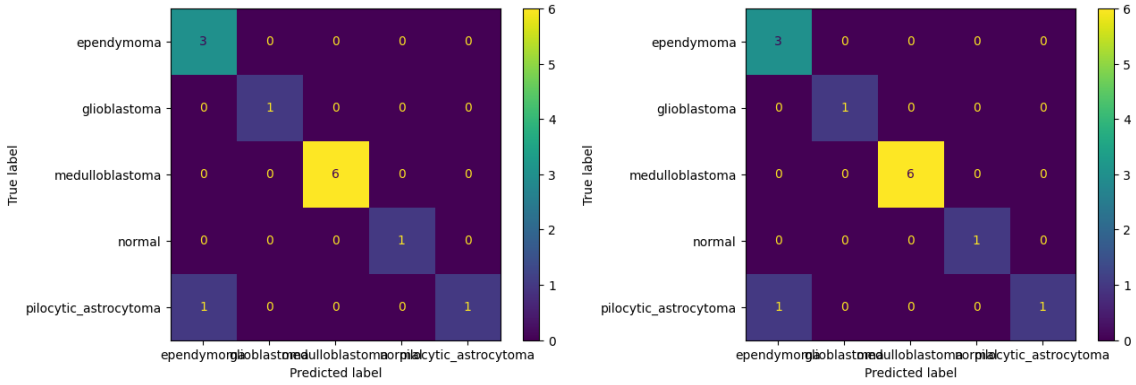
(a) Scikit-learn

(b) Quantized Plaintext

(c) FHE

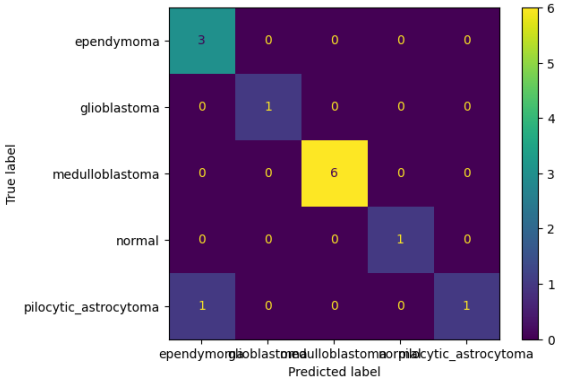
Figure 9: Confusion matrices of the models in run 2.

The results of the second run in figure 9 also shows the same results observed in the first run. All models have the same confusion matrix which means they have the same misclassified samples and correctly classified samples.



(a) Scikit-learn

(b) Quantized Plaintext



(c) FHE

Figure 10: Confusion matrices of the models in run 3.

For the third run of the model, the same results were still observed. That is, scikit-learn, quantized plaintext, and FHE had the same confusion matrix. Misclassification of the scikit-learn model is the same as the misclassification of the quantized plaintext and the FHE model. These results are also expected since the performance metrics obtained for logistic regression did not show any difference between the three kinds of models trained.

A..3 Ciphertext Size and Key Size

Additional factors such as ciphertext size and key size were recorded to further discuss the performance of the FHE model. Since a ConcreteML FHE model has fixed cryptographic security parameters computed by its Concrete-Numpy backend [63] and

saved to the client and server files, the encryption process is the same every time an FHE inference is performed. Thus, explaining why the key size and ciphertext size measured in ConcreteML are constant as shown in tables 14 and 15.

	Key Size (kB)	Increase
ConcreteML FHE Evaluation Key	0.023	-
ConcreteML FHE Private Key	4.000	65.70%
128-bit Security Level RSA Encryption	2.414	-

Table 14: Comparison between ConcreteML FHE and 128-bit security level RSA encryption key size (in kilobytes).

In cryptography, the current strongest standard key size for encryption is the 2048-bit RSA key [98]. It follows the National Institute of Standards and Technology (NIST) recommendation to use keys with a minimum strength of 112 bits of security in data protection until year 2030 since a 2048-bit RSA key provides 112-bit of security [99]. However, ConcreteML's security level is set to 128-bit [63]. To ensure a fair comparison, the standard reference used in this study is the RSA encryption with 128-bit security level.

Table 14 indicates that there is an increase of 65.70% in the private key size of ConcreteML FHE as compared to the standard RSA 128-bit security algorithm. From 2.414 kilobytes, the key size increased to 4 kilobytes. This can be expected since a stronger encryption, in this case FHE that allows computations on ciphertext, is expected to have higher key size [100]. However, this doesn't mean that FHE is entirely better than the RSA algorithm. The large increase in the key size also implies that there is a trade-off between efficiency and memory. The result shows that FHE requires more memory for its private key just to attain the same security level as the RSA algorithm. A larger key size also means that more computation time is needed on both sides of the system [99].

	Clear Input Size (kB)
Run 1	0.325
Run 2	0.330
Run 3	0.329
Run 4	0.332
Run 5	0.332
Average	0.330

Table 15: Size of clear input for encryption over five runs (in kilobytes).

	Ciphertext Size (kB)	Increase
ConcreteML FHE	240.197	63952.53%
128-bit Security Level RSA Encryption	0.375	-

Table 16: Comparison between ConcreteML FHE and standard encryption ciphertext size (in kilobytes).

FHE preserves the privacy of data by only using encrypted data as an input for inference. The trade-off comes in the size of the encrypted input data. To evaluate the system in this aspect, five runs using different gene expression data inputs were subjected to two encryption schemes: FHE and RSA. The average size of the clear input for encryption is 0.330 kilobytes, as indicated in table 15. Table 16 shows that for all encrypted data produced, the size is consistent for the two encryption schemes. Before the encryption process, the raw data inputs undergo the same preprocessing, in this case feature selection, that already results to a designated format to be then encrypted. The same encryption process is also applied to every input so it is expected that the size of the ciphertext produced is also the same for each run.

RSA's ciphertext size is 0.375 kilobytes while FHE's ciphertext size is 240.197 kilobytes. This shows an increase of 63952.53% in the standard encryption. Similar

to the key size, these results demonstrate the trade-off between efficiency and memory when using FHE. While FHE provides a stronger encryption in the data, it needs more space for the ciphertext it generates to achieve the same security level as the RSA encryption.

A..4 Additional Investigation on Error Analysis

While not directly related to the final model's performance evaluation, a unique case encountered in the early stages of trying out the model training is when the two models from ConcreteML performed better than the scikit-learn model. This is particularly observed in a Random Forest model.

Accuracy			Balanced Accuracy			F1 Score		
Sklearn	Quantized	FHE	Sklearn	Quantized	FHE	Sklearn	Quantized	FHE
84.6154%	92.3077%	92.3077%	85.4167%	93.7500%	93.7500%	84.4156%	92.3077%	92.3077%

Table 17: Performance metrics of RF models where ConcreteML performed better than scikit-learn.

In table 17, there is a noticeable difference between the performance of the scikit-learn model and the ConcreteML models. Normally, this can be expected as the quantization process may cause degradation in the model performance. However, this case shows that no degradation was observed in the ConcreteML models and they even performed better than scikit-learn.

```
SKLEARN PREDICTION:
 [0 1 3 4 0 4 0 1 0 0 4 1 0]
QUANTIZED CLEAR PREDICTION:
 [0 1 3 4 0 4 0 1 0 1 4 1 0]
FHE PREDICTION:
 [0 1 3 4 0 4 0 1 0 1 4 1 0]
ACTUAL:
 [0 4 3 4 0 4 0 1 0 1 4 1 0]


Quantized vs FHE Comparison: 100% similar
Sklearn vs FHE Comparison: 92% similar
```

Figure 11: Similarity on the prediction of the RF models on the entire test set.

Figure 11 shows the prediction of the RF models on the entire test set and the actual labels of the samples in the set. The testing set comprises 10% (13 samples) of the dataset. Out of the two misclassifications predicted by the scikit-learn model, one is also misclassified by the ConcreteML models while the other is correctly classified by them. Thus, explaining why the similarity between the prediction of the scikit-learn model and the FHE model is only 92%.

To better understand this case, we computed the condition number of each RF model on the entire testing set. The condition number is used to measure the sensitivity of a solution to changes in the input data [101]. It gives an idea on the precision of a solution wherein a larger value implies a larger change in the output when input changes are observed [102]. The condition number, however, does not directly describe a comparison of performance between the models. Results are recorded in the following table.

	Condition Number
Sklearn	27.1759
Quantized	65.2992
FHE	65.2992

Table 18: Condition number of the models in the RF algorithm case where ConcreteML performed better than scikit-learn.

Looking at table 18, the condition number of the ConcreteML models are significantly larger than that of the scikit-learn. If the same change was performed over the three models, the ConcreteML models tend to have greater changes in their outputs. In this case, there is a high possibility that the ConcreteML models are able to produce the correct classifications only because of chance caused by the large change in their outputs. However, further investigations are still needed to ensure the reasoning behind such cases.

B. System Assessment

Next to model evaluation is the testing of the actual client-server system. Upon running all functionalities of the system, all of the main objectives were implemented. The client was able to preprocess their data, generate keys to be used for encryption and decryption, generate evaluation keys to be sent to the server, and encrypt their preprocessed data. Following the key generation and data encryption, the client GUI was able to successfully send the required files for the server to perform FHE classification. The functionality for the client to decrypt the prediction sent by the server is also working as intended. Likewise, the server's functionalities also fulfilled the objectives of the study. It was able to perform FHE classification with the client's encrypted input and was able to send this back to the client for decryption.

The developed client-server system stands out from existing tumor classification

FHE works for several reasons. The works of Seoul National University [30] and Inpher [54] used HEaaN and TFHE libraries respectively in implementing homomorphic encryption. The ConcreteML library that was used to fulfill the main objective of the study was not used in the previous works. Both HEaan and TFHE are capable of implementing FHE. However, these libraries do not offer built-in machine learning models unlike ConcreteML. Additionally, ConcreteML is relatively easy to use as it follows the standard workflow of scikit-learn. The use of ConcreteML in the study made the overall process easier with its feature of easily converting the quantized plaintext model into an FHE model and automatically setting the optimal cryptographic security parameters through its `compile()` function. Its built-in functions on key generation, encryption, and decryption also made the use of the system easier for the client. All the client needs to do in the developed system is to upload their data and these processes are already called upon submission.

SNU and Inpher's works used only a single type of ML model in their studies. SNU used a shallow neural network and Inpher used logistic regression. Meanwhile, this study used different machine learning algorithms before deciding which model to apply in the final development of the proposed system. This allowed for a comparison to see which model worked best for the dataset. The comparison between the standard plaintext ML model and FHE ML model included in this study was also not done in the previous works. Doing this provided information on the difference between the performances of the two models which could be a great factor to consider when developing FHE ML models in the future.

Overall, the developed client-server system addresses the problem of privacy preservation in ML outsourcing. It allows for a secure tumor classification prediction tool that can help medical professionals outsource ML services without violating data privacy regulations. The use of genomic data and having them undergo encryption addressed the problem of sensitive and confidential data involved in conducting ge-

nomic machine learning studies. Lastly, the system that was built can be easily repurposed by simply changing and updating the model files applied in the server and the client files required in the client GUI.

VII. Conclusions

The proposed system in the study is a client-server system that implements FHE-based logistic regression for multi-class tumor classification. In order to achieve this, several steps were done in the development stage.

From obtaining a publicly available cancer gene expression dataset, the data went through preprocessing before being used to train three ML algorithms namely: logistic regression, random forest, and support vector classifier. For each algorithm, three models were created: scikit-learn plaintext, quantized plaintext, and finally FHE model. The performance of the models were compared by using metrics such as accuracy, balanced accuracy, and F1 score. After comparing the results, logistic regression with an accuracy of 92.3077%, balanced accuracy of 95.0000%, and F1 score of 95.7265% was selected to be the best model for system implementation.

The client-server system was developed with a GUI for the client and a web server. The GUI is Python-based and was mainly built with CustomTkinter library while Django web framework was used to build the server side of the system. The client GUI handles data preprocessing, key generation, input encryption and prediction decryption for the client. The server holds the model to be used for FHE inference once it receives an input from the client. FHE multi-class tumor classification was successfully implemented both in the client and server sides of the system. ConcreteML contributed greatly to the success of FHE implementation in machine learning with its built-in FHE functions and FHE models.

By incorporating FHE into multi-class tumor classification, the system was able to address the problem of potential data privacy risks with ML outsourcing. It also addressed the need for ensured confidentiality on genomic data which is deemed more sensitive that traditional medical data. The system developed in this study showed the potential of FHE in building privacy-preserving ML solutions in the medical industry.

VIII. Recommendations

The client-server system built with FHE-based logistic regression for secure multi-class tumor classification can be further improved in different aspects. In future studies, it is recommended to:

1. Use different methods of handling data imbalance such as resampling;
2. Apply different feature selection methods;
3. Use machine learning models that weren't tested in the study;
4. Build the system with a different genomic dataset for other tumor types;
5. Investigate cases where FHE tends to perform better than plaintext models;
6. Explore other open-source FHE libraries to implement FHE machine learning; and
7. Implement batch FHE inference.

IX. Bibliography

[1] Zama-AI, "Inference in the Cloud," Nov 2022. https://docs.zama.ai/concrete-ml/getting-started/cloud.

[2] "Tumor: What is it, types, symptoms, treatment & prevention." https://my.clevelandclinic.org/health/diseases/21881-tumor.

[3] "Three tracks of competition tasks." http://www.humangenomeprivacy.org/2020/competition-tasks.html.

[4] K. A. Bhavsar, J. Singla, Y. D. Al-Otaibi, O.-Y. Song, Y. B. Zikriya, and A. K. Bashir, "Medical Diagnosis Using Machine Learning: A Statistical Review," *Computers, Materials & Continua*, vol. 67, no. 1, p. 107–125, 2021.

[5] J. A. Diao, I. S. Kohane, and A. K. Manrai, "Biomedical Informatics and Machine Learning for Clinical Genomics," *Human Molecular Genetics*, vol. 27, no. R1, 2018.

[6] H. Abdelhalim, A. Berber, M. Lodi, R. Jain, A. Nair, A. Pappu, K. Patel, V. Venkat, C. Venkatesan, R. Wable, and et al., "Artificial Intelligence, Healthcare, Clinical Genomics, and Pharmacogenomics Approaches in Precision Medicine," *Frontiers in Genetics*, vol. 13, Jul 2022.

[7] M. Templ and M. Sariyar, "A systematic overview on methods to protect sensitive data provided for various analyses," *International Journal of Information Security*, vol. 21, no. 6, p. 1233–1246, 2022.

[8] N. Holohan, S. Antonatos, S. Braghin, and P. Mac Aonghusa, "(k, ϵ)-Anonymity: k-Anonymity with ϵ-Differential Privacy," *arXiv preprint arXiv:1710.01615*, 2017.

[9] K. Rajendran, M. Jayabalan, and M. E. Rana, "A Study on k-anonymity, l-diversity, and t-closeness Techniques," *IJCSNS*, vol. 17, no. 12, p. 172, 2017.

[10] M. Naveed, E. Ayday, E. W. Clayton, J. Fellay, C. A. Gunter, J.-P. Hubaux, B. A. Malin, and X. Wang, "Privacy in the Genomic Era," *ACM Computing Surveys (CSUR)*, vol. 48, no. 1, pp. 1–44, 2015.

[11] C. Dilmegani, "Machine Learning Outsourcing in 2022: Benefits & Challenges," Aug 2021. https://research.aimultiple.com/ml-outsourcing/.

[12] "Health Insurance Portability and Accountability Act of 1996 (HIPAA)," Jun 2022. https://www.cdc.gov/phlp/publications/topic/hipaa.html.

[13] Office of the Australian Information Commissioner, "Privacy for Health Service Providers." https://www.oaic.gov.au/privacy/privacy-for-health-service-providers.

[14] European Data Protection Supervisor, "Health." https://edps.europa.eu/data-protection/our-work/subjects/health_en.

[15] "Republic Act 10173 – Data Privacy Act of 2012," Nov 2021. https://www.privacy.gov.ph/data-privacy-act/, journal=National Privacy Commission.

[16] National eHealth Governance, "Health Privacy Code." http://ehealth.doh.gov.ph/images/HealthPrivacyCode.pdf.

[17] C. Dilmegani, "What is Homomorphic Encryption? Benefits & Challenges," Aug 2021. https://research.aimultiple.com/homomorphic-encryption/.

[18] National Cancer Institute Center for Cancer Research, "Genomic Classification of Tumors." https://ccr.cancer.gov/news/landmarks/article/genomic-classification-of-tumors.

[19] S. Quazi, "Artificial intelligence and machine learning in precision and genomic medicine," *Medical Oncology*, vol. 39, Jun 2022.

[20] J. Waring, C. Lindvall, and R. Umeton, "Automated machine learning: Review of the state-of-the-art and opportunities for healthcare," *Artificial Intelligence in Medicine*, vol. 104, p. 101822, 2020.

[21] T. T. Tang, J. A. Zawaski, K. N. Francis, A. A. Qutub, and M. W. Gaber, "Image-based classification of tumor type and growth rate using Machine Learning: A preclinical study," *Scientific Reports*, vol. 9, Aug 2019.

[22] S. Rinesh, K. Maheswari, B. Arthi, P. Sherubha, A. Vijay, S. Sridhar, T. Rajendran, and Y. A. Waji, "Investigations on brain tumor classification using hybrid machine learning algorithms," *Journal of Healthcare Engineering*, vol. 2022, Feb 2022.

[23] A. S. Assiri, S. Nazir, and S. A. Velastin, "Breast tumor classification using an ensemble machine learning method," *Journal of Imaging*, vol. 6, no. 6, p. 39, 2020.

[24] W. H. Wolberg, N. Street, and O. L. Mangasarian, "Breast Cancer Wisconsin (Diagnostic) Data Set." https://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+(diagnostic).

[25] A. Carbone, "Cancer Classification at the Crossroads," *Cancers*, vol. 12, p. 980, Apr 2020.

[26] Y. Li, K. Kang, J. M. Krahn, N. Croutwater, K. Lee, D. M. Umbach, and L. Li, "A comprehensive genomic pan-cancer classification using The Cancer Genome Atlas gene expression data," *BMC Genomics*, vol. 18, Jul 2017.

[27] J. Liu, X. Wang, Y. Cheng, and L. Zhang, "Tumor gene expression data classification via sample expansion-based deep learning," *Oncotarget*, vol. 8, p. 109646–109660, Nov 2017.

[28] M. A. Sahi, H. Abbas, K. Saleem, X. Yang, A. Derhab, M. A. Orgun, W. Iqbal, I. Rashid, and A. Yaseen, "Privacy Preservation in e-healthcare Environments: State of the Art and Future Directions," *IEEE Access*, vol. 6, p. 464–478, Feb 2018.

[29] H. Fang and Q. Qian, "Privacy preserving machine learning with homomorphic encryption and federated learning," *Future Internet*, vol. 13, p. 94, Apr 2021.

[30] S. Hong, J. H. Park, W. Cho, H. Choe, and J. H. Cheon, "Secure tumor classification by shallow neural network using homomorphic encryption," *BMC Genomics*, vol. 23, Apr 2022.

[31] M. Alloghani, M. M. Alani, D. Al-Jumeily, T. Baker, J. Mustafina, A. Hussain, and A. J. Aljaaf, "A systematic review on the status and progress of homomorphic encryption technologies," *Journal of Information Security and Applications*, vol. 48, p. 102362, Oct 2019.

[32] Ravital, "An Intro to Fully Homomorphic Encryption for Engineers," Aug 2021. https://blog.sunscreen.tech/an-intro-to-fully-homomorphic-encryption-for-engineers/.

[33] S. Mondal, M. S. Gharote, and S. P. Lodha, "Privacy of Personal Information," *ACM Queue*, vol. 20, Jul 2022.

[34] M. Phillips, "Can Genomic Data Be Anonymised?," 2018. https://www.ga4gh.org/news/can-genomic-data-be-anonymised/.

[35] Z. He, Y. Li, J. Li, K. Li, Q. Cai, and Y. Liang, "Achieving differential privacy of genomic data releasing via belief propagation," *Tsinghua Science and Technology*, vol. 23, no. 4, pp. 389–395, 2018.

[36] M. Creeger, "The Rise of Fully Homomorphic Encryption," *ACM Queue*, vol. 20, Sep 2022.

[37] H. Weng, C. Hettiarachchi, C. Nolan, H. Suominen, and A. Lenskiy, "Ensuring security of artificial pancreas device system using homomorphic encryption," *Biomedical Signal Processing and Control*, vol. 79, p. 104044, 2023.

[38] D. Arnold, J. Saniie, and A. Heifetz, "Homomorphic Encryption for Machine Learning and Artificial Intelligence applications," *Argonne National Laboratory*, Aug 2022.

[39] J.-W. Lee, H. Kang, Y. Lee, W. Choi, J. Eom, M. Deryabin, E. Lee, J. Lee, D. Yoo, Y.-S. Kim, and et al., "Privacy-Preserving Machine Learning with Fully Homomorphic Encryption for Deep Neural Network," *IEEE Access*, vol. 10, p. 30039–30054, 2022.

[40] A. Wood, K. Najarian, and D. Kahrobaei, "Homomorphic Encryption for Machine Learning in Medicine and Bioinformatics," *ACM Computing Surveys*, vol. 53, no. 4, p. 1–35, 2020.

[41] A. A. Badawi, L. Hoang, C. F. Mun, K. Laine, and K. M. Aung, "PrivFT: Private and Fast Text Classification with Homomorphic Encryption," *IEEE Access*, vol. 8, p. 226544–226556, 2020.

[42] A. Akavia, B. Galili, H. Shaul, M. Weiss, and Z. Yakhini, "Efficient privacy-preserving viral strain classification via K-Mer signatures and FHE," May 2022.

[43] "Microsoft SEAL (release 4.0)." https://github.com/Microsoft/SEAL, March 2022.

[44] F. Boemer, Y. Lao, R. Cammarota, and C. Wierzynski, "nGraph-HE: A Graph Compiler for Deep Learning on Homomorphically Encrypted Data," *Proceedings of the 16th ACM International Conference on Computing Frontiers*, 2018.

[45] F. Boemer, A. Costache, R. Cammarota, and C. Wierzynski, "nGraph-HE2: A High-Throughput Framework for Neural Network Inference on Encrypted Data," *Proceedings of the 7th ACM Workshop on Encrypted Computing & Applied Homomorphic Cryptography - WAHC'19*, 2019.

[46] E. Aharoni, A. Adir, M. Baruch, N. Drucker, G. Ezov, A. Farkash, L. Greenberg, R. Masalha, G. Moshkowich, D. Murik, H. Shaul, and O. Soceanu, "HeLayers: A Tile Tensors Framework for Large Neural Networks on Encrypted Data," *Privacy Enhancing Technology Symposium (PETs) 2023*, 2023.

[47] A. Benaissa, B. Retiat, B. Cebere, and A. E. Belfedhal, "TenSEAL: A Library for Encrypted Tensor Operations Using Homomorphic Encryption," 2021.

[48] Zama-AI, "What is Concrete ML?," Nov 2022. https://docs.zama.ai/concrete-ml.

[49] A. Meyre, B. Chevallier-Mames, J. Frery, A. Stoian, R. Bredehoft, L. Montero, and C. Kherfallah, "Concrete-ML: a privacy-preserving machine learning library using fully homomorphic encryption for data scientists," 2022-*. https://github.com/zama-ai/concrete-ml.

[50] Zama-AI, "Operations and Examples," Nov 2022. https://docs.zama.ai/concrete/getting-started/operations_and_examples.

[51] Zama-AI, "Tree-based Models," Nov 2022. https://docs.zama.ai/concrete-ml/built-in-models/tree.

[52] M. Joye, "Homomorphic Encryption 101," Dec 2021. https://www.zama.ai/post/homomorphic-encryption-101.

[53] Zama-AI, "Quantization," Nov 2022. https://docs.zama.ai/concrete-ml/advanced-topics/quantization.

[54] S. Carpov, N. Gama, M. Georgieva, and D. Jetchev, "GenoPPML–a framework for genomic privacy-preserving machine learning," in *2022 IEEE 15th International Conference on Cloud Computing (CLOUD)*, pp. 532–542, IEEE, 2022.

[55] R. S. Sagar, "Inpher's Research on Privacy-Preserving Machine Learning Published in Two Premier Conferences," July 2022. https://inpher.io/news/inphers-research-on-privacy-preserving-machine-learning-published-in-two-pre

[56] R. S. Sagar, "Inpher wins the iDASH Secure Genome Analysis Competition," Dec 2020. https://inpher.io/news/inpher-wins-the-idash-secure-genome-analysis-competition/.

[57] "Samsung SDS Wins iDASH Competition with Homomorphic Encryption Tech," Dec 2020. https://mobileidworld.com/samsung-sds-wins-idash-competition-homomorphic-encryption-tech-122206/.

[58] A. Bhattacharya, "Homomorphic Encryption - Basics," Dec 2020. https://www.encryptionconsulting.com/introduction-to-homomorphic-encryption/.

[59] M. Townend, "How Homomorphic Encryption Works & When To Use It," Nov 2022. https://www.splunk.com/en_us/blog/learn/homomorphic-encryption.html.

[60] R. A. Hallman, K. Laine, W. Dai, N. Gama, A. J. Malozemoff, Y. Polyakov, and S. Carpov, "Building Applications with Homomorphic Encryption," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pp. 2160–2162, 2018.

[61] Zama-AI, "Lifecycle of a ConcreteML Model," Nov 2022. https://docs.zama.ai/concrete-ml/getting-started/concepts.

[62] Zama-AI, "Compilation," Nov 2022. https://docs.zama.ai/concrete-ml/advanced-topics/compilation.

[63] J. Frery, A. Stoian, R. Bredehoft, L. Montero, C. Kherfallah, B. Chevallier-Mames, and A. Meyre, "Privacy-preserving tree-based inference with fully homomorphic encryption," *arXiv preprint arXiv:2303.01254*, 2023.

[64] Zama-AI, "Production Deployment," Nov 2022. https://docs.zama.ai/concrete-ml/advanced-topics/client_server.

[65] Zama-AI, "concrete.ml.deployment.fhe_client_server," Nov 2022. https://docs.zama.ai/concrete-ml/developer-guide/api/concrete.ml.deployment.fhe_client_server.

[66] Zama-AI, "Client Server in Concrete ML," Sep 2022. https://github.com/zama-ai/concrete-ml/blob/release/0.5.x/docs/advanced_examples/ClientServer.ipynb.

[67] Zama-AI, "Linear Models," Nov 2022. https://docs.zama.ai/concrete-ml/built-in-models/linear.

[68] Zama-AI, "Neural Networks," Nov 2022. https://docs.zama.ai/concrete-ml/built-in-models/neural-networks.

[69] Zama-AI, "concrete.ml.sklearn.qnn," Nov 2022. https://docs.zama.ai/concrete-ml/developer-guide/api/concrete.ml.sklearn.qnn.

[70] A. Tripathi, "What is Logistic Regression?," Jun 2019. https://towardsdatascience.com/what-is-logistic-regression-60a273e6bd91.

[71] IBM, "What is logistic regression?." https://www.ibm.com/topics/logistic-regression.

[72] A. Pradhan, S. Prabhu, K. Chadaga, S. Sengupta, and G. Nath, "Supervised learning models for the preliminary detection of covid-19 in patients using demographic and epidemiological parameters," *Information*, vol. 13, no. 7, p. 330, 2022.

[73] A. C. Chang, "Machine and deep learning," *Intelligence-based medicine: Artificial intelligence and human cognition in clinical medicine and healthcare*, pp. 67–140, 2020.

[74] B. Grisci, "Brain cancer gene expression - cumida," Feb 2020. https://www.kaggle.com/datasets/brunogrisci/brain-cancer-gene-expression-cumida.

[75] B. C. Feltes, E. B. Chandelier, B. I. Grisci, and M. Dorn, "Cumida: an extensively curated microarray database for benchmarking and testing of machine learning approaches in cancer research," *Journal of Computational Biology*, vol. 26, no. 4, pp. 376–386, 2019.

[76] "Ependymoma," Apr 2023. https://www.mayoclinic.org/diseases-conditions/ependymoma/cdc-20350144.

[77] "Glioblastoma," Jan 2023. https://www.mayoclinic.org/diseases-conditions/glioblastoma/cdc-20350148.

[78] "Medulloblastoma," Apr 2023. https://www.mayoclinic.org/diseases-conditions/medulloblastoma/cdc-20363524.

[79] "Astrocytoma," Mar 2023. https://my.clevelandclinic.org/health/diseases/17863-astrocytoma.

[80] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[81] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, *et al.*, "Api design for machine learning software: experiences from the scikit-learn project," *arXiv preprint arXiv:1309.0238*, 2013.

[82] "Classification: Accuracy." https://developers.google.com/machine-learning/crash-course/classification/accuracy.

[83] S. Mazumder, "5 Techniques to Handle Imbalanced Data for a Classification Problem," Dec 2022. https://www.analyticsvidhya.com/blog/2021/06/5-techniques-to-handle-imbalanced-data-for-a-classification-problem/.

[84] S. Allwright, "What is a good balanced accuracy score?," Aug 2022. https://stephenallwright.com/balanced-accuracy/.

[85] "Pandas documentation," May 2023. https://pandas.pydata.org/docs/#pandas-documentation.

[86] "Google colaboratory." https://colab.google/.

[87] Craigloewen-Msft, "What is the Windows Subsystem for Linux?," Aug 2022. https://learn.microsoft.com/en-us/windows/wsl/about.

[88] "tkinter - Python interface to Tcl/Tk." https://docs.python.org/3/library/tkinter.html.

[89] "Customtkinter." https://customtkinter.tomschimansky.com/.

[90] Django Software Foundation, "Django." https://djangoproject.com.

[91] "What is Bootstrap?." https://www.w3schools.com/whatis/whatis_bootstrap.asp.

[92] P. Li, F. Michel, and J. Wilson, "FHE and Machine Learning," Nov 2022. https://optalysys.com/fhe-and-machine-learning-a-student-perspective-with-examples/.

[93] Zama-AI, "Classifier Comparison," Apr 2023. https://github.com/zama-ai/concrete-ml/blob/release/1.0.x/docs/advanced_examples/ClassifierComparison.ipynb.

[94] R. Hendricks, "What is a good accuracy score in machine learning?," Nov 2022. https://deepchecks.com/question/what-is-a-good-accuracy-score-in-machine-learning/#:~:text=Industry%20standards%20are%20between%2070%25%20and%2090%25.%20Everything,as%20a%20realistic%20and%20valuable%20model%20data%20output.

[95] S. Allwright, "What is a good f1 score?," Apr 2022. https://stephenallwright.com/good-f1-score/.

[96] D. Corvoysier, "A brief introduction to Machine Learning models quantization," May 2023. https://www.kaizou.org/2023/05/machine-learning-quantization-introduction.html.

[97] Google, "Building a quantization paradigm from first principles," Jan 2019. https://github.com/google/gemmlowp/blob/master/doc/quantization.md.

[98] S. Security, "Understand Private Key and Public Key with an Example," Jan 2019. https://cheapsslsecurity.com/blog/private-key-and-public-key-explained/#:~:text=As%20such%2C%20they%20help%20encrypt%20and%20protect%20users%E2%80%99,strongest%20industry%20standard%20is%20a%202048-bit%20RSA%20key.

[99] "TLS Key Size: Why Bigger isn't Always Better," Nov 2022. https://www.fastly.com/blog/key-size-for-tls, journal=Fastly, author=Thayer, Wayne.

[100] "What is key length in cryptography and why is it important?," Oct 2022. https://justcryptography.com/key-length/, journal=Just Cryptography.

[101] N. Higham, "What is a condition number?," Mar 2020. https://nhigham.com/2020/03/19/what-is-a-condition-number/.

[102] S. Glen, "Ill-Conditioned & Condition Number." https://www.statisticshowto.com/calculus-definitions/ill-conditioned-condition-number/.

X. Appendix

A. Source Code

```
# Brain Tumor Classifier Final Training Script for Logistic Regression using Google Colaboratory

# Installing packages for concrete-ml on colab
!pip install -U pip wheel setuptools
!pip install concrete-ml


# Accessing google drive
from google.colab import drive
drive.mount('/content/gdrive')


# Importing modules
from sklearn.model_selection import train_test_split
from sklearn import preprocessing

from sklearn.metrics import accuracy_score
from sklearn.metrics import balanced_accuracy_score
from sklearn.metrics import f1_score

from concrete.ml.sklearn import LogisticRegression
from sklearn.linear_model import LogisticRegression as skLR

import time
import pandas as pd
import numpy as np
from numpy import mean
from numpy import std


# Reading the dataset
dataset = pd.read_csv("gdrive/MyDrive/Special-Problem-Colab/Brain_GSE50161.csv")
dataset.head()

feature_cols = [c for c in dataset.columns[2:]]
x = dataset.loc[:,feature_cols].values # must be floats
y = dataset.loc[:,'type'].values # must be integers


# Preprocessing with labels for the lineage
le = preprocessing.LabelEncoder()
y = le.fit_transform(y)
le_mapping = dict(zip(le.classes_, range(len(le.classes_))))
print(le_mapping)
print(le.classes_)


# Feature Selection
from sklearn.feature_selection import SelectKBest, chi2

print("\nUsing K best features feature selection...")
print("Shape of x before selection: ", x.shape)
selector = SelectKBest(chi2, k = 20)
x_new = selector.fit_transform(x, y)
x = x_new
print("Shape of x after selection: ", x.shape)
print("\n", x)


# Get most important features accorting to Kbest
cols_index = selector.get_support(indices=True)
most_important_features = []

print("\nSelected features: ")
for col in cols_index:
  most_important_features.append(str(feature_cols[col]))
print(most_important_features)


# Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    x, y, test_size=0.10)

print(f"training set size: {X_train.shape[0]}")
print(f"testing set size: {X_test.shape[0]}")


# Logistic Regression Model Training
# Scikit-learn plaintext
skmodel_LR = skLR(class_weight='balanced')
start_time = time.time()
```

```
skmodel_LR.fit(X_train,y_train)
print(f"Running time for sklearn training is {time.time() - start_time} seconds")
start_time = time.time()
y_pred_clear_LR = skmodel_LR.predict(X_test)
print(f"Running time for sklearn prediction is {time.time() - start_time} seconds")

# Quantized plaintext
quant_LR = LogisticRegression(class_weight='balanced')
start_time = time.time()
quant_LR.fit(X_train, y_train)
print(f"Running time for quantized plaintext training is {time.time() - start_time} seconds")
start_time = time.time()
y_pred_q_LR = quant_LR.predict(X_test)
print(f"Running time for quantized plaintext prediction is {time.time() - start_time} seconds")


# Metrics for scikit-learn and quantized plaintext
print("\n Logistic Regression Results \n")
# Accuracy
skLR_accuracy = accuracy_score(y_test, y_pred_clear_LR) * 100
quantLR_accuracy = accuracy_score(y_test, y_pred_q_LR) * 100
print(f"Sklearn accuracy: {skLR_accuracy:.4f}")
print(f"Quantized Clear Accuracy: {quantLR_accuracy:.4f}")
# Balanced Accuracy
skLR_bal_accuracy = balanced_accuracy_score(y_test, y_pred_clear_LR) * 100
quantLR_bal_accuracy = balanced_accuracy_score(y_test, y_pred_q_LR) * 100
print(f"Sklearn Balanced accuracy: {skLR_bal_accuracy:.4f}")
print(f"Quantized Clear Balanced Accuracy: {quantLR_bal_accuracy:.4f}")
# F1 Score
skLR_f1 = f1_score(y_test, y_pred_clear_LR, average='weighted') * 100
quantLR_f1 = f1_score(y_test, y_pred_q_LR, average='weighted') * 100
print(f"Sklearn F1 Score: {skLR_f1:.4f}")
print(f"Quantized Clear F1 Score: {quantLR_f1:.4f}")


# Logistic Regression FHE Model Compilation and Prediction on Test Set
start_time = time.time()
fhe_LR = quant_LR.compile(x)
print(f"Running time for FHE compilation is {time.time() - start_time} seconds")
start_time = time.time()
y_pred_fhe_LR = quant_LR.predict(X_test, fhe='execute')
print(f"Running time for FHE prediction is {time.time() - start_time} seconds")


# Metrics for FHE Model
print("FHE Logistic Regression Results \n")
# Accuracy
fheLR_accuracy = accuracy_score(y_test, y_pred_fhe_LR) * 100
print(f"Accuracy: {fheLR_accuracy:.4f}")
# Balanced Accuracy
fheLR_bal_accuracy = balanced_accuracy_score(y_test, y_pred_fhe_LR) * 100
print(f"Balanced accuracy: {fheLR_bal_accuracy:.4f}")
# F1 Score
fheLR_f1 = f1_score(y_test, y_pred_fhe_LR, average='weighted') * 100
print(f"F1 Score: {fheLR_f1:.4f}")


# Model prediction on test set vs Actual test set for error analysis
print("\n")
print("SKLEARN PREDICTION:\n", y_pred_clear_LR)
print("QUANTIZED CLEAR PREDICTION:\n", y_pred_q_LR)
print("FHE PREDICTION:\n", y_pred_fhe_LR)
print("ACTUAL:\n", y_test)

print("\n")
print(f"Quantized vs FHE Comparison: {int((y_pred_fhe_LR == y_pred_q_LR).sum()/len(y_pred_fhe_LR)*100)}% similar"
    )
print(f"Sklearn vs FHE Comparison: {int((y_pred_fhe_LR == y_pred_clear_LR).sum()/len(y_pred_fhe_LR)*100)}%
    similar")


# Error analysis using confusion matrix
start_time = time.time()
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

print("***Note: The diagonal elements are the correctly predicted samples. ***")

print("Confusion matrix for SKLearn Plaintext: ")
sklearn_cm_display = ConfusionMatrixDisplay(confusion_matrix(y_test, y_pred_clear_LR), display_labels=le.classes_
    )
sklearn_cm_display.plot()
plt.show()

print("Confusion matrix for Quantized Plaintext: ")
concrete_plain_display = ConfusionMatrixDisplay(confusion_matrix(y_test, y_pred_q_LR), display_labels=le.classes_
    )
concrete_plain_display.plot()
plt.show()

print("Confusion matrix for FHE: ")
concrete_fhe_display = ConfusionMatrixDisplay(confusion_matrix(y_test, y_pred_fhe_LR), display_labels=le.classes_
```

```
    )
concrete_fhe_display.plot()
plt.show()

print(f"Running time is {time.time() - start_time} seconds")


# Obtaining one sample per class
ependymoma_sample = dataset[dataset['samples'] == 879][most_important_features].to_numpy(dtype="uint16")
glioblastoma_sample = dataset[dataset['samples'] == 913][most_important_features].to_numpy(dtype="uint16")
medulloblastoma_sample = dataset[dataset['samples'] == 935][most_important_features].to_numpy(dtype="uint16")
normal_sample = dataset[dataset['samples'] == 948][most_important_features].to_numpy(dtype="uint16")
pilocytic_astrocytoma_sample = dataset[dataset['samples'] == 963][most_important_features].to_numpy(dtype="uint16")


# Sklearn Inference Time
average = 0

start_time = time.time()
skmodel_LR.predict(ependymoma_sample)
end_time = time.time()
print(f"Running time for Sklearn inference of ependymoma_sample is {end_time - start_time} seconds")

average += end_time - start_time

start_time = time.time()
skmodel_LR.predict(glioblastoma_sample)
end_time = time.time()
print(f"Running time for Sklearn inference of glioblastoma_sample is {end_time - start_time} seconds")

average += end_time - start_time

start_time = time.time()
skmodel_LR.predict(medulloblastoma_sample)
end_time = time.time()
print(f"Running time for Sklearn inference of medulloblastoma_sample is {end_time - start_time} seconds")

average += end_time - start_time

start_time = time.time()
skmodel_LR.predict(normal_sample)
end_time = time.time()
print(f"Running time for Sklearn inference of normal_sample is {end_time - start_time} seconds")

average += end_time - start_time

start_time = time.time()
skmodel_LR.predict(pilocytic_astrocytoma_sample)
end_time = time.time()
print(f"Running time for Sklearn inference of pilocytic_astrocytoma_sample is {end_time - start_time} seconds")

average += end_time - start_time

average /= 5

print(f"Average running time of Sklearn inference for each class is {average} seconds")


# Quantized Plaintext Inference Time
average = 0

start_time = time.time()
quant_LR.predict(ependymoma_sample)
end_time = time.time()
print(f"Running time for quantized plaintext inference of ependymoma_sample is {end_time - start_time} seconds")

average += end_time - start_time

start_time = time.time()
quant_LR.predict(glioblastoma_sample)
end_time = time.time()
print(f"Running time for quantized plaintext inference of glioblastoma_sample is {end_time - start_time} seconds")

average += end_time - start_time

start_time = time.time()
quant_LR.predict(medulloblastoma_sample)
end_time = time.time()
print(f"Running time for quantized plaintext inference of medulloblastoma_sample is {end_time - start_time} seconds")

average += end_time - start_time

start_time = time.time()
quant_LR.predict(normal_sample)
end_time = time.time()
print(f"Running time for quantized plaintext inference of normal_sample is {end_time - start_time} seconds")

average += end_time - start_time

```

```
start_time = time.time()
quant_LR.predict(pilocytic_astrocytoma_sample)
end_time = time.time()
print(f"Running time for quantized plaintext inference of pilocytic_astrocytoma_sample is {end_time - start_time}
      seconds")

average += end_time - start_time

average /= 5

print(f"Average running time of quantized plaintext inference for each class is {average} seconds")


# FHE Inference Time
average = 0

start_time = time.time()
quant_LR.predict(ependymoma_sample, fhe="execute")
end_time = time.time()
print(f"Running time for FHE inference of ependymoma_sample is {end_time - start_time} seconds")

average += end_time - start_time

start_time = time.time()
quant_LR.predict(glioblastoma_sample, fhe="execute")
end_time = time.time()
print(f"Running time for FHE inference of glioblastoma_sample is {end_time - start_time} seconds")

average += end_time - start_time

start_time = time.time()
quant_LR.predict(medulloblastoma_sample, fhe="execute")
end_time = time.time()
print(f"Running time for FHE inference of medulloblastoma_sample is {end_time - start_time} seconds")

average += end_time - start_time

start_time = time.time()
quant_LR.predict(normal_sample, fhe="execute")
end_time = time.time()
print(f"Running time for FHE inference of normal_sample is {end_time - start_time} seconds")

average += end_time - start_time

start_time = time.time()
quant_LR.predict(pilocytic_astrocytoma_sample, fhe="execute")
end_time = time.time()
print(f"Running time for FHE inference of pilocytic_astrocytoma_sample is {end_time - start_time} seconds")

average += end_time - start_time

average /= 5

print(f"Average running time of FHE inference for each class is {average} seconds")


# Saving the model into desired directory/path
from concrete.ml.deployment import FHEModelClient, FHEModelDev, FHEModelServer

start_time = time.time()

fhemodel_dev = FHEModelDev("gdrive/MyDrive/Special-Problem-Colab/Brain-Tumor-Models/", quant_LR)
fhemodel_dev.save()

print(f"Running time for saving the FHE model is {time.time() - start_time} seconds")


# Saving the selected features and the classes into  text file
import json

for col in cols_index:
  print(feature_cols[col])

for classLabel in le.classes_:
  print(classLabel)

with open("features_and_classes.txt", "w") as f:
    classes_list = list(le.classes_)
    temp_dict = {"features":[feature_cols[col] for col in cols_index], "classes":{classes_list.index(x):x for x
     in classes_list}}

    f.write(json.dumps(temp_dict))
```

Listing 1: Logistic regression model training (logisticRegression-training.py).

```
# Brain Tumor Classifier Final Training Script for Random Forest using Google Colaboratory

# Installing packages for concrete-ml on colab
!pip install -U pip wheel setuptools
!pip install concrete-ml
```

```
6
7
8  # Accessing google drive
9  from google.colab import drive
10 drive.mount('/content/gdrive')
11
12
13 # Importing modules
14 from sklearn.model_selection import train_test_split
15 from sklearn import preprocessing
16
17 from sklearn.metrics import accuracy_score
18 from sklearn.metrics import balanced_accuracy_score
19 from sklearn.metrics import f1_score
20
21 from concrete.ml.sklearn.rf import RandomForestClassifier
22 from sklearn.ensemble import RandomForestClassifier as skRF
23
24 import time
25 import pandas as pd
26 import numpy as np
27 from numpy import mean
28 from numpy import std
29
30
31 # Reading the dataset
32 dataset = pd.read_csv("gdrive/MyDrive/Special-Problem-Colab/Brain_GSE50161.csv")
33 dataset.head()
34
35 feature_cols = [c for c in dataset.columns[2:]]
36 x = dataset.loc[:,feature_cols].values # must be floats
37 y = dataset.loc[:,'type'].values # must be integers
38
39
40 # Preprocessing with labels for the lineage
41 le = preprocessing.LabelEncoder()
42 y = le.fit_transform(y)
43 le_mapping = dict(zip(le.classes_, range(len(le.classes_))))
44 print(le_mapping)
45 print(le.classes_)
46
47
48 # Feature Selection
49 from sklearn.feature_selection import SelectKBest, chi2
50
51 print("\nUsing K best features feature selection...")
52 print("Shape of x before selection: ", x.shape)
53 selector = SelectKBest(chi2, k = 20)
54 x_new = selector.fit_transform(x, y)
55 x = x_new
56 print("Shape of x after selection: ", x.shape)
57 print("\n", x)
58
59
60 # Get most important features accorting to Kbest
61 cols_index = selector.get_support(indices=True)
62 most_important_features = []
63
64 print("\nSelected features: ")
65 for col in cols_index:
66   most_important_features.append(str(feature_cols[col]))
67 print(most_important_features)
68
69
70 # Train-test split
71 X_train, X_test, y_train, y_test = train_test_split(
72     x, y, test_size=0.10)
73
74 print(f"training set size: {X_train.shape[0]}")
75 print(f"testing set size: {X_test.shape[0]}")
76
77
78 # Random Forest Model Training
79 # Scikit-learn plaintext
80 skmodel_RF = skRF(class_weight='balanced')
81 start_time = time.time()
82 skmodel_RF.fit(X_train,y_train)
83 print(f"Running time for sklearn training is {time.time() - start_time} seconds")
84 start_time = time.time()
85 y_pred_clear_RF = skmodel_RF.predict(X_test)
86 print(f"Running time for sklearn prediction is {time.time() - start_time} seconds")
87
88 # Quantized plaintext
89 quant_RF = RandomForestClassifier(class_weight='balanced')
90 start_time = time.time()
91 quant_RF.fit(X_train, y_train)
92 print(f"Running time for quantized plaintext training is {time.time() - start_time} seconds")
93 start_time = time.time()
94 y_pred_q_RF = quant_RF.predict(X_test)
95 print(f"Running time for quantized plaintext prediction is {time.time() - start_time} seconds")
96
97
```

```
# Metrics for scikit-learn and quantized plaintext
print("\n Random Forest Results \n")
# Accuracy
skRF_accuracy = accuracy_score(y_test, y_pred_clear_RF) * 100
quantRF_accuracy = accuracy_score(y_test, y_pred_q_RF) * 100
print(f"Sklearn accuracy: {skRF_accuracy:.4f}")
print(f"Quantized Clear Accuracy: {quantRF_accuracy:.4f}")
# Balanced Accuracy
skRF_bal_accuracy = balanced_accuracy_score(y_test, y_pred_clear_RF) * 100
quantRF_bal_accuracy = balanced_accuracy_score(y_test, y_pred_q_RF) * 100
print(f"Sklearn Balanced accuracy: {skRF_bal_accuracy:.4f}")
print(f"Quantized Clear Balanced Accuracy: {quantRF_bal_accuracy:.4f}")
# F1 Score
skRF_f1 = f1_score(y_test, y_pred_clear_RF, average='weighted') * 100
quantRF_f1 = f1_score(y_test, y_pred_q_RF, average='weighted') * 100
print(f"Sklearn F1 Score: {skRF_f1:.4f}")
print(f"Quantized Clear F1 Score: {quantRF_f1:.4f}")


# Random Forest FHE Model Compilation and Prediction on Test Set
start_time = time.time()
fhe_RF = quant_RF.compile(x)
print(f"Running time for FHE compilation is {time.time() - start_time} seconds")
start_time = time.time()
y_pred_fhe_RF = quant_RF.predict(X_test, fhe="execute")
print(f"Running time for FHE prediction is {time.time() - start_time} seconds")


# Metrics for FHE Model
print("FHE Random Forest Results \n")
# Accuracy
fheRF_accuracy = accuracy_score(y_test, y_pred_fhe_RF) * 100
print(f"Accuracy: {fheRF_accuracy:.4f}")
# Balanced Accuracy
fheRF_bal_accuracy = balanced_accuracy_score(y_test, y_pred_fhe_RF) * 100
print(f"Balanced accuracy: {fheRF_bal_accuracy:.4f}")
# F1 Score
fheRF_f1 = f1_score(y_test, y_pred_fhe_RF, average='weighted') * 100
print(f"F1 Score: {fheRF_f1:.4f}")


# Model prediction on test set vs Actual test set
print("\n")
print("SKLEARN PREDICTION:\n", y_pred_clear_RF)
print("QUANTIZED CLEAR PREDICTION:\n", y_pred_q_RF)
print("FHE PREDICTION:\n", y_pred_fhe_RF)
print("ACTUAL:\n", y_test)

print("\n")
print(f"Quantized vs FHE Comparison: {int((y_pred_fhe_RF == y_pred_q_RF).sum()/len(y_pred_fhe_RF)*100)}% similar"
    )
print(f"Sklearn vs FHE Comparison: {int((y_pred_fhe_RF == y_pred_clear_RF).sum()/len(y_pred_fhe_RF)*100)}%
    similar")


# Obtaining one sample per class
ependymoma_sample = dataset[dataset['samples'] == 879][most_important_features].to_numpy(dtype="uint16")
glioblastoma_sample = dataset[dataset['samples'] == 913][most_important_features].to_numpy(dtype="uint16")
medulloblastoma_sample = dataset[dataset['samples'] == 935][most_important_features].to_numpy(dtype="uint16")
normal_sample = dataset[dataset['samples'] == 948][most_important_features].to_numpy(dtype="uint16")
pilocytic_astrocytoma_sample = dataset[dataset['samples'] == 963][most_important_features].to_numpy(dtype="uint16
    ")


# Sklearn Inference Time
average = 0

start_time = time.time()
skmodel_RF.predict(ependymoma_sample)
end_time = time.time()
print(f"Running time for Sklearn inference of ependymoma_sample is {end_time - start_time} seconds")

average += end_time - start_time

start_time = time.time()
skmodel_RF.predict(glioblastoma_sample)
end_time = time.time()
print(f"Running time for Sklearn inference of glioblastoma_sample is {end_time - start_time} seconds")

average += end_time - start_time

start_time = time.time()
skmodel_RF.predict(medulloblastoma_sample)
end_time = time.time()
print(f"Running time for Sklearn inference of medulloblastoma_sample is {end_time - start_time} seconds")

average += end_time - start_time

start_time = time.time()
skmodel_RF.predict(normal_sample)
end_time = time.time()
print(f"Running time for Sklearn inference of normal_sample is {end_time - start_time} seconds")
```

```
187
188 average += end_time - start_time
189
190 start_time = time.time()
191 skmodel_RF.predict(pilocytic_astrocytoma_sample)
192 end_time = time.time()
193 print(f"Running time for Sklearn inference of pilocytic_astrocytoma_sample is {end_time - start_time} seconds")
194
195 average += end_time - start_time
196
197 average /= 5
198
199 print(f"Average running time of Sklearn inference for each class is {average} seconds")
200
201
202 # Quantized Plaintext Inference Time
203 average = 0
204
205 start_time = time.time()
206 quant_RF.predict(ependymoma_sample)
207 end_time = time.time()
208 print(f"Running time for quantized plaintext inference of ependymoma_sample is {end_time - start_time} seconds")
209
210 average += end_time - start_time
211
212 start_time = time.time()
213 quant_RF.predict(glioblastoma_sample)
214 end_time = time.time()
215 print(f"Running time for quantized plaintext inference of glioblastoma_sample is {end_time - start_time} seconds"
      )
216
217 average += end_time - start_time
218
219 start_time = time.time()
220 quant_RF.predict(medulloblastoma_sample)
221 end_time = time.time()
222 print(f"Running time for quantized plaintext inference of medulloblastoma_sample is {end_time - start_time}
      seconds")
223
224 average += end_time - start_time
225
226 start_time = time.time()
227 quant_RF.predict(normal_sample)
228 end_time = time.time()
229 print(f"Running time for quantized plaintext inference of normal_sample is {end_time - start_time} seconds")
230
231 average += end_time - start_time
232
233 start_time = time.time()
234 quant_RF.predict(pilocytic_astrocytoma_sample)
235 end_time = time.time()
236 print(f"Running time for quantized plaintext inference of pilocytic_astrocytoma_sample is {end_time - start_time}
       seconds")
237
238 average += end_time - start_time
239
240 average /= 5
241
242 print(f"Average running time of quantized plaintext inference for each class is {average} seconds")
243
244
245 # FHE Inference Time
246 average = 0
247
248 start_time = time.time()
249 quant_RF.predict(ependymoma_sample, fhe="execute")
250 end_time = time.time()
251 print(f"Running time for FHE inference of ependymoma_sample is {end_time - start_time} seconds")
252
253 average += end_time - start_time
254
255 start_time = time.time()
256 quant_RF.predict(glioblastoma_sample, fhe="execute")
257 end_time = time.time()
258 print(f"Running time for FHE inference of glioblastoma_sample is {end_time - start_time} seconds")
259
260 average += end_time - start_time
261
262 start_time = time.time()
263 quant_RF.predict(medulloblastoma_sample, fhe="execute")
264 end_time = time.time()
265 print(f"Running time for FHE inference of medulloblastoma_sample is {end_time - start_time} seconds")
266
267 average += end_time - start_time
268
269 start_time = time.time()
270 quant_RF.predict(normal_sample, fhe="execute")
271 end_time = time.time()
272 print(f"Running time for FHE inference of normal_sample is {end_time - start_time} seconds")
273
274 average += end_time - start_time
275
```

```
start_time = time.time()
quant_RF.predict(pilocytic_astrocytoma_sample, fhe="execute")
end_time = time.time()
print(f"Running time for FHE inference of pilocytic_astrocytoma_sample is {end_time - start_time} seconds")

average += end_time - start_time

average /= 5

print(f"Average running time of FHE inference for each class is {average} seconds")


# Saving the model into desired directory/path
from concrete.ml.deployment import FHEModelClient, FHEModelDev, FHEModelServer

start_time = time.time()

fhemodel_dev = FHEModelDev("gdrive/MyDrive/Special-Problem-Colab/Brain-Tumor-Models/", quant_RF)
fhemodel_dev.save()

print(f"Running time for saving the FHE model is {time.time() - start_time} seconds")


# Saving the selected features and the classes into  text file
import json

for col in cols_index:
  print(feature_cols[col])

for classLabel in le.classes_:
  print(classLabel)

with open("features_and_classes.txt", "w") as f:
    classes_list = list(le.classes_)
    temp_dict = {"features":[feature_cols[col] for col in cols_index], "classes":{classes_list.index(x):x for x
     in classes_list}}

    f.write(json.dumps(temp_dict))
```

Listing 2: Random forest model training (randomForest-training.py).

```
# Brain Tumor Classifier Final Training Script for Linear SVC using Google Colaboratory

# Installing packages for concrete-ml on colab
!pip install -U pip wheel setuptools
!pip install concrete-ml


# Accessing google drive
from google.colab import drive
drive.mount('/content/gdrive')


# Importing modules
from sklearn.model_selection import train_test_split
from sklearn import preprocessing

from sklearn.metrics import accuracy_score
from sklearn.metrics import balanced_accuracy_score
from sklearn.metrics import f1_score

from concrete.ml.sklearn.svm import LinearSVC
from sklearn.svm import LinearSVC as skSVC

import time
import pandas as pd
import numpy as np
from numpy import mean
from numpy import std


# Reading the dataset
dataset = pd.read_csv("gdrive/MyDrive/Special-Problem-Colab/Brain_GSE50161.csv")
dataset.head()

feature_cols = [c for c in dataset.columns[2:]]
x = dataset.loc[:,feature_cols].values # must be floats
y = dataset.loc[:,'type'].values # must be integers


# Preprocessing with labels for the lineage
le = preprocessing.LabelEncoder()
y = le.fit_transform(y)
le_mapping = dict(zip(le.classes_, range(len(le.classes_))))
print(le_mapping)
print(le.classes_)


# Feature Selection
from sklearn.feature_selection import SelectKBest, chi2
```

```
50
51 print("\nUsing K best features feature selection...")
52 print("Shape of x before selection: ", x.shape)
53 selector = SelectKBest(chi2, k = 20)
54 x_new = selector.fit_transform(x, y)
55 x = x_new
56 print("Shape of x after selection: ", x.shape)
57 print("\n", x)
58
59
60 # Get most important features accorting to Kbest
61 cols_index = selector.get_support(indices=True)
62 most_important_features = []
63
64 print("\nSelected features: ")
65 for col in cols_index:
66   most_important_features.append(str(feature_cols[col]))
67 print(most_important_features)
68
69
70 # Train-test split
71 X_train, X_test, y_train, y_test = train_test_split(
72     x, y, test_size=0.10)
73
74 print(f"training set size: {X_train.shape[0]}")
75 print(f"testing set size: {X_test.shape[0]}")
76
77
78 # Linear SVC Model Training
79 # Scikit-learn plaintext
80 skmodel_SVC = skSVC(class_weight='balanced')
81 start_time = time.time()
82 skmodel_SVC.fit(X_train,y_train)
83 print(f"Running time for sklearn training is {time.time() - start_time} seconds")
84 start_time = time.time()
85 y_pred_clear_SVC = skmodel_SVC.predict(X_test)
86 print(f"Running time for sklearn prediction is {time.time() - start_time} seconds")
87
88 # Quantized plaintext
89 quant_SVC = LinearSVC(class_weight='balanced')
90 start_time = time.time()
91 quant_SVC.fit(X_train, y_train)
92 print(f"Running time for quantized plaintext training is {time.time() - start_time} seconds")
93 start_time = time.time()
94 y_pred_q_SVC = quant_SVC.predict(X_test)
95 print(f"Running time for quantized plaintext prediction is {time.time() - start_time} seconds")
96
97
98 # Metrics for scikit-learn and quantized plaintext
99 print("\n Linear SVC Results \n")
100 # Accuracy
101 skSVC_accuracy = accuracy_score(y_test, y_pred_clear_SVC) * 100
102 quantSVC_accuracy = accuracy_score(y_test, y_pred_q_SVC) * 100
103 print(f"Sklearn accuracy: {skSVC_accuracy:.4f}")
104 print(f"Quantized Clear Accuracy: {quantSVC_accuracy:.4f}")
105 # Balanced Accuracy
106 skSVC_bal_accuracy = balanced_accuracy_score(y_test, y_pred_clear_SVC) * 100
107 quantSVC_bal_accuracy = balanced_accuracy_score(y_test, y_pred_q_SVC) * 100
108 print(f"Sklearn Balanced accuracy: {skSVC_bal_accuracy:.4f}")
109 print(f"Quantized Clear Balanced Accuracy: {quantSVC_bal_accuracy:.4f}")
110 # F1 Score
111 skSVC_f1 = f1_score(y_test, y_pred_clear_SVC, average='weighted') * 100
112 quantSVC_f1 = f1_score(y_test, y_pred_q_SVC, average='weighted') * 100
113 print(f"Sklearn F1 Score: {skSVC_f1:.4f}")
114 print(f"Quantized Clear F1 Score: {quantSVC_f1:.4f}")
115
116
117 # Linear SVC FHE Model Compilation and Prediction on Test Set
118 start_time = time.time()
119 fhe_SVC = quant_SVC.compile(x)
120 print(f"Running time for FHE compilation is {time.time() - start_time} seconds")
121 start_time = time.time()
122 y_pred_fhe_SVC = quant_SVC.predict(X_test, fhe="execute")
123 print(f"Running time for FHE prediction is {time.time() - start_time} seconds")
124
125
126 # Metrics for FHE Model
127 print("FHE Linear SVC Results \n")
128 # Accuracy
129 fheSVC_accuracy = accuracy_score(y_test, y_pred_fhe_SVC) * 100
130 print(f"Accuracy: {fheSVC_accuracy:.4f}")
131 # Balanced Accuracy
132 fheSVC_bal_accuracy = balanced_accuracy_score(y_test, y_pred_fhe_SVC) * 100
133 print(f"Balanced accuracy: {fheSVC_bal_accuracy:.4f}")
134 # F1 Score
135 fheRF_f1 = f1_score(y_test, y_pred_fhe_SVC, average='weighted') * 100
136 print(f"F1 Score: {fheRF_f1:.4f}")
137
138
139 # Model prediction on test set vs Actual test set
140 print("\n")
141 print("SKLEARN PREDICTION:\n", y_pred_clear_SVC)
```

```
print("QUANTIZED CLEAR PREDICTION:\n", y_pred_q_SVC)
print("FHE PREDICTION:\n", y_pred_fhe_SVC)
print("ACTUAL:\n", y_test)

print("\n")
print(f"Quantized vs FHE Comparison: {int((y_pred_fhe_SVC == y_pred_q_SVC).sum()/len(y_pred_fhe_SVC)*100)}% similar")
print(f"Sklearn vs FHE Comparison: {int((y_pred_fhe_SVC == y_pred_clear_SVC).sum()/len(y_pred_fhe_SVC)*100)}% similar")


# Obtaining one sample per class
ependymoma_sample = dataset[dataset['samples'] == 879][most_important_features].to_numpy(dtype="uint16")
glioblastoma_sample = dataset[dataset['samples'] == 913][most_important_features].to_numpy(dtype="uint16")
medulloblastoma_sample = dataset[dataset['samples'] == 935][most_important_features].to_numpy(dtype="uint16")
normal_sample = dataset[dataset['samples'] == 948][most_important_features].to_numpy(dtype="uint16")
pilocytic_astrocytoma_sample = dataset[dataset['samples'] == 963][most_important_features].to_numpy(dtype="uint16")


# Sklearn Inference Time
average = 0

start_time = time.time()
skmodel_SVC.predict(ependymoma_sample)
end_time = time.time()
print(f"Running time for Sklearn inference of ependymoma_sample is {end_time - start_time} seconds")

average += end_time - start_time

start_time = time.time()
skmodel_SVC.predict(glioblastoma_sample)
end_time = time.time()
print(f"Running time for Sklearn inference of glioblastoma_sample is {end_time - start_time} seconds")

average += end_time - start_time

start_time = time.time()
skmodel_SVC.predict(medulloblastoma_sample)
end_time = time.time()
print(f"Running time for Sklearn inference of medulloblastoma_sample is {end_time - start_time} seconds")

average += end_time - start_time

start_time = time.time()
skmodel_SVC.predict(normal_sample)
end_time = time.time()
print(f"Running time for Sklearn inference of normal_sample is {end_time - start_time} seconds")

average += end_time - start_time

start_time = time.time()
skmodel_SVC.predict(pilocytic_astrocytoma_sample)
end_time = time.time()
print(f"Running time for Sklearn inference of pilocytic_astrocytoma_sample is {end_time - start_time} seconds")

average += end_time - start_time

average /= 5

print(f"Average running time of Sklearn inference for each class is {average} seconds")


# Quantized Plaintext Inference Time
average = 0

start_time = time.time()
quant_SVC.predict(ependymoma_sample)
end_time = time.time()
print(f"Running time for quantized plaintext inference of ependymoma_sample is {end_time - start_time} seconds")

average += end_time - start_time

start_time = time.time()
quant_SVC.predict(glioblastoma_sample)
end_time = time.time()
print(f"Running time for quantized plaintext inference of glioblastoma_sample is {end_time - start_time} seconds")

average += end_time - start_time

start_time = time.time()
quant_SVC.predict(medulloblastoma_sample)
end_time = time.time()
print(f"Running time for quantized plaintext inference of medulloblastoma_sample is {end_time - start_time} seconds")

average += end_time - start_time

start_time = time.time()
quant_SVC.predict(normal_sample)
end_time = time.time()
```

```
print(f"Running time for quantized plaintext inference of normal_sample is {end_time - start_time} seconds")

average += end_time - start_time

start_time = time.time()
quant_SVC.predict(pilocytic_astrocytoma_sample)
end_time = time.time()
print(f"Running time for quantized plaintext inference of pilocytic_astrocytoma_sample is {end_time - start_time}
      seconds")

average += end_time - start_time

average /= 5

print(f"Average running time of quantized plaintext inference for each class is {average} seconds")


# FHE Inference Time
average = 0

start_time = time.time()
quant_SVC.predict(ependymoma_sample, fhe="execute")
end_time = time.time()
print(f"Running time for FHE inference of ependymoma_sample is {end_time - start_time} seconds")

average += end_time - start_time

start_time = time.time()
quant_SVC.predict(glioblastoma_sample, fhe="execute")
end_time = time.time()
print(f"Running time for FHE inference of glioblastoma_sample is {end_time - start_time} seconds")

average += end_time - start_time

start_time = time.time()
quant_SVC.predict(medulloblastoma_sample, fhe="execute")
end_time = time.time()
print(f"Running time for FHE inference of medulloblastoma_sample is {end_time - start_time} seconds")

average += end_time - start_time

start_time = time.time()
quant_SVC.predict(normal_sample, fhe="execute")
end_time = time.time()
print(f"Running time for FHE inference of normal_sample is {end_time - start_time} seconds")

average += end_time - start_time

start_time = time.time()
quant_SVC.predict(pilocytic_astrocytoma_sample, fhe="execute")
end_time = time.time()
print(f"Running time for FHE inference of pilocytic_astrocytoma_sample is {end_time - start_time} seconds")

average += end_time - start_time

average /= 5

print(f"Average running time of FHE inference for each class is {average} seconds")


# Saving the model into desired directory/path
from concrete.ml.deployment import FHEModelClient, FHEModelDev, FHEModelServer

start_time = time.time()

fhemodel_dev = FHEModelDev("gdrive/MyDrive/Special-Problem-Colab/Brain-Tumor-Models/", quant_SVC)
fhemodel_dev.save()

print(f"Running time for saving the FHE model is {time.time() - start_time} seconds")


# Saving the selected features and the classes into  text file
import json

for col in cols_index:
  print(feature_cols[col])

for classLabel in le.classes_:
  print(classLabel)

with open("features_and_classes.txt", "w") as f:
    classes_list = list(le.classes_)
    temp_dict = {"features":[feature_cols[col] for col in cols_index], "classes":{classes_list.index(x):x for x
     in classes_list}}

    f.write(json.dumps(temp_dict))
```

Listing 3: Linear SVC model training (linearSVC-training.py).

```
from django.shortcuts import render, HttpResponse, HttpResponseRedirect
```

```
from django.http import HttpResponse, FileResponse
from web_project.settings import BASE_DIR
import os, io, zipfile, requests, shutil, subprocess
from pandas import DataFrame as pd
from pandas import read_csv
from concrete.ml.deployment import FHEModelServer

### Loading the home page
def index(request):
    return render(request, 'index.html',
                  context = {'classes_list':{0: 'ependymoma', 1: 'glioblastoma', 2: 'medulloblastoma', 3: 'normal
     ', 4: 'pilocytic_astrocytoma'}}
                  )


### Perform FHE inference
def start_classification(request):

    clean_predictions_folder()

    count = 0
    model_path =os.path.join(BASE_DIR, "FHE-Compiled-Model/")
    keys_path = os.path.join(BASE_DIR, "tumorClassifier/keys")
    keys_file = request.FILES['keys_file']
    pred_dir = os.path.join(BASE_DIR, "tumorClassifier/predictions")

    data = request.FILES['inputs'].read().strip()
    # print(f"Data received from client is {data[:200]}")

    enc_file_list = []

    count += 1
    serialized_evaluation_keys = keys_file.read()
    encrypted_prediction = FHEModelServer(model_path).run(data, serialized_evaluation_keys)
    pred_file_name = f"encrypted_prediction_{count}.enc"
    pred_file_path = os.path.join(pred_dir, pred_file_name)
    with open(pred_file_path, "wb") as f:
        f.write(encrypted_prediction)

    # Send prediction to client as a zip file
    enc_file_list.append(pred_file_path)
    zipfile = create_zip(enc_file_list)

    return zipfile


### Creating a zip file
def create_zip(file_list):
    count = 0
    zip_filename = os.path.join(BASE_DIR, "tumorClassifier/predictions/enc_predictions.zip")
    zip_download_name = "enc_predictions.zip"
    buffer = io.BytesIO()
    zip_file = zipfile.ZipFile(buffer, 'w')

    for filename in file_list:
        count += 1
        with open(filename, "rb") as file_read:
            zip_file.write(filename, f"encrypted_prediction_{count}.enc")
    zip_file.close()

    # Craft download response
    resp = HttpResponse(buffer.getvalue(), content_type = "application/force-download")
    resp['Content-Disposition'] = f'attachment; filename={zip_download_name}'

    return resp


def clean_predictions_folder():
    pred_dir = os.path.join(BASE_DIR, f"tumorClassifier/predictions")

    if(os.listdir(pred_dir)):
        for f in os.listdir(pred_dir):
            os.remove(os.path.join(pred_dir, f))
```

Listing 4: Server functionalities (views.py).

```
from django.urls import path
from tumorClassifier import views

urlpatterns = [
    path("", views.index, name="home"),
    path("index", views.index, name="index"),
    path("start_classification", views.start_classification, name="start_classification"),
]
```

Listing 5: Server URL settings (urls.py).

```

```

```
<!DOCTYPE html>
<html lang="en">
<head>
    {% load static %}


    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
    <title>FHE Brain Tumor Classifier</title>
    <link rel="icon" href="{% static 'img/favicon.png' %}">
    <!-- Bootstrap CSS -->
    <link rel="stylesheet" href="{% static 'css/bootstrap.min.css' %}">
    <!-- animate CSS -->
    <link rel="stylesheet" href="{% static 'css/animate.css' %}">
    <!-- owl carousel CSS -->
    <link rel="stylesheet" href="{% static 'css/owl.carousel.min.css' %}">
    <!-- themify CSS -->
    <link rel="stylesheet" href="{% static 'css/themify-icons.css' %}">
    <!-- flaticon CSS -->
    <link rel="stylesheet" href="{% static 'css/flaticon.css' %}">
    <!-- magnific popup CSS -->
    <link rel="stylesheet" href="{% static 'css/magnific-popup.css' %}">
    <!-- nice select CSS -->
    <link rel="stylesheet" href="{% static 'css/nice-select.css' %}">
    <!-- swiper CSS -->
    <link rel="stylesheet" href="{% static 'css/slick.css' %}">
    <!-- style CSS -->
    <link rel="stylesheet" href="{% static 'css/style.css' %}">

</head>

<body>
    <!--::header part start::-->
    <header class="main_menu home_menu">
        <div class="container">
            <div class="row align-items-center">
                <div class="col-lg-12">
                    <nav class="navbar navbar-expand-lg navbar-light">
                        <h2 class="navbar-brand" href="#"></h2>
                        <button class="navbar-toggler" type="button" data-toggle="collapse"
                            data-target="#navbarSupportedContent" aria-controls="navbarSupportedContent"
                            aria-expanded="false" aria-label="Toggle navigation">
                            <span class="navbar-toggler-icon"></span>
                        </button>

                        <div class="collapse navbar-collapse main-menu-item justify-content-center"
                            id="navbarSupportedContent">
                            <ul class="navbar-nav align-items-left">
                                <li class="nav-item active">
                                    <a class="nav-link" href="{% url 'index' %}" style="font-weight: bold;">Home<
 /a>
                                </li>
                                <li class="nav-item">
                                    <a class="nav-link" href="https://github.com/gcrosario/concreteML-FHE-Tumor-
 Classification" target="_blank" style="font-weight: bold;">Github</a>
                                </li>
                            </ul>
                        </div>
                    </nav>
                </div>
            </div>
        </div>
    </header>
    <!-- Header part end-->

    {%block content%}
    {%endblock%}

    <!-- footer part start-->
    <footer class="footer-area">
        <div class="copyright_part">
            <div class="container">
                <div class="row align-items-center">
                    <p class="footer-text m-0 col-lg-8 col-md-12">By Gwyneth Rose C. Rosario (June 2023)</p>
                </div>
            </div>
        </div>
    </footer>

    <!-- footer part end-->

    <script src="{% static 'js/jquery-1.12.1.min.js' %}"></script>
    <!-- popper js -->
    <script src="{% static 'js/popper.min.js' %}"></script>
    <!-- bootstrap js -->
    <script src="{% static 'js/bootstrap.min.js' %}"></script>
    <!-- owl carousel js -->
    <script src="{% static 'js/owl.carousel.min.js' %}"></script>
    <script src="{% static 'js/jquery.nice-select.min.js' %}"></script>
    <!-- contact js -->
    <script src="{% static 'js/jquery.ajaxchimp.min.js' %}"></script>
```

```
    <script src="{% static 'js/jquery.form.js' %}"></script>
    <script src="{% static 'js/jquery.validate.min.js' %}"></script>
    <script src="{% static 'js/mail-script.js' %}"></script>
    <script src="{% static 'js/contact.js' %}"></script>
    <!-- custom js -->
    <script src="{% static 'js/custom.js' %}"></script>

</body>

</html>
```

Listing 6: Base code for server home page (base.html).

```
{% extends 'base.html' %}
{% load static %}
{% block content %}


    <!-- banner part start-->
    <section class="banner_part">
        <div class="container">
            <div class="row align-items-center">
                <div class="col-lg-5 col-xl-5">
                    <div class="banner_text">
                        <div class="banner_text_iner">
                            <h1>FHE-based Brain Tumor Classifier (Server)</h1>
                            <h5>Powered by Concrete-ML</h5>

                                <form accept-charset="UTF-8" action="{% url 'start_classification' %}" method="POST" enctype="multipart/form-data">
                                    {% csrf_token %}
                                </form>
                        </div>
                    </div>
                </div>
                <div class="col-lg-7">
                    <div class="banner_img">
                        <img src="{% static 'img/banner_img.png' %}" alt="">
                    </div>
                </div>
            </div>
        </div>
    </section>
    <!-- banner part start-->

{% endblock %}
```

Listing 7: Server home page (index.html).

```
import tkinter as tk
import customtkinter

from tkinter import messagebox, filedialog, END, INSERT
from customtkinter import (
    CTk,
    CTkButton,
    CTkEntry,
    CTkFont,
    CTkFrame,
    CTkLabel,
    CTkTextbox,
    IntVar,
    StringVar,
    set_appearance_mode,
    set_default_color_theme)

import os, requests, stat, pathlib, shutil, subprocess, zipfile, traceback, urllib, json
import pandas, numpy
from pandas import DataFrame, read_csv
from numpy import save
from datetime import datetime
from concrete.ml.deployment import FHEModelClient

class ClientGUI:
    def __init__(self, master=None):

        # Initialize FHEModelClient
        self.fhe_model_client = FHEModelClient(os.path.dirname(__file__), os.path.join(os.path.dirname(__file__), "keys"))
        self.data_dictionary = {}


        # Create folder for keys and predictions if they don't exist
        this_folder = os.path.dirname(__file__)
        required_folder_names = ["keys", "predictions"]

        for name in required_folder_names:
            if not os.path.exists(os.path.join(this_folder, f"{name}")):
```

```
39                 os.mkdir(os.path.join(this_folder, f"{name}"))
40
41
42         ### Building the user interface of the app
43         # Initialize customtkinter
44         self.root = CTk()
45
46         # System Settings
47         set_appearance_mode("system")
48         set_default_color_theme("green")
49
50         # Custom CTk appearance
51         self.root.configure(padx=20, pady=20,
52                             fg_color='#3d4539',
53                             )
54         self.root.geometry("900x950")
55         self.root.resizable(True, True)
56         self.root.title("FHE-based Brain Tumor Classifier (Client)")
57
58         # Top Label
59         self.title = CTkLabel(self.root)
60         self.title.configure(
61             text='FHE-based Multi-Class Brain Tumor Classifier (Client)',
62             fg_color='#aee895',
63             font=CTkFont(size=30, weight='bold'),
64             text_color='#1d2b17',
65             justify ='center',
66             )
67         self.title.pack(fill='x', pady = 10,)
68
69         # Description of the App
70         self.description_frame = CTkFrame(self.root)
71         self.about_label = CTkLabel(self.description_frame)
72         self.about_label.configure(
73             font=CTkFont(size=20, weight='bold'),
74             text='About the Tool',
75             text_color='#1d2b17',
76             )
77         self.about_label.pack(
78                             # expand=False, fill="both",
79                             pady=10, side="top"
80                               )
81         self.description_label = CTkLabel(self.description_frame)
82         self.description_label.configure(
83             justify='center',
84             text='This tool implements FHE-based logistic regression model for tumor classification using gene
    expression data.',
85             font=CTkFont(size=16),
86             text_color='#1d2b17',
87             fg_color='white',
88             )
89         self.description_label.pack(expand=False, fill="x", side="top")
90         self.description_frame.pack(
91             fill="both", ipady=10, padx=20, pady=20, side="top")
92
93         # Data Preprocessing: Feature Selection and Encryption
94         # Variables for filenames
95         self.preprocessing_var = StringVar()
96         self.decryption_var = StringVar()
97
98         # Preprocessing Frame
99         self.preprocessing_frame = CTkFrame(self.root)
100
101         self.preprocessing_label = CTkLabel(self.preprocessing_frame)
102         self.preprocessing_label.configure(
103             text='Upload your .csv file for feature selection, encryption, and prediction here:',
104             justify='left',
105             text_color='#1d2b17',
106             font=CTkFont(size=16),
107             )
108         self.preprocessing_label.grid(row=0, column=0, padx=10, pady=10, sticky="nw")
109
110         self.preprocessing_filename = CTkEntry(self.preprocessing_frame, textvariable=self.preprocessing_var)
111         self.preprocessing_filename.configure(
112             justify='left',
113             width=640,
114             exportselection=False,
115             state="disabled",
116             takefocus=False,
117             )
118         self.preprocessing_filename.grid(row=1, column=0, padx=10)
119
120         self.preprocessing_browse = CTkButton(
121             self.preprocessing_frame,
122             hover=True,
123             )
124         self.preprocessing_browse.configure(
125             hover_color='#2c780b',
126             text='Browse File',
127             # width=300,
128             font=CTkFont(size=15),
129             command=self.browseRawFile
```

```
130         )
131         self.preprocessing_browse.grid(row=1, column=2, padx=10)
132 
133         self.preprocessing_begin = CTkButton(self.preprocessing_frame)
134         self.preprocessing_begin.configure(
135             hover_color="#2c780b",
136             text='Submit Data for FHE Classification',
137             width=300,
138             font=CTkFont(size=15),
139             command = self.processInput
140             )
141         self.preprocessing_begin.grid(row=2, column=0, columnspan=3, pady=10)
142 
143         self.preprocessing_frame.pack( anchor="w", fill="x", padx=20, pady=10, side="top")
144 
145         # Output Status Frame
146         output_tracker_frame = CTkFrame(self.root)
147 
148         self.output_tracker_label = CTkLabel(output_tracker_frame)
149         self.output_tracker_label.configure(
150             text='Output Window',
151             text_color='#1d2b17',
152             font=CTkFont(size=20, weight='bold'),
153             justify='center',
154             )
155         self.output_tracker_label.pack(pady=10, side="top")
156         self.output_tracker = CTkTextbox(output_tracker_frame)
157         self.output_tracker.configure(height=75, state="disabled")
158         _text_ = 'Track the status of your data here.'
159         self.output_tracker.configure(state="normal",
160                                       text_color='#1d2b17',
161                                       font=CTkFont(size=18),
162                                       )
163         self.output_tracker.insert("0.0", _text_)
164         self.output_tracker.configure(state="disabled")
165         self.output_tracker.pack(expand=True, fill="both", padx=10, pady=10)
166 
167         output_tracker_frame.pack(expand=True, fill="both", padx=20, pady=10, side="top")
168 
169         self.root.protocol("WM_DELETE_WINDOW", self.on_closing)
170 
171         # Main Widget
172         self.mainwindow = self.root
173 
174     ### Running the UI
175     def run(self):
176         self.mainwindow.mainloop()
177 
178 
179     ### Verifying the action of closing the app
180     def on_closing(self):
181         if messagebox.askyesno(title="Quit?", message="Do you really want to quit?"):
182             self.root.destroy()
183 
184 
185     ### Function for writing argument 'string' to the app's output window. Set argument 'delete_switch' to True
     to clear the window before printing.
186     def writeOutput(self, string, delete_switch = False):
187         self.output_tracker.configure(state="normal")
188         if(delete_switch):
189             self.output_tracker.delete("1.0", END) #tk.END
190             self.output_tracker.insert("0.0", f"{string}\n\n")
191         else:
192             self.output_tracker.insert(INSERT, f"{string}\n\n")
193         self.output_tracker.see(END)
194         self.output_tracker.configure(state="disabled")
195 
196 
197     # Function for getting the size of a file (i.e. private key, eval key, encrypted input)
198     def get_size(self, file_path, unit='bytes'):
199         file_size = os.path.getsize(file_path)
200         exponents_map = {'bytes': 0, 'kb': 1, 'mb': 2, 'gb': 3}
201         if unit not in exponents_map:
202             raise ValueError("Must select from \
203             ['bytes', 'kb', 'mb', 'gb']")
204         else:
205             size = file_size / 1024 ** exponents_map[unit]
206             return round(size, 3)
207 
208 
209     ### Function for running the entire processing of input data from feature selection to FHE inference proper
210     def processInput(self):
211         self.getFeaturesAndClasses()
212         self.dropColumns()
213         self.encryptInput()
214         self.decryptPrediction()
215 
216 
217     ### Browse raw input file of client
218     def browseRawFile(self):
219         filename = filedialog.askopenfilename(initialdir = "./",
220                                               title = "Select a File",
```

```
                                        #   filetypes = (("all files","*.*"))
                                       )
        self.preprocessing_var.set(filename)


    ### Get the classes and final features used in the model from features_and_classes.txt file
    def getFeaturesAndClasses(self, file = os.path.join(os.path.dirname(__file__), "features_and_classes.txt")):
        with open(file, "r") as fc_file:
            dictionary = json.loads(fc_file.readline())
            self.selected_features = dictionary["features"]
            self.classes_labels = dictionary["classes"]
            self.classes_labels = {int(key):value for key, value in self.classes_labels.items()}


    ### Feature Selection
    def dropColumns(self):
        filename = self.preprocessing_var.get()

        if(not filename.endswith(".csv")):
                raise Exception("Invalid file type. Only .csv files are supported.")

        self.writeOutput("Beginning to process your data for feature selection...")

        features = self.selected_features
        feature_list = ["samples"] + features

        drop_df = read_csv(filename)
        drop_df = drop_df[[column for column in feature_list]]
        drop_df.to_csv("./client-gui/feature_selection_output.csv", index=False, header=True)

        self.writeOutput("Feature Selection DONE!")


    ### Encryption of pre-processed client input (feature_selection_output.csv)
    def encryptInput(self):
        try:

            for f in os.path.join(os.path.dirname(__file__)):
                if f.split("/")[-1] in ["encrypted_input.txt", "serialized_evaluation_keys.ekl"]:
                    os.remove(f)

            self.writeOutput("Generating keys...")

            # Client generates private key and evaluation key
            self.generateKeys()

            encryption_input = os.path.join(os.path.dirname(__file__), "feature_selection_output.csv")
            df = read_csv(encryption_input)
            arr_no_id = df.drop(columns=['samples']).to_numpy(dtype="uint16")

            # Encrypted rows for input to server
            encrypted_rows = []

            # Encrypted dictionary for outputs
            count = 0
            for id in df['samples']:
                self.data_dictionary[count] = {'id':id, 'result':''}

            for row in range(0, arr_no_id.shape[0]):
                clear_input = arr_no_id[[row],:]
                encrypted_input = self.fhe_model_client.quantize_encrypt_serialize(clear_input)
                self.writeOutput("Encrypting pre-processed data...")
                encrypted_rows.append(encrypted_input)

            # Final encrypted input
            self.encrypted_rows = encrypted_rows

            self.writeOutput("Data Encryption DONE!")

            # Save encrypted input into a .txt file and the eval key into a .ekl file
            self.saveEncryption()

            self.writeOutput("Encrypted inputs and key files saved to 'encrypted_input.txt' and '
     serialized_evaluation_keys.ekl'. Please do not move these files until after prediction.")

            # Size of encrypted input
            encrypted_input_path = os.path.join(os.path.dirname(__file__), "encrypted_input.txt")
            encrypted_input_size = self.get_size(encrypted_input_path, 'kb')
            print("Encrypted input size (kB): ", encrypted_input_size)

            # Initialize requests object for client-server interaction
            app_url = "http://localhost:8000"
            client = requests.session()
            client.get(app_url)

            pred_zip_name = self.sendEncryptRequestToServer(client=client)

            self.decryption_var.set(pred_zip_name)

        except Exception as e:
            self.writeOutput(f"Error: {traceback.format_exc()}")
```

```
312
313     ### Client key generation function
314     def generateKeys(self):
315         model_dir = os.path.dirname(__file__)
316         key_dir = os.path.join(os.path.dirname(__file__), "keys")
317
318         if(os.listdir(key_dir)):
319             for f in os.listdir(key_dir):
320                 shutil.rmtree(os.path.join(key_dir, f))
321
322         fhemodel_client = FHEModelClient(model_dir, key_dir=key_dir)
323
324         # The client first need to create the private and evaluation keys.
325         fhemodel_client.generate_private_and_evaluation_keys()
326
327         # Get the serialized evaluation key
328         self.serialized_evaluation_keys = fhemodel_client.get_serialized_evaluation_keys()
329
330         # Check the size of the private key (in kB)
331         priv_key_size = self.get_size("./client-gui/keys", 'kb')
332         print("Private key size (kB): ", priv_key_size)
333
334
335     ### Saving the output of encryption into a .txt file and the generated eval key into .ekl file
336     def saveEncryption(self):
337         filename = "encrypted_input.txt"
338         with open(os.path.join(os.path.dirname(__file__), filename), "wb") as enc_file:
339             for line in self.encrypted_rows:
340                 enc_file.write(line)
341
342         with open(os.path.join(os.path.dirname(__file__), r'serialized_evaluation_keys.ekl'), "wb") as f:
343             f.write(self.serialized_evaluation_keys)
344
345         # Check the size of the evaluation key (in kB)
346         eval_key_size = self.get_size("./client-gui/serialized_evaluation_keys.ekl", 'kb')
347         print("Evaluation key size (kB): ", eval_key_size)
348
349
350     ### Sends encrypted_input.txt and serialized_evaluation_keys.ekl (expected to be located in the same
          directory as the app) to the server-side app through the Python requests library. URL is set to localhost
          :8000 in development.
351     def sendEncryptRequestToServer(self, client):
352
353         app_url = "http://localhost:8000"
354
355         if 'csrftoken' in client.cookies:
356             # Django 1.6 and up
357             csrftoken = client.cookies['csrftoken']
358         else:
359             # Older versions
360             csrftoken = client.cookies['csrf']
361
362         eval_keys_file = open(
363                             (os.path.join(os.path.dirname(__file__), "serialized_evaluation_keys.ekl")),
364                             "rb"
365                             )
366         inputs_file = open(
367                           (os.path.join(os.path.dirname(__file__), "encrypted_input.txt")),
368                           "rb"
369                           )
370         request_data = dict(csrfmiddlewaretoken=csrftoken)
371         request_files = dict(inputs=inputs_file, keys_file=eval_keys_file)
372
373         self.writeOutput("Sending encrypted data and evaluation keys to server...")
374         self.writeOutput("Waiting for server's response...")
375
376         # Sending the files (encrypted input and eval key) to "localhost:8000/{function_name}" (server's FHE
          inference function)
377         request_output = client.post(f"{app_url}/start_classification", data = request_data, files=request_files,
          headers=dict(Referer=app_url))
378
379         if request_output.ok:
380             self.writeOutput(f"Response Code: {request_output.status_code}. FHE Classification DONE!")
381
382             # Save FHE inference result into a .zip file
383             with open(os.path.join(os.path.dirname(__file__), "predictions/enc_predictions.zip"), "wb") as z:
384                 z.write(request_output.content)
385
386         return os.path.join(os.path.dirname(__file__), "predictions/enc_predictions.zip")
387
388
389     ### Decryption of FHE inference result received by the client from the server
390     def decryptPrediction(self):
391
392         # Expects the input filepath (self.decrypt_name_var) to be a .zip file, and raises an error if not
393         try:
394             filename = self.decryption_var.get()
395
396             if not filename.endswith(".zip"):
397                 raise Exception("Invalid file type: Only .zip files are supported.")
398
399             decrypted_predictions = []
```

```
400
401             # Setting classes dictionary to be used for final output translation since the model used label
        encoding in training
402             try:
403                 classes_dict = self.classes_labels
404             except:
405                 classes_dict = {0: 'ependymoma', 1: 'glioblastoma', 2: 'medulloblastoma', 3: 'normal', 4: '
        pilocytic_astrocytoma'}
406
407
408             pred_folder = os.path.join(os.path.dirname(__file__), "predictions")
409             zip_name = filename
410
411             with zipfile.ZipFile(zip_name, "r") as zObject:
412                 zObject.extractall(path=pred_folder)
413
414             enc_file_list = [filename for filename in os.listdir(pred_folder) if filename.endswith(".enc")]
415
416             for filename in enc_file_list:
417                 with open(os.path.join(pred_folder, filename), "rb") as f:
418                     decrypted_prediction = self.fhe_model_client.deserialize_decrypt_dequantize(f.read())[0]
419                     decrypted_predictions.append(decrypted_prediction)
420
421             decrypted_predictions_classes = numpy.array(decrypted_predictions).argmax(axis=1)
422             final_output = [classes_dict[output] for output in decrypted_predictions_classes]
423
424             for i in range(len(final_output)):
425                 self.data_dictionary[i]['result'] = final_output[i]
426
427             decrypted_pred = [dictionary for dictionary in self.data_dictionary.values()]
428             print(decrypted_pred)
429
430             final_str = "The classification of your sample is: " + final_output[0].upper()
431             self.writeOutput(final_str)
432
433             # Save decrypted FHE inference result into a .csv file
434             self.savePrediction(decrypted_pred)
435
436         except Exception as e:
437             self.writeOutput(f"Error: {str(e)}")
438
439
440     ### Save final result into a .csv file
441     def savePrediction(self, dictionary):
442         final_pred = pandas.DataFrame.from_dict(dictionary)
443
444         now = datetime.now()
445         date = now.strftime("%Y_%d_%m")
446
447         fname = "predictions/" + date + "_final_prediction_output.csv"
448
449         final_pred.to_csv((os.path.join(os.path.dirname(__file__), fname)),
450                           index=False, header=True)
451
452         self.writeOutput("Your final prediction output has been saved! Check the predictions folder to view it.")
453
454
455 ### Download required client files from the project's GitHub Repository.
456 ### By default, targets the project's GitHub repository for downloading the files, but can be set to localhost
        :8000 to download from the local deployment server.
457 def getClientFiles():
458     files = [
459         r"https://github.com/gcrosario/concreteML-FHE-Tumor-Classification/raw/master/FHE-Compiled-Model/client.
        zip",
460         r"https://raw.githubusercontent.com/gcrosario/concreteML-FHE-Tumor-Classification/master/FHE-Compiled-
        Model/features_and_classes.txt",
461         ]
462     for file in files:
463         print(file.split("/")[-1].replace("%20", " "))
464         if file.split("/")[-1].replace("%20", " ") not in os.listdir(os.path.dirname(__file__)):
465             download(file, os.path.dirname(__file__))
466
467 def download(url, dest_folder):
468     if not os.path.exists(dest_folder):
469         os.makedirs(dest_folder)
470
471     filename = url.split('/')[-1].replace(" ", "_")
472     file_path = os.path.join(dest_folder, filename)
473
474     r = requests.get(url, stream=True)
475
476     if r.ok:
477         print("saving to", os.path.abspath(file_path))
478         with open(file_path, 'wb') as f:
479             for chunk in r.iter_content(chunk_size=1024 * 8):
480                 if chunk:
481                     f.write(chunk)
482                     f.flush()
483                     os.fsync(f.fileno())
484     else:  # HTTP status code 4XX/5XX
485         print("Download failed: status code {}\n{}".format(r.status_code, r.text))
486
```

```
487 if __name__ == "__main__":
488
489     print("Downloading required client files.... Client app will open once downloads are finished.")
490     # Download required client files upon launching of the client GUI app
491     getClientFiles()
492     print("Required client files saved! Launching the app... \n")
493
494     app = ClientGUI()
495     app.run()
```

Listing 8: Client GUI app (client-app.py).

XI. Acknowledgment

First, I would like to praise and thank God, for granting me His guidance and blessing in this chapter of my life.

I would also like to express my deepest gratitude to everyone who extended support and assistance throughout my SP and college journey. This project would not have been possible without the following people:

My adviser, Sir Richard Bryann Chua, for giving me the opportunity to work on this study and to be a part of the Security and Cryptography research group. Thank you for continuously guiding me throughout the whole SP process. I'm very grateful for the frequent consultations as they helped me understand the topic more and make regular progress with my SP. Your feedback greatly impacted the output as it allowed me to see points for revision and improvement.

My fellow member of the Homomorphic Encryption subgroup, Johann Benjamin Vivas, for collaborating with me and exchanging ideas in implementing our projects and in writing our documentations.

My family for always believing in me. Thank you Nanay and Tatay for providing my needs and for your support. I would also like to thank my siblings for encouraging me as I faced this challenge. Your belief in me has given me strength to overcome my fears during this process.

My best friend, Sophia Veronique, for being with me in this journey. Thank you for lending your ears whenever I need someone to listen to my stories, be it of success or of failure, as I worked in this project. I'm also thankful for your words of encouragement that gave me emotional support during my lowest moments.

My blockmates and friends, for supporting and inspiring me. Thank you for helping me succeed in this journey by sharing your time and knowledge. Thank you also for giving me comfort with your words and our memories.

Lastly, NCT Dream, for giving me motivation and strength through their music.