Privacy-Preserving Viral Strain Classification Through a Client-Server Application Using An Open-Source Fully Homomorphic Encryption Library

Johann Benjamin P. Vivas
Department of Physical Sciences and Mathematics
University of the Philippines Manila

Adviser: Richard Bryann Chua

Contents

1 Introduction

1.1 Background of the Study

In recent years, the issue of data privacy has become one of the most talked-about topics in multiple domains. Among these are the legal, technological, medical, and financial domains, where it is often mentioned how a dangerous situation may arise from leaked financial, medical, legal, or personal data, and how practicing data privacy is paramount to avoid such situations. [1, 2] Even with efforts including acts such as the General Data Protection Regulation in the European Union and European Economic Area [3], and the Data Privacy Act of 2012 in the Philippines [4], there are many issues and concerns that remain unresolved.

These pressing issues include the privacy concerns with sharing viral genomic sequencing data, as discussed by Song et al [5]. In their work, they brought attention to the nuances in this matter, acknowledging that the sharing of viral genomic sequencing data, in light of recent developments in the COVID-19 pandemic, is tantamount to properly informing regional, national, and international public health responses. It was also important to the development and improvement of clinical therapies and vaccines and allowing researchers, scientists, and experts to better understand the SARS-CoV-2 virus. They also maintain that while the sharing of SARS-CoV-2 viral genomic sequences alone is extremely unlikely to lead to the re-identification of the data' owners due to the SARS-CoV-2 virus's short mutation rate and even shorter serial interval, some minimal contextual data is required to properly assess and process the data itself. These contextual data fields can take the form of the gender, age, province/territory, or a combination of these factors, or can also take other forms, depending on what would bring the most value to the data while minimizing privacy risks.

Further, Song et al. mention that, in most situations, the inclusion of such contextual data fields in association with the sequencing data is not contradictory to privacy laws. The authors do, however, note that extra care should be taken to evaluate whether the inclusion of these contextual data fields with viral sequence data would disproportionately increase the risk of re-identification, and thus the risk to privacy on the whole. They explained that this can happen due to the sensitivity of the data in relation to factors such as the population pool and confirmed cases in the specific province. Following this, some particular examples in which certain contextual data fields disproportionately increased the risk of re-identification were brought up by the authors, beginning by mentioning the Gordon v. Canada 2008 federal court case, in which the data field of “province” or “territory” can create a disproportionate risk of re-identification in provinces and territories with a smaller population [6, 7]. Song et al. next discussed the potential re-identification risks of providing data fields similar to “collection agency”, which they noted is not uncommonly the name of a local hospital that could provide more detailed geographical information, increasing re-identification risk.

Next, the gender and age data fields were mentioned and described as potentially disproportionately increasing the risk of re-identification when paired with other contextual data, as also discussed by [8, 9, 10]. The amount that the re-identification risk increases varies widely depending on the age bracket in question, as the very elderly or very young individuals may in some cases make up a significantly smaller fraction of a particular population, and should be considered more risk-prone as a result. In such cases, special mitigating measures should be taken to reduce the risks of re-identification, and the authors also recommend the periodic monitoring of re-identification risk to account for the increased efficiency of diagnostic methods and other relevant developments that could potentially increase privacy risks.

Adding to the points already raised are the recent events concerning SARS-CoV-2 sequences and metagenomics data from China that was uploaded to the GISAID database, which were associated with samples dated January 2020 from the Huanan Seafood Wholesale Market in Wuhan, China [11]. Various sources including the World Health Organization and Ars Technica report that analyses of these data suggest that apart from SARS-CoV-2 sequences, some samples also contained human DNA, as well as mitochondrial DNA of several animal species, including some that are known to be susceptible to SARS-CoV-2 [12, 13, 14]. This development raises the concern that SARS-CoV-2 sequences hosted on public databases such as GISAID may contain trace amounts of human DNA, which is corroborated by the work of Lehrer & Rheinstein [15] and Li et al. [16]. These studies discuss that many SARS-CoV-2 human samples do contain trace amounts of human DNA and RNA, which has the potential to increase the risk of reidentification, especially when considered in tandem with the privacy risks that even minimal contextual data fields associated with the sequences pose and further warrants research into ways to reduce the likelihood of reidentification associated with sharing SARS-CoV-2 sequences and contextual data.

Further, the events previously discussed have also affected GISAID's credibility, and have caused the repository to come under scrutiny in light of other shortcomings with regard to data sharing limitations and the manner in which GISAID handles disputes of credit and sanctions scientists who have allegedly violated the terms and conditions of the repository [17]. This has led to a growth in sentiment for fully open-access data sharing, as evidenced by an open letter published by a group of scientists urging other researchers to publish their datasets on open-access databases [18]. However, publishing datasets, particularly those as timely as SARS-COV-2 sequence datasets, on open-access databases poses a risk of the data being 'scooped', a practice in other scientists publish studies using the dataset first instead of the original authors and creators of the dataset. This prevents the timely sharing of potentially crucial data.

From the works of the aforementioned authors and the developments that have transpired in recent years, it can be clearly seen that there is a need for low-risk, privacy-preserving viral genomic data that will allow researchers and experts to improve their understanding of COVID-19, as well as other viral diseases, as well as share their findings and genomic data with other researchers to allow for faster and more successful strides towards better medical responses and healthcare. One potential solution to this is the use of Homomorphic Encryption, which, as defined by various sources, is a relatively new form of encryption technology that allows operations such as addition and multiplication to be performed on encrypted data without the need for decryption beforehand. Virtually any algorithm can be computed and performed on encrypted numbers with the two available operations, and this can even include the analysis and classification of encrypted biomedical data, such as DNA sequences [19, 20, 21]

1.2 Statement of the Problem

The sharing of viral sequence data along with some of its contextual data is vital to the analysis and surveillance of the SARS-CoV-2 pandemic through various processes such as viral sequence classification. That being said, while Song et al [5] explain that viral sequence data alone does not substantially increase the risk of reidentification of an individual due SARS-CoV-2's mutation rate and serial interval, these characteristics can lead to the identification of groups of individuals. To elaborate, SARS-CoV-2's serial interval, the interval between the onset of symptoms in an infector (individual that transmits the virus) and the infectee (individual infected by the virus from the infector), is shorter than its rate of mutation. This means that multiple infector-infectee pairs can share the same viral sequence and that single individuals cannot be identified by any particular strain. However, entire groups of individuals can be identified by these same characteristics. Given this, the identification of groups of individuals can still raise ethical concerns as it could possibly lead to ostracization or discrimination due to the social stigma that being diagnosed with SARS-CoV-2 carries.

What this means is that viral strain classification, along with other methods to study, classify and protect against SARS-CoV-2 strains based on patients' viral sequence data is potentially at odds with legal and ethical boundaries regarding the patients' rights to data privacy and protection, and thus poses a risk of endangering the data's owner. In the context of the privacy concerns regarding the sharing of viral sequencing data and the use of viral strain classification, the use of Homomorphic Encryption instead addresses the problem from the angle of improving the privacy-preserving properties of our methods as opposed to adjusting the data to meet our needs. This was explored in one of the challenges put forward in the 2021 IDASH Privacy & Security Workshop [22]. The workshop had brought attention to the issue and contributed to the development of various models that were able to successfully preserve the privacy of the patients who owned the viral sequence data by allowing viral strain classification to be performed on encrypted data without prior decryption, thereby significantly reducing the risk of re-identification.

While the results of the iDASH competition were reassuring in that viral strain classification was shown to be a practical task, as the performance of most of the models in performing privacy-preserving viral strain classification was impressive, there were still some optimization issues that led to large variability in the time cost of the classification of the sequences. It also brings attention to the potential of FHE in viral strain classification, while also encouraging many others to explore the use of other publicly available methods to implement solutions to the privacy concerns that plague the sharing and classification of viral sequencing data.

Thus, at the core of this dilemma lies the following questions: "How well does an open-source FHE library lend itself towards the implementation of practical privacy-preserving viral strain classification?", and "What is the most appropriate machine learning method for viral strain classification that can be implemented in FHE?"

This study aims to answer these questions through the discussion and use of existing FHE libraries with features that lend themselves to machine learning implementations.

1.3 Objectives of the Study

The objective of this study is to develop an application that allows researchers and service providers to perform the homomorphic computations required to conduct viral strain classification on data that is provided by clients on the client version of the application, effectively ensuring data is encrypted during the entire servicing process. The study resulted in the creation of an application with a client-server model. Here, the client represents a hospital, clinic, or other medical institution that would require the use of ML services to provide better diagnoses and treatment for viral diseases and would need to outsource such tasks to ML service providers. Likewise, the server represents an ML service provider.

The proposed model has the following functionalities:

Allow the hospital (client) to:

1. Encrypt and upload their preprocessed viral strain data

2. Send their encrypted data to the ML service provider for classification

3. Receive and decrypt the results of the classification

Provide the ML service provider (server) with the following capabilities:

1. Receive the encrypted viral strain data from the hospital
2. Perform homomorphic viral strain classification via Logistic Regression
3. Send the results of the homomorphic classification to the hospital

1.4 Significance of the Project

The exploration and implementation of a privacy-preserving viral strain classification application are of great benefit to various medical fields, as through it, clients, such as medical professionals and researchers, are better equipped to study viral strain sequences without the risk of violating privacy standards, potentially allowing them to develop and/or roll out appropriate treatment and medical initiatives more effectively.

Patients would also benefit greatly from the implementation of this application as well. The use of FHE to encrypt their data significantly reduces the chances of their viral sequence data being used to re-identify them.

1.5 Scope and Limitations

For this study, the author will work primarily with the existing FHE library Concrete-ML. Concrete-ML was chosen for the implementation of the application in this study due to its propensity for being used in Machine Learning or Deep Learning models and applications, by virtue of its various built-in ML models that can be easily compiled from its plaintext version into a corresponding FHE version of the model. These models are also compatible with Scikit-learn modules, processes, and workflows. Concrete-ML is also implemented in Python, allowing for easier programming due to the high-level nature of the language. One major advantage that Concrete-ML boasts is that it also supports tree-based classification models in FHE, which meant that the implementation of such was significantly easier than first envisioned as there are no known general conditional statements in FHE at the time of writing, which is important in the creating of tree-based classifiers from scratch.

As Concrete-ML compiles their plaintext models into equivalent FHE circuits through Concrete's FHE compiler to produce an equivalent FHE circuit, which is performed under the hood. Additionally, Concrete-ML's FHE compiler makes use of the TFHE scheme only, and adjustments, such as the quantization of non-integer inputs, are required. Concrete-ML's FHE engine also does not support data batching, so loops are required for the implementation of ML through Concrete-ML.

The application will be built primarily for the Windows operating system due to the ease of use and large user base of the system. However, due to operating system limitations for tools considered essential to the study such as Dashing, the Windows Subsystem for Linux will be used for functions and tools that do not support Windows, such as Dashing.

This setup allows future studies to reproduce and improve upon this study with more ease, allowing future researchers to focus their efforts on more important aspects.

1.6 Assumptions

1. The client in the model will provide valid viral strain sequences in FASTA format, which is ideally preprocessed using techniques such as Dashing [23], as performed by the A*FHE team, who were behind the CoVnita framework [24].

2 Review of Related Literature

2.1 Machine Learning and its use in Viral Strain Classification

In recent years, Machine Learning has often been used in the diagnosis of viral diseases with the long-term goal of preventing their spread. One such instance of the use of ML can be seen in Remita et al's work [25], where in the course of the study, the group developed a virus classification platform, dubbed CASTOR, which simulates, in silico, the restriction digestion of genomic material by different enzymes into fragments. Remita et al. explored and incorporated various machine learning algorithms, each of which fell under one of three types: symbolic methods, characterized by decision trees and random forests, statistical methods, such as the naive Bayes classifier and support vector machine, and k-nearest neighbors, and ensemble methods, under which Adaboost and Bagging fell. The performance of the platform was benchmarked for the classification of distinct datasets of human papillomaviruses (HPV), hepatitis B viruses (HBV), and human immunodeficiency viruses type 1 (HIV-1). The classification results ended with the authors noting that, in general, SVM had performed better in four of the five datasets utilized with the mean of weighted $F-measures$ higher than 0.906, ranking first in HPV Alpha species, HBV genotypes and HIV-1 subtypes classifications, and fourth in HPV genera classification. SVM is followed by Random Forest, Naive Bayes Classifier, and K-Nearest Neighbors in terms of performance, though Remita et al. note that the Random Forest and Naive Bayes classifiers are affected by large variances.

Kim et al. [26] also achieved the successful classification of porcine reproductive and respiratory syndrome virus (PRRSV) sublineages, while also detecting key amino acid positions. The authors achieved this by implementing four ML approaches based on the amino acid scores of the ORF5 gene. The approaches implemented were random forest, support vector machine, k-nearest neighbors, and multilayer perceptron. Kim et al. also performed experiments to see how the number of amino acid sequences affected the performance and time consumption of the ML algorithms. The results of their study revealed that all four ML approaches tested for the classification of PRRSV sublineages had been able to perform accurate classification of the sublineages (all algorithms having an AUC of 0.98 and above) when using at least 2 amino acid positions: the two amino acid positions with the highest RF scores. When only one amino acid sequence (with the highest RF importance score) was used, the accuracy had dropped down to around 80% for all algorithms. The results also showed that the accuracy for all four ML algorithms in classifying the PPRSV sublineages were identical.

Other studies, such as those by Lopez et al. [27] and Câmara et al. [28], focused on the classification of Sars-CoV-2 viral strains, with both studies choosing to utilize deep learning in the form of convolutional neural networks, or CNNs, to facilitate their operations. To elaborate on this, in their study, Lopez et al. trained a convolutional neural network on 553 sequences from the National Genomics Data Center (NGDC) repository, resulting in a classifier capable of separating the genome of different virus strains from the Coronavirus family with 98.73% accuracy. Câmara et al. worked with 1557 instances of SARS-CoV-2 from the National Center for Biotechnology Information (NCBI), and 14,684 different viruses from the Virus-Host DB. Further, as their CNN was highly customizable with several changeable parameters, the tests were performed on forty-eight different architectures with the best of these having an accuracy of $91.94 \pm 2.62\%$ in classifying viruses into their realms correctly.

2.2 Privacy concerns with Machine Learning in the Medical Domain

It cannot be understated how much of an impact these studies, along with many others, have collectively had on the medical and computer science domains. These studies have allowed medical professionals and health informatics experts to identify and either prevent the spread or treat those who have been diagnosed with viral diseases. However, these methods have also had concerns of various natures raised against them, which Char et al. [29, 30] illustrate in their work, discussing concerns such as those that ML algorithms may mirror human biases in decision-making, as well as concerns about the intent behind the design of machine-learning systems, as algorithms can be designed to perform in unethical ways. Despite this, they acknowledge that the incorporation of machine learning into clinical medicine holds much promise as ML is capable of improving the overall quality and speed of healthcare delivery and can serve as a foundation on which several initiatives and projects headed by public and private companies can be built. Lastly, Char et al. reiterate that the consideration of ethical issues and problems inherent in the implementation and use of machine learning in healthcare systems is also warranted [29, 30].

The main issue being addressed in this study, however, is the issue of data privacy, integrity, and overall protection from potential bad actors. In recent years, as machine learning methods and techniques have become more complex and resource-intensive, researchers and companies have turned to the use of cloud computing to achieve the results they need in a cheaper and more cost-effective manner. With that, when a machine learning model is trained or classified in a cloud environment, the server itself obtains data from the user side. Given this, the privacy of the data then depends on the service provider, and thus it could be very easy for malicious acquisition and utilization of data to occur [31, 32]. As can be seen from the issues previously discussed by [5, 30, 29, 31, 32], there is a need for methods to address the concerns of privacy and data security.

2.3 Fully Homomorphic Encryption

While there have been several methods of achieving data privacy and anonymization in the past, as evidenced by [33], one such approach has been touted by several as the "holy grail" of cryptography: Homomorphic Encryption, and in particular, Fully Homomorphic Encryption. Homomorphic encryption (HE), as described by numerous works [32, 34, 35, 31], can be described as a relatively new form of technology that allows data to be encrypted and for complex operations to be performed on it without having to decrypt that data in the process. Several different types of HE have been developed over time, including, but not limited to, partially homomorphic encryption (PHE) which originally only had support for either additional homomorphism or multiplicative homomorphism, meaning that only one of the two operations between encrypted data could be achieved to the intended effect, and fully homomorphic encryption (FHE), which fully realized the concept by allowing both addition and multiplication operations to be performed on encrypted datasets, as any arbitrary function could theoretically be implemented with these two operations.

2.4 FHE libraries and the Concrete-ML FHE ML library

There have been several advances in HE since its original conception by Rivest et al. [36]. All of these advances have culminated in today's FHE libraries, all implemented in various languages, with some implemented in relatively low-level languages like C++ and others in higher-level languages like Python. Some examples of these libraries are Microsoft SEAL, OpenFHE, TenSEAL, Concrete-ML, HElayers, and Pyfhel [37, 38, 34, 39, 40, 35].

While some of these libraries have seen use in different applications and studies which involve machine learning with data that is considered sensitive, such as Microsoft SEAL and HElayers, which were utilized by two of the top three contenders of the iDASH 2021 competition [41, 32, 24], Concrete-ML stands out from other FHE libraries for its specialization in incorporating FHE into its built-in ML models that are based on, and are compatible with, Scikit-learn processes and workflows. It also has the edge over other libraries in that, aside from already having pre-built ML algorithms, there are built-in algorithms for tree-based models, which are difficult to implement in FHE due to lacking general homomorphic conditional statements for comparison purposes. Concrete-ML was also implemented in Python, which lends to the user-friendliness of the library due to high-level abstractions and interfaces. Additionally, much of the parameter setting in Concrete-ML is handled by its native FHE compiler. All these traits make Concrete-ML a very accessible and relatively easy-to-implement-and-improve FHE ML library and are the reason for the author's choice to utilize this particular library for this study.

2.5 Applications of FHE

Applications that implement FHE in some of their features have also been implemented and even rolled out. These include works that involve privacy-preserving genome-wide association studies [21, 42], genotype imputation [43, 26, 43], and federated analytics [44]. One other noteworthy implementation of FHE [45] had introduced the Password Monitor feature to Microsoft Edge, which allowed Microsoft to monitor and alert users if their password had been found in a third-party breach. The password monitor features homomorphic encryption, allowing it to avoid learning the passwords of the user while monitoring its status.

It eventually came to be implemented in different domains that deal with very sensitive data. Machine Learning applications that implement FHE methods have been created and shared with the community, setting a precedent for the use of privacy-preserving encryption methods that allow for the processing of sensitive data without putting the owners of the data at risk [25, 31]. In direct relation is the creation and publication of various privacy-preserving viral strain classification applications as part of the iDASH 2021 competition [41, 32]. The results of the competition serve as a great example of the use of FHE to improve the state of Machine Learning and its use in the delivery of healthcare services, all while preserving the privacy of patient data. At the time of writing, there are two solutions from the competition that have received publications in websites and journals, or are currently being peer-reviewed: IBM's HElayers-based Privacy-Preserving Viral Strain Classification based on k-mer signatures and FHE [46, 32, 47, 48], and I2R's A*FHE-2 Team's End-to-end Privacy-preserving SARS-CoV-2 Classification Framework entitled CoVnita [24].

Akavia et al.'s work focused on the implementation of a client-server protocol through the Microsoft SEAL and HElayers libraries, wherein the model owner would provide viral strain classification services on encrypted data, which is uploaded by the DNA-Sequence owner. To elaborate, the process would begin with the client computing their DNA-Sequence's k-mer set. Next, the client would encrypt this set through FHE using any FHE scheme that supports the encryption of complex numbers, and send it to the server. Lastly, the actual classification is performed based on the k-mer sets derived from the DNA sequences, and the Jaccard similarity between a particular strain in the data and various sequences representative of each of the different Sars-CoV-2 viral strains. Once Jaccard similarity scores were successfully computed for the k-mer sets and normalized, the strain with the highest similarity score is chosen to be the classification of that particular strain, and classification would continue until the process was completed. On the other hand, A*FHE-2's solution was CoVnita, an end-to-end privacy-preserving framework for SARS-COV-2 classification. They implemented this framework, resulting in a viral strain classification model that used a logistic regression algorithm and was built on a federated learning approach, wherein various data providers train their own local models using their own data, after which a joint global model that does not require data providers to share their genomic samples, is trained. The model's FHE capabilities were powered by Microsoft SEAL. A*FHE also noted that they had chosen logistic regression as they felt that it is an appropriate machine learning model for the purpose of minimizing data exposure, as, unlike machine learning models like k nearest neighbors, which will expose all the individual data values, logistic regression restricts the sharing of information and also prevents the exposure of individual data values. It should also be noted that both publicly documented solutions had placed in the top three for the iDASH competition, and both used open-source libraries which were implemented in an ML context. Thus, this study aims to investigate whether practical privacy-preserving classification can be achieved with other open-source libraries, particularly those more specialized for applications in ML such as Concrete-ML.

3 Theoretical Framework

As Creeger and Cornami Inc. [49] put it, Homomorphic Encryption, in particular, Fully Homomorphic Encryption, aka FHE, are new and burgeoning technologies that have the potential to provide quantum-secure computing over encrypted data. FHE guarantees that the plaintext data and the results derived from its analysis, decomposition, computation, or other forms of processing, are kept secure from breach even with compromised infrastructures or models. Most of the FHE schemes today are based on lattice mathematics [50], and are considered safe from breaches via quantum computing, and to that end, are considered post-quantum cryptography.

3.1 Developments of FHE

The original concept of Homomorphic Encryption began to form in 1978 when Rivest et al. [51] recognized it and addressed homomorphic behavior in their study. They showed that when two RSA (Rivest-Shamir-Adleman) encrypted numbers are added together, their result, when decrypted, is equivalent to the result of the same operation if done on the unencrypted numbers [36]. This is called privacy homomorphism and is present in certain encryption schemes. By leveraging these properties, the separation of data processing and data access can be achieved, encrypted queries could be made possible, and data involved in such operations were never decrypted, at rest or during its life cycle. More than 30 years later in 2009, Craig Gentry [52] proposed the very first FHE scheme. Gentry defined the algorithms of the scheme as a series of logic gates, and unrestricted computation was possible with the scheme, and the results of computation with encrypted numbers were encrypted in much the same way. While Gentry's FHE scheme was extremely slow, taking half an hour to finish a single logic gate on available standard x86 hardware at the time as evidenced by the results of its implementation [53], it helped kickstart the development of other FHE schemes, which can be divided into four distinct generations:

3.2 FHE Across the Generations

The first generation of FHE can be characterized by the release of Gentry's FHE scheme. Gentry's publication paved the way for the further development of fundamental concepts in FHE, such as the introduction of noise to a public key by DGHV [54]. DGHV developed an encryption scheme that allowed homomorphic operations over integers, which they used to replace the SHE portion of Gentry's originally conceived scheme. The second generation of developments in FHE followed, defined by the development of the BFV (Brakerski/Fan-Vercauteren) and BGV (Brakerski-Gentry-Vaikuntanathan) schemes [49]. These schemes introduced the LWE (Learning With Error) and RLWE (Ring Learning With Error) security models, as well as leveling schemes that allowed for the execution of a logic-gate circuit of a certain set depth before requiring a bootstrap to manage the noise level. Meanwhile, the most noteworthy events that formed the third generation of FHE developments included the introduction of the GSW (Gentry-Sahai-Waters) homomorphic encryption scheme, which avoided relinearization, which was considered computationally expensive, when performing homomorphic multiplication. Due to this, the scheme exhibited slower noise growth. The development of more efficient ring variants with FHEW (Ducas-Micciancio — "Fastest Homomorphic Encryption in the West"), as well as the simplification and increased optimization of bootstrapping, were also observed in this generation. Lastly, the fourth generation brought the CKKS (Cheon-Kim-Kim-Song) scheme, which introduced efficient rounding operations for encrypted values, controlling noise rate increases in HE multiplication and reducing the number of bootstraps required in a logic circuit. The generation also brought the concept of PBS (programmable bootstrapping) to TFHE (torus fully homomorphic encryption) [55], reducing the number of bootstraps required in a logic circuit.

3.3 The Concrete-ML FHE ML Library

We will be utilizing the open-source FHE ML library Concrete-ML for its specialization in machine learning and deep learning, and execution of ML/DL algorithms in FHE, as well as its implementation in Python, allowing easier programming thanks to its high-level nature and abstractions. It is also compatible with Scikit-learn modules and workflows, and has built-in support for implementation in client-server architectures. Concrete-ML also features supports a variety of regression and classification models that can be compiled to FHE equivalents. These supported models are categorized into linear models, tree-based models, and neural networks for deep learning.

For linear models, Concrete-ML supports linear regression, logistic regression, linear SVC, linear SVR, Poisson regressor, Tweedie regressor, gamma regressor, lasso, ridge, and elastic net [56]. For tree-based models, it supports decision tree classifier, decision tree regressor, random forest classifier, and random forest regressor, which are scikit-learn compatible, as well as the XG classifier and XG regressor, from XG Boost [57]. Lastly, Concrete-ML supports neural net classifiers and neural net regressors for its neural network repertoire [58]. All in all, these models provide a better interface for exploring FHE ML methods and finding the best fit for viral strain classification [39].

3.4 Concrete-ML Workflow

A Concrete-ML model's lifecycle can be divided into the model's development and its deployment [59]. Under the model development phase, the following actions are taken:

1. The **training** of the model, where a model is trained on plaintext/unencrypted training data. Since ConcreteML only supports FHE inference at the time of writing, this particular step of model development can only be performed using plaintext data.

2. The **quantization** of the model. The process of quantization converts inputs, model weights, and all intermediate values of the inference computation to integers. The point at which quantization is performed depends on the model type. It can be performed either during training (also dubbed Quantization-Aware Training) for Neural Networks or after training (Post-Training Quantization) for Linear Models. For tree-based models, the data instead of the model is quantized into its integer equivalent. The main parameter that affects the quantization process is the number of bits, *n_bits*.

   In the case of linear models, *n_bits* can use a single integer value dependent on the number of attributes. For a higher number of attributes, it is recommended to use a lower value. On the other hand, a dictionary of integer values can also be passed to *n_bits* to allow it to use different parameters. For tree-based models, the maximum accumulator bit-width is determined to be *n_bits*+1 bits. As the Concrete FHE framework is only limited to 8-bit integers, *n_bits* must be less than 8. Further information on the process can be accessed via the Zama AI documentation on Quantization [60].

3. The **compilation** of the model. This process is handled in its entirety by Concrete-Numpy [61]. Concrete-ML hides away most of the complexity of this step by allowing users to compile the model by simply using the `compile()` function [62].

4. Performing **FHE inference** using the compiled FHE Concrete model [59, 60, 61].

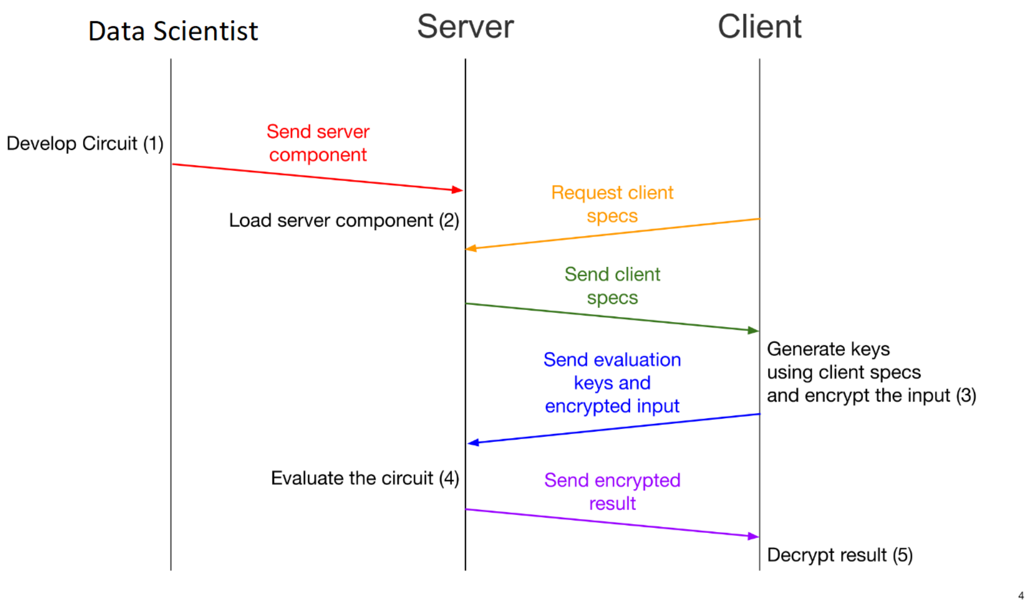


Figure 1: Summary of the overall communications protocol for deploying machine learning services [63].

Next, under the model deployment phase, Concrete-ML provides built-in functionalities for deploying the compiled FHE models in a client/server setting. The deployment workflow and model serving pattern are described as follows: for deployment, when the training and compilation of the model to its FHE equivalent is performed, three files are created when saving the model. These are:

1. **client.zip**, which contains the secure cryptographic parameters (`client.specs.json`) needed for the client to generate private and evaluation keys. It also contains **serialized_processing.json**, which contains required metadata about pre and post-processing, such as quantization parameters to quantize the input and dequantize the output, and a copy of `versions.json`.

2. **server.zip**, which contains the compiled model. This file is used to run the model on a server.

3. **versions.json**, which contains information about the version of Concrete-ML used to compile the model. This file is important since compiled Concrete-ML models do not work with older versions of Concrete-ML.

Next, ConcreteML provides users with the concrete.ml.deployment.fhe_client_server module, which contains several APIs for FHE deployment, to make this process possible. Users make use of various classes included in this module for the deployment process. The FHEModelDev class is used for saving and exporting the files needed for the client-server system by first instantiating the class and using the `save()` method. The process of loading and running the compiled FHE circuit is performed by using the `run()` methods of the FHEModelServer class. Lastly, the FHEModelClient class is used for the encryption and decryption of the client's data. Once the client is created and the model is loaded, the private and evaluation keys are generated via the `generate_private_and_evaluation_keys()` method. After that, the serialization evaluation keys are retrieved and stored in a variable using the `get_serialized_evaluation_keys()` method. The evaluation key is then sent to the server, therefore ensuring that all requirements for client-server interaction are met. With this, the user/client can then begin preparing their input data for inference by quantizing, encrypting, and serializing the data using the `quantize_encrypt_serialize()` method. After FHE inference has been completed and the results have been sent back to the client, they can view the results of the inference by deserializing, decrypting, and dequantizing the data with the `deserialize_decrypt_dequantize()` method. [61, 64, 65, 66].

3.5 Viral Strain Classification Workflow

Similar to the workflow outlined by Sim et al. and Akavia et al. [24, 48], the general workflow for performing viral strain classification can be divided into the following steps, adjusted to integrate with the Concrete-ML workflow (Section 3.4):

1. The **training and compilation** of the classification model. As discussed in subsection 3.4, this step of the workflow is to be performed on plaintext data. After this, the compilation and saving of the model follow, resulting in the FHE equivalent of the model capable of performing Homomorphic Inference.

2. The **dissemination of prerequisite files to the client for key generation, encryption, and decryption**. The client requests for the prerequisite files which it uses to generate the private and evaluation keys. Additional required files will also be

requested by the client from the server. These include the Dashing binary releases and accompanying shell scripts for CSV file formatting (discussed in the next subsection), `features_and_classes.txt`, which hosts the results of the feature selection during training that allows the client to drop non-selected columns, and the original class labels for translating the classifier's outputs back into its text labels.

3. **Preprocessing, quantization, encoding, and encryption** of the input data. The client preprocesses the data using Dashing, which is discussed in subsection 3.6. After preprocessing, the data is then quantized, encrypted, and serialized using Concrete-ML's built-in API.

4. Lastly, **Homomorphic Inference** on the prepared data using the compiled model is performed. The results of the inference are then sent back to the client, who deserializes, decrypts, and dequantizes the data using the Concrete-ML API to view the results.

3.6 The Dashing Preprocessing Tool

Dashing is a software tool for estimating similarities of genomes or sequencing datasets [23]. The similarity estimation is performed by splitting a provided genome sequence into k-mers and converting each k-mer into a 64-bit hash. Dashing also uses the HyperLogLog sketch, which is an approximate counting method in $O(\log_2 \log_2(n))$ space that estimates a count by incrementing counters with exponentially decaying probability [23, 67], alongside other cardinality estimation methods (which are specialized for set unions and intersections) to estimate the cardinality of the hashed genomes. The tool is able to summarize genomes more rapidly than previous MinHash-based methods while providing greater accuracy across a wide range of input sizes and sketch sizes [23]. Sim et al. [24] utilized this tool to allow for a layer of abstraction from the raw sequence data, thereby reducing the dimensionality of the data and allowing easier processing of it. The tool is accessible via the following GitHub link: `https://github.com/dnbaker/dashing` and can be downloaded in the form of a binary release. These binary releases work only on specific instruction sets, and are the following:

- `dashing_s128`, which works on systems supporting the 128-bit SSE2 SIMD instruction set
- `dashing_s256`, which works on systems that support the AVX2 256-bit SIMD instruction set
- `dashing_s512`, which runs on systems that feature the AVX512BW 512-bit SIMD instruction set

If a binary release does not work on a given system, then a release with a lower number should be used.

3.7 Logistic Regression

Logistic Regression is a type of statistical approach that is used quite often in predictive analysis. It is also used extensively when performing binary or multi-class classification and is also relatively simple to implement. [68, 69] Chang [70] noted that logistic regression is relatively fast when compared to other supervised classification techniques such as SVM or ensemble methods. While logistic regression suffers to some degree in terms of accuracy as a result of its speed, Pradhan et al. and Zeller et al. note that logistic regression performs well when used with linearly separable classes and classification, potentially mitigating the model's lower accuracy [69, 71]. Zeller et al. used logistic regression in viral strain classification, since the genetic patterns in sequences, which come about as a result of genetic divergence over time, are linearly separable across aligned amino acid positions. This quality lends itself well to the use of logistic regression as well as other supervised machine learning methods, such as random forest classification. Lastly, Sim et al. [24] also used logistic regression from a privacy perspective in their proposed framework for privacy-preserving viral strain classification, noting that the use of logistic regression in the classification of viral strains is more appropriate as the classification approach does not expose the individual data values used to train the model in contrast to models such as k-Nearest Neighbors that do. The model's ease of implementation, speed of classification, and excellent performance when used with linearly separable classes make it the ideal algorithm for use in viral strain classification.

4 Design and Implementation

4.1 Threat Model

Our system follows a semi-honest (honest-but-curious) threat model. That is, we assume that our server (an ML service provider) will stay honest and will adhere to the system's workflow, but is curious about the client's private information. We also assume that all communication channels between the two parties (client and server) are secure. Given this, if the security of the encryption scheme is guaranteed, then there will be no leakage of the client's private data to any potentially malicious parties, including the server. We also assume that the client (the owner of the secret key) does not collude with the server. Our security goals are as follows:

1. The server should not obtain or receive any information on the client's encrypted inputs.
2. The server should not find out any information regarding the encrypted prediction results.

4.2 Dataset

We aim to use data from the publicly available repository Global Initiative on Sharing Avian Influenza Data (GISAID) [72, 73, 74], specifically the data of Sim et al. [24] (Episet ID `EPI_SET_220924cw`, accessible at `https://doi.org/10.55876/gis8.220924cw`). Their dataset of nearly 9000 sequences is comprised primarily of four strains of interest. The strains are the B.1.617.2 (Delta), C.37 (Lambda), B.1.621 (Mu), and B.1.1.529 (Omicron) lineages, and will be made the main focus of the study. This will result in a smaller, unbalanced dataset of 6805 samples. To address this imbalance, we will perform an upsampling of the minority (the B.1.1.529 and the B.1.617.2 classes) through the acquisition of additional samples from GISAID via the GISAIDR library [75].

4.3 Input File Structure

Clients will interact with our viral strain classification through the client-side GUI application, which takes one FASTA file as input for classification. The format for FASTA files is as follows:

1. The file begins with the FASTA definition line, which is included before the nucleotide sequence. It must begin with a carat (">") and should be followed by a unique sequence identifier (SeqID) [76].
    - The FASTA definition line in this sequence should follow the following format: `>Reference/Database|AccessionID|DateCollected`
    - The SeqID should be unique for each nucleotide sequence.
    - The SeqID should not contain any spaces.
    - The SeqID should only contain the following characters: letters, digits, underscores (_), periods (.), asterisks (*), colons (:), hyphens (-), and number signs (#).
    - All the information should be on a single line of text. The FASTA definition line should not include any hard returns.
2. The sequence begins after the FASTA definition line, and can contain returns. NCBI recommends that each line should be no more than 80 characters, and should only contain IUPAC symbols only, with "N" being used to symbolize ambiguous characters [76].

4.4 Preprocessing Techniques and Tools

To improve preprocessing and model training speeds, we will follow Sim et al's framework and truncate the first 20kB of each of the genome sequences in the dataset. The truncated regions will correspond to the regions of the sequences that preceded the S gene, which is the key driver of biological mutations between the SARS-CoV-2 strains [24]. Additionally, due to viral strains being typically defined based on their phenotypic characteristics instead of simple sequence similarity, alignment-free preprocessing methods will be explored and utilized to prepare the viral sequences for strain classification. These methods transform raw genomic sequences into feature vectors, which can then be used as inputs for training classification models. The alignment-free tool that will be used is Dashing [23]. Dashing provides a layer of abstraction from the raw sequence data by splitting an individual sequence into k-mers, converting each k-mer into a 64-bit hash, and then computing the HyperLogLog sketch of the sequence to estimate the cardinality of the hash sets. The output of this process, which will be performed for each sequence in the dataset, is a vector containing the cardinality of 512 hash sets, represented by buckets. These buckets will then be used as the features in training the machine learning model [24].

We aim to use the following Dashing command along with shell scripts to transform the resulting HLL sketches into a CSV file:

```
    ./dashing\_s512 sketch -k31 -p13 -S9 -F path.txt
```

Listing 1: Dashing command to be used to generate the HLL sketches (using Dashing's 512-bit release)

In the Dashing commands we will be using in the study shown in Listing 1, k will be set to 31 using "-k31", the number of threads will be set to 13 using "-p13", and the sketch size set to 9 using "-S9", for 2^9 bytes, with setting the sketch size affecting the number of features generated (a sketch size of 10 results in 1024 features being generated). After using Dashing to convert the individual SARS-COV-2 sequences into feature buckets, the output will be joined with its respective `Lineage` and `Accession ID` values to form a dataset containing the GISAID accession ID, the lineage (also called variant or class) of the sample, and the features generated from Dashing.

4.5 Feature Selection Algorithm

To improve the performance of the logistic regression model, a feature selection algorithm will be used to address the high number of features generated from Dashing the training sequences. Scikit-learn's k-best features algorithm (`sklearn.feature_ selection.SelectK Best`), a univariate feature selection algorithm, will be used to select the k highest scoring features based on univariate statistical tests [77, 78, 79]. This algorithm will be used due to its simplicity and fast running time, as well as its ability to ensure that feature selection results in a set with lower dimensionality. The k value to be used is 20, effectively selecting the 20 highest-scoring features based on statistical tests. The feature selection algorithm will be used to reduce the dimensionality of the dataset from the initial count of 512 features to 20 features.

4.6 Implementation of Classification

Following Sim et al's Covnita framework, a logistic regression classifier will be used as the main classification model for the system due to its speed, simplicity in implementation, good performance with linearly separable data, and privacy advantages as discussed in section 3.7 of Chapter 3.

In terms of training, the `LogisticRegression` implementations of both scikit-learn and Concrete-ML will be developed and trained on the same dataset in order to gauge the performance of the Concrete-ML model in comparison to the scikit-learn model, given that Concrete-ML is said to be compatible with scikit-learn workflows. This will also provide some information regarding the effect of Concrete-ML's quantization process on its accuracy and overall performance, with the scikit-learn model being used as a comparison. A standard train-test split of 80% training data and 20% testing data shall be utilized via scikit-learn's `train_test_split` function, and the univariate feature selection algorithm will be applied to reduce the dimensionality of the dataset to 20 features. The training, development, compilation, and saving of the models are to be performed via the Concrete ML Docker container and Google Colab.

4.7 System Architecture

The following technologies and tools shall be utilized to implement the client and server-side applications:

- **Pygubu and Pygubu-designer** - A "what you see is what you get" (WYSIWYG) GUI designer for the Python's `tkinter` module, as well as `CustomTkinter`

- **Tkinter** - Standard Python interface to the Tk GUI toolkit
- **CustomTkinter** - A Python UI library based on tkinter that provides more modern UI widgets to allow for more up-to-date UI designs
- **Dashing** - Software tool for sketching similarities of genomes or sequencing datasets.
- **pandas** - Python software library for data manipulation and analysis
- **Concrete-ML** - Privacy-preserving FHE machine learning library built on Concrete
- **WSL2** - A Windows compatibility layer for Linux that enables running a Linux terminal environment on Windows without virtualization
- **scikit-learn** - Software machine learning library for the Python programming language
- **Django Framework** - Open source Python-based Model-View-Controller (MVC) web framework
- **Bootstrap** - Open-source CSS front-end development framework for web applications
- **Google Colab** - Cloud-based Jupyter notebook environment

The logistic regression classification model that is stored on the server-side application shall be developed using the `pandas`, `scikit-learn` and `Concrete-ML` libraries on the Google Colab notebook environment. The `pandas` library will be used to parse the dataset used for training into a dataframe for pre-processing and training, while the `scikit-learn` library will provide the modules for feature selection using the univariate feature selection algorithm, splitting the dataset into training and testing sets, and for the scikit-learn logistic regression model, which will be used as a standard for comparison of model performance. Lastly, the `Concrete-M` library will be used to create the system's final logistic regression classification model.

The client-side application will serve as the primary method that clients of the system will use to interact with the system, allowing the client to preprocess, encrypt, and send their SARS-CoV-2 sequences in one action. The app UI shall be implemented using CustomTkinter and Tkinter GUI libraries to develop the user interface of the application, with Pygubu and Pygubu-designer acting as rapid application development tools to allow more efficient UI development. The Dashing tool and `pandas` library will provide its sequence hashing and column selection functionalities, respectively, and the Concrete-ML library will provide the client-side application with encryption and decryption functionality. The app will communicate with the server-side application's API to send the preprocessed and encrypted sequences for prediction and uses its decryption functionalities to view the results of the prediction.

The server-side application will be developed using the Django framework and Bootstrap and provides an API endpoint through which the client application will pass its encrypted inputs and generated keys to the server. The server-side application will then return its encrypted prediction results to the client for decryption.

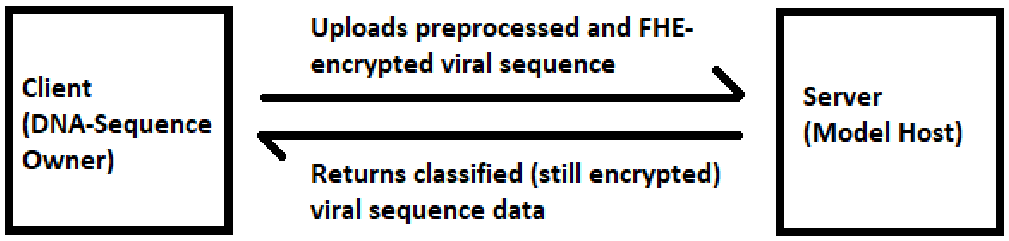
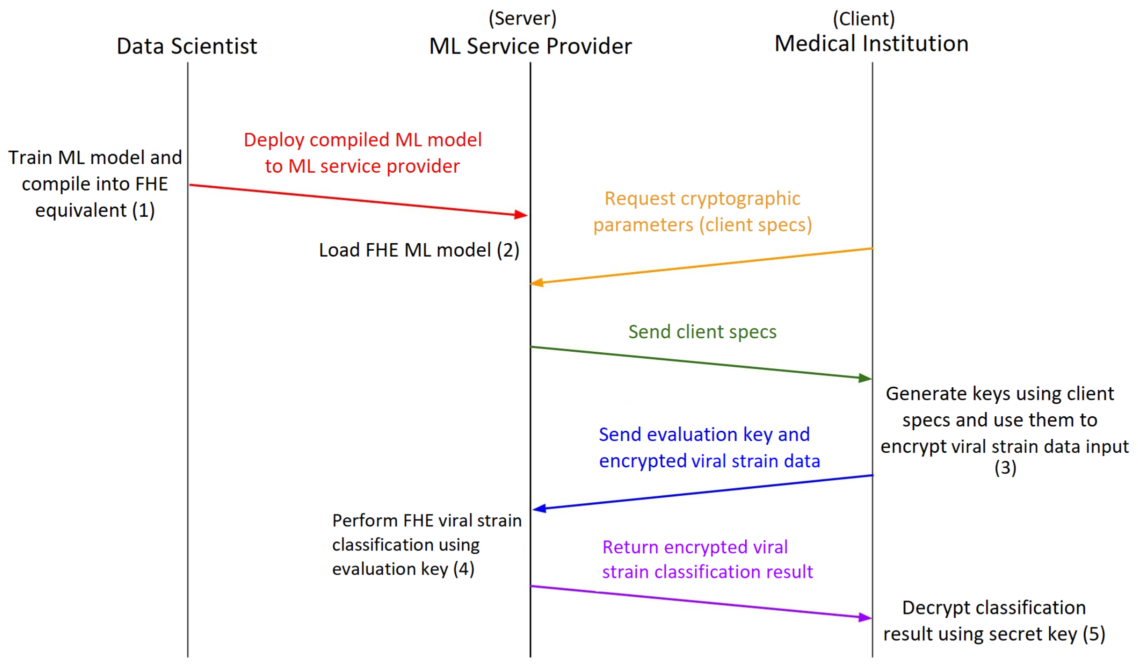

Client (DNA-Sequence Owner)
Uploads preprocessed and FHE-encrypted viral sequence
Returns classified (still encrypted) viral sequence data
Server (Model Host)

Figure 2: The client-server architecture.



(Server)
(Client)
Data Scientist
ML Service Provider
Medical Institution
Train ML model and compile into FHE equivalent (1)
Deploy compiled ML model to ML service provider
Load FHE ML model (2)
Request cryptographic parameters (client specs)
Send client specs
Generate keys using client specs and use them to encrypt viral strain data input (3)
Send evaluation key and encrypted viral strain data
Perform FHE viral strain classification using evaluation key (4)
Return encrypted viral strain classification result
Decrypt classification result using secret key (5)

Figure 3: Detailed workflow of the client-server system.

As shown in Figure 3, the detailed workflow will begin with the data scientists training, compiling, and saving the model into its FHE equivalent, with the last step being performed using the `FHEModelDev.save()` command, as outlined in subsection 3.4 of Chapter 3. Next, the data scientists will store the saved model (`server.zip`) on the server machine, along with the cryptographic parameters (`client.zip`) to allow the server to send them to client machines at their request. In development, these files (along with `versions.json`, also discussed in subsection 3.4 of Chapter 3) will be stored in the `Compiled Model` folder of the Django project directory (`BASE_DIR`), which the server will then access using relative file paths as needed via Python's `os.path.join()` function. The training, saving, and storing process will be similar to the steps taken to store a saved scikit-learn model on a server for a similar use case, with the main difference being that instead of storing only a single file on the server like in traditional scikit-learn workflows, the process will involve storing two files on the server: `client.zip` for dissemination to client machines, and `server.zip` for classification of clients' encrypted sequences. This allows data scientists that are already familiar with scikit-learn workflows to acclimate to Concrete-ML's workflow more easily.

Once the previous step is completed, the client (which represents users such as medical professionals, medical institutions, or researchers) can open the client-side app, which will automatically request the cryptographic parameters (`client.zip`, also discussed in subsection 3.4 of Chapter 3) from the server using the `requests` library. It will also request `features_and_classes.txt` (discussed in subsection 3.5 of Chapter 3) for column filtering. Once the client-side application has ensured that `client.zip` and other required files have been downloaded, the client will then begin preprocessing and encrypting their data using the client-side application, which will abstract the Dashing, evaluation key generation, encryption of the clients' inputs behind a single file input, and filtering of non-selected features and class translation post-prediction (via `features_and_classes.txt`). It will also account for any instruction set incompatibilities for the Dashing tool as discussed in section 3.6 of Chapter 3, ensuring that the correct version of the Dashing tool is utilized in the sequence hashing set. The key generation and encryption steps will be performed using Concrete-ML's API, in particular, through the following commands from the `FHEModelClient` module:

- `generate_private_and_evaluation_keys()` for key generation
- `get_serialized_evaluation_keys()` to access the keys within the program for saving and sending to the server
- `quantize_encrypt_serialize()` to facilitate the encryption of the inputs

The client-side application will also automatically send the encrypted file input to the server for classification along with the generated evaluation keys for classification. This shall be done using Python's `requests` library to interact with the classification endpoint server's API. In development, the classification API endpoint will be `http://localhost:8000/start_classification`, with `localhost:8000` being a locally-run Django-development server. Once the server receives the clients' evaluation keys and encrypted viral strain sequences, it will then perform FHE inference on the sequences using the `FHEModelServer.run()` command. Then, it will save the encrypted predictions to separate files, compress the encrypted predictions to a ZIP file, and send the ZIP file back to the client via the client-side application, which will automatically unpack the ZIP file and use the `FHEModelClient.deserialize` `_decrypt_dequantize()` method to facilitate the decryption of the prediction results using the keys generated earlier in the workflow and the display of the results for the client to view.

4.8 Technical Architecture

The requirements for running the system at full capacity will include the following specifications:

- **Operating System**: Linux or Windows Subsystem for Linux 2
- **Memory**: 4GB minimum
- **Storage space**: 7GB free disk space for Concrete-ML package and dependencies

An estimated 7 gigabytes of free disk space is required to run the full system due to `Concrete-ML`'s dependencies, including `torch` and `concrete`. Additionally, as the client application will require the Dashing tool, the appropriate shell scripts, a text file listing the selected features for classification and original class labels (`features_and_classes.txt`), and the `client.zip` file, which will hold the required cryptographic parameters for key generation, encryption, and decryption, the application will also require an internet connection to download the files needed from the system's server or repository.

5 Expected Output/Timeline

5.1 Overview

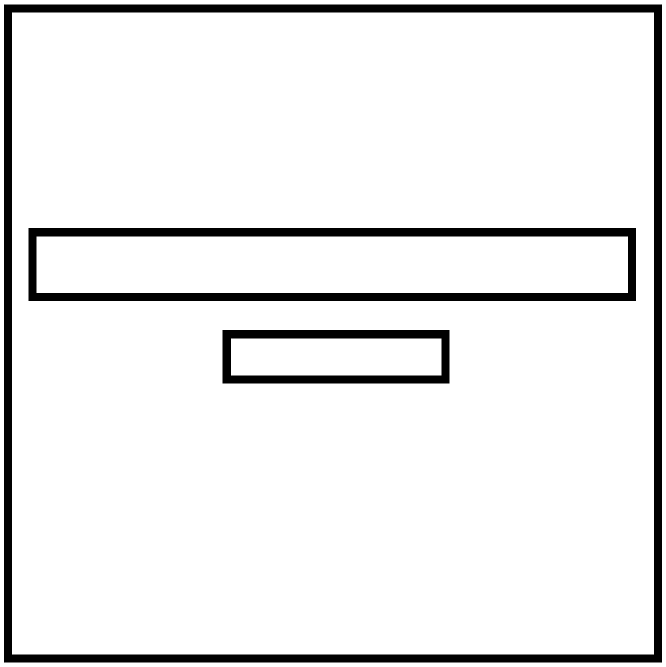


Figure 4: Rough overview of the client-side interface for uploading prepared data to the server for classification.

The expected output of this study is a privacy-preserving viral strain classification model that is implemented using Concrete-ML and utilizes a client-server setup to allow the client to interact with the classification model by sending their prepared input data for the classification services. The author also aims to obtain insights into the usefulness and performance of the ML-specialized Concrete-ML FHE library compared to other more general-purpose FHE libraries such as Microsoft SEAL and HElayers with regard to viral strain classification.

5.2 Schedule of Activities

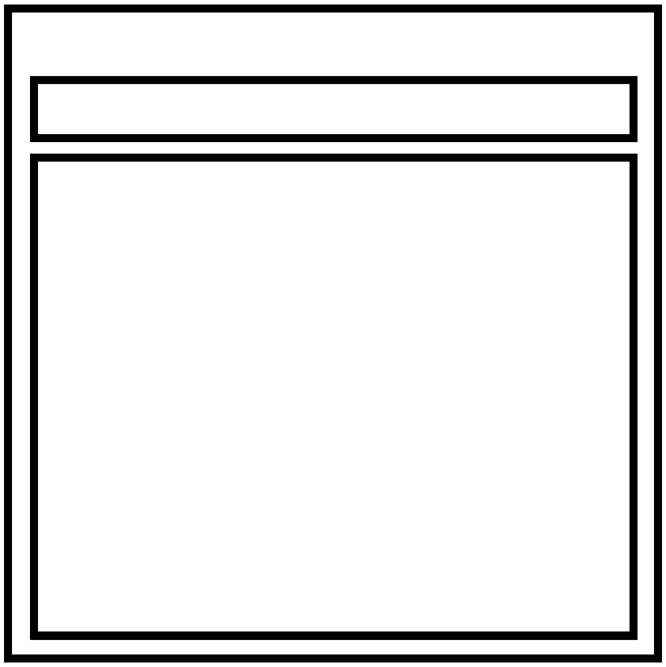


Figure 5: Overview of the server-side interface, with window to display server logs.

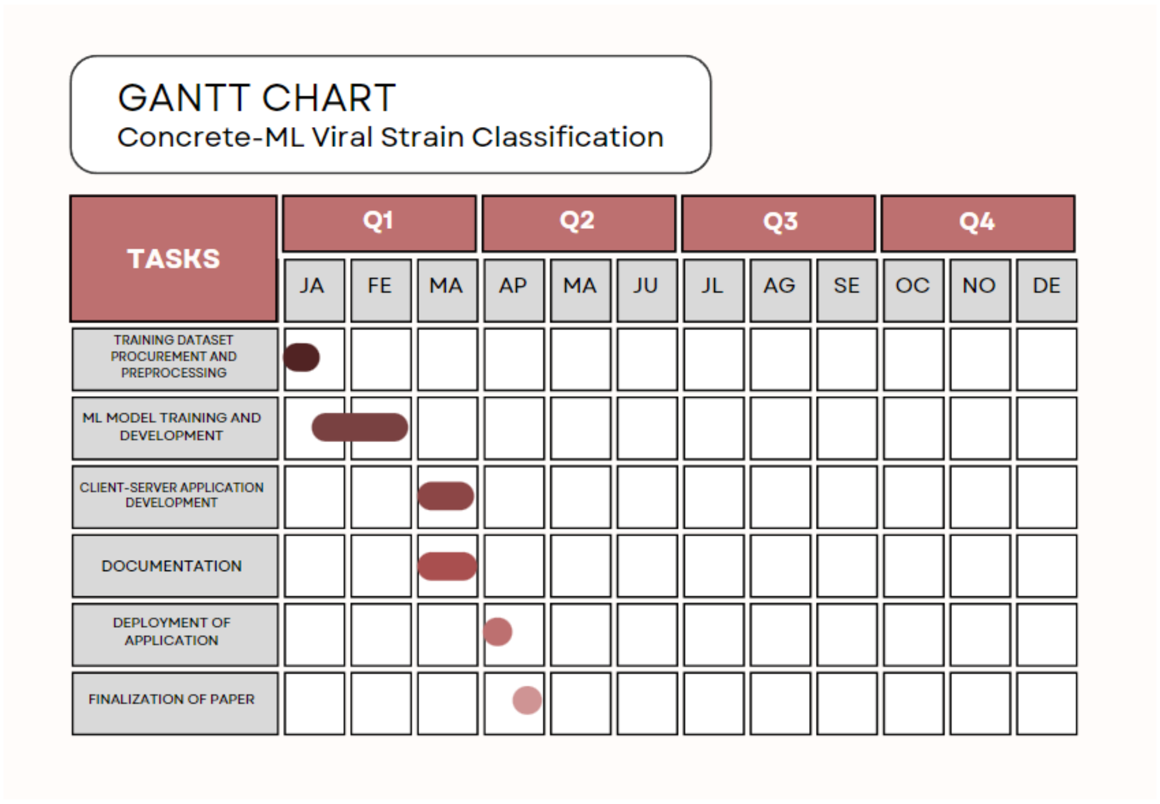


Figure 6: The proposed timeline for the study

6 References

[1] D. Tobin, “What is data privacy-and why is it important?,” *Integrate.io*, May 2021.

[2] “What is data privacy? — privacy definition — Cloudflare,” *Cloudflare*.

[3] “Data protection in the EU,” *European Commission - European Commission*, Oct 2022.

[4] “Data privacy act primer,” *National Privacy Commission*, Oct 2022.

[5] L. Song, H. Liu, F. S. Brinkman, E. Gill, E. J. Griffiths, W. W. Hsiao, S. Savić-Kallesøe, S. Moreira, G. Van Domselaar, M. H. Zawati, and et al., “Addressing privacy concerns in sharing viral sequences and minimum contextual data in a public repository during the covid-19 pandemic,” *Frontiers in Genetics*, vol. 12, 2022.

[6] “Gordon v. Canada (Health), 2008 FC 258,” *Office of the Privacy Commissioner of Canada*, Jun 2014.

[7] “Gordon v. Canada (Minister of Health), 2008 FC 258,” *vLex*.

[8] L. Sweeney, “Simple demographics often identify people uniquely,” *figshare*, Jun 2018.

[9] P. Golle, “Revisiting the uniqueness of simple demographics in the us population,” *Proceedings of the 5th ACM workshop on Privacy in electronic society - WPES ’06*, 2006.

[10] L. Rocher, J. M. Hendrickx, and Y.-A. de Montjoye, “Estimating the success of re-identifications in incomplete datasets using generative models,” *Nature Communications*, vol. 10, no. 1, 2019.

[11] K. J. Wu, “A major clue to covid’s origins is just out of reach,” Mar 2023.

[12] “Sago statement on newly released sars-cov-2 metagenomics data from china cdc on gisaid,” Mar 2023.

[13] B. Mole, “Genetic data links sars-cov-2 to raccoon dogs in china market, scientists say,” Mar 2023.

[14] K. Cullinan, “High drama as scientists who may have found covid ’animal x’ are kicked off data-sharing platformnbsp;,” Mar 2023.

[15] S. LEHRER and P. H. RHEINSTEIN, “Human gene sequences in sars-cov-2 and other viruses,” *In Vivo*, vol. 34, p. 1633–1636, Jun 2020.

[16] H. Li, X. Hong, L. Ding, S. Meng, R. Liao, Z. Jiang, and D. Liu, “Sequence similarity of sars-cov-2 and humans: Implications for sars-cov-2 detection,” *Frontiers in Genetics*, vol. 13, Jul 2022.

[17] M. Lenharo, “Gisaid in crisis: Can the controversial covid genome database survive?,” May 2023.

[18] R. Van Noorden, “Scientists call for fully open sharing of coronavirus genome data,” Feb 2021.

[19] A. Aloufi, P. Hu, Y. Song, and K. Lauter, “Computing blindfolded on data homomorphically encrypted under multiple keys: A survey,” *ACM Computing Surveys*, vol. 54, p. 1–37, Dec 2022.

[20] M. Chase, H. Chen, J. Ding, S. Goldwasser, S. Gorbunov, J. Hoffstein, K. Lauter, S. Lokam, D. Moody, T. Morrison, and et al., “Security of homomorphic encryption - microsoft.com,” Jan 2018.

[21] J. A. Cruz and R. Chua, “Secure remote genome-wide association studies using fully homomorphic encryption,” *Theory and Practice of Computation Proceedings of the Workshop on Computation: Theory and Practice (WCTP 2019)*, 2020.

[22] *Competition tasks - IDASH privacy security workshop 2021 - secure genome analysis competition*, 2021.

[23] D. N. Baker and B. Langmead, “Dashing: Fast and accurate genomic distances with hyperloglog,” *Genome Biology*, vol. 20, no. 1, 2019.

[24] J. J. Sim, W. Zhou, F. M. Chan, M. S. Annamalai, X. Deng, B. H. Tan, and K. M. Aung, “CoVnita: An End-to-end Privacy-preserving Framework for SARS-CoV-2 Clas-

sification," *CoVnita: An End-to-end Privacy-preserving Framework for SARS-CoV-2 Classification PREPRINT (Version 1)*, Nov 2022.

[25] M. A. Remita, A. Halioui, A. A. Malick Diouara, B. Daigle, G. Kiani, and A. B. Diallo, "A machine learning approach for viral genome classification," *BMC Bioinformatics*, vol. 18, no. 1, 2017.

[26] J. Kim, K. Lee, R. Rupasinghe, S. Rezaei, B. Martínez-López, and X. Liu, "Applications of machine learning for the classification of porcine reproductive and respiratory syndrome virus sublineages using amino acid scores of ORF5 gene," *Frontiers in Veterinary Science*, vol. 8, 2021.

[27] A. Lopez-Rincon, A. Tonda, L. Mendoza-Maldonado, D. G. Mulders, R. Molenkamp, C. A. Perez-Romero, E. Claassen, J. Garssen, and A. D. Kraneveld, "Classification and specific primer design for accurate detection of SARS-COV-2 using Deep Learning," *Scientific Reports*, vol. 11, no. 1, 2021.

[28] G. B. Câmara, M. G. Coutinho, L. M. Silva, W. V. Gadelha, M. F. Torquato, R. d. Barbosa, and M. A. Fernandes, "Convolutional neural network applied to SARS-COV-2 sequence classification," *Sensors*, vol. 22, no. 15, p. 5730, 2022.

[29] D. S. Char, N. H. Shah, and D. Magnus, "Implementing machine learning in health care — addressing ethical challenges," *New England Journal of Medicine*, vol. 378, no. 11, p. 981–983, 2018.

[30] D. S. Char, M. D. Abràmoff, and C. Feudtner, "Identifying ethical considerations for Machine Learning Healthcare Applications," *The American Journal of Bioethics*, vol. 20, no. 11, p. 7–17, 2020.

[31] J. Li, X. Kuang, S. Lin, X. Ma, and Y. Tang, "Privacy preservation for machine learning training and classification based on homomorphic encryption schemes," *Information Sciences*, vol. 526, p. 166–179, Jul 2020.

[32] A. Akavia, B. Galili, H. Shaul, M. Weiss, and Z. Yakhini, “Efficient privacy-preserving viral strain classification via K-Mer signatures and FHE,” *IBM Research Publications*, May 2022.

[33] M. Templ and M. Sariyar, “A systematic overview on methods to protect sensitive data provided for various analyses,” *SpringerLink*, Aug 2022.

[34] A. Benaissa, B. Retiat, B. Cebere, and A. E. Belfedhal, “TenSEAL: A library for encrypted tensor operations using homomorphic encryption,” *arXiv.org*, Apr 2021.

[35] A. Ibarrondo and A. Viand, “Pyfhel: PYthon For Homomorphic Encryption Libraries,” *Proceedings of the 9th on Workshop on Encrypted Computing Applied Homomorphic Cryptography*, p. 11–16, Nov 2021.

[36] R. L. Rivest, A. Shamir, and L. Adleman, “A method for obtaining digital signatures and public-key cryptosystems,” *Communications of the ACM*, vol. 21, no. 2, p. 120–126, 1978.

[37] Microsoft, “Microsoft/SEAL,” *GitHub*, Mar 2022. Available at `https://github.com/microsoft/SEAL`.

[38] Openfheorg, “Openfheorg/openfhe-development,” *GitHub*, Nov 2022. Available at `https://github.com/openfheorg/openfhe-development`.

[39] Zama-AI, “Concrete-ML,” *GitHub*, Nov 2022. Available at `https://github.com/zama-ai/concrete-ml`.

[40] D. Murik, A. Bitar, and S. Martinelli, “IBM/HElayers: IBM HElayers Homomorphic Encryption SDK for C++ and Python,” *GitHub*, Nov 2022. Available at `https://github.com/IBM/helayers`.

[41] T.-T. Kuo, T. Bath, S. Ma, N. Pattengale, M. Yang, Y. Cao, C. M. Hudson, J. Kim, K. Post, L. Xiong, and et al., “Benchmarking blockchain-based gene-drug interaction data sharing methods: A case study from the iDASH 2019 secure genome analysis competition blockchain track,” *International Journal of Medical Informatics*, vol. 154, p. 104559, 2021.

[42] A. K. Ladisla and R. B. Chua, “Theory and practice of computation,” *Theory and Practice of Computation Proceedings of the Workshop on Computation: Theory and Practice (WCTP 2018)*, 2019.

[43] E. Sarkar, E. Chielle, G. Gursoy, O. Mazonka, M. Gerstein, and M. Maniatakos, “Fast and scalable private genotype imputation using machine learning and partially homomorphic encryption,” *IEEE Access*, vol. 9, p. 93097–93110, 2021.

[44] D. Froelicher, J. R. Troncoso-Pastoriza, J. L. Raisaro, M. A. Cuendet, J. S. Sousa, H. Cho, B. Berger, J. Fellay, and J.-P. Hubaux, “Truly privacy-preserving federated analytics for precision medicine with multiparty homomorphic encryption,” *Nature Communications*, vol. 12, no. 1, 2021.

[45] K. Lauter, S. Kannepalli, K. Laine, and R. C. Moreno, “Password Monitor: Safeguarding Passwords in Microsoft Edge,” *Microsoft Research*, Jan 2021.

[46] IBM *Efficient Privacy-Preserving Viral Strain Classification via k-mer Signatures and FHE (Poster)*, 2022. Available at `https://drive.google.com/file/d/1hAcn_DZPQ5KA837JVoMkMwg__W4FtESA/view`.

[47] H. Shaul, B. Galili, A. Akavia, M. Weiss, and Z. Yakhini, “IBM homomorphic encryption: A DASHing solution for healthcare data privacy,” *IBM Developer*, Feb 2022.

[48] A. Akavia, B. Galili, H. Shaul, M. Weiss, and Z. Yakhini, “Efficient privacy-preserving viral strain classification via K-Mer signatures and FHE,” *Cryptology ePrint Archive*, Jan 2023.

[49] M. Creeger and C. Inc., “The rise of fully homomorphic encryption,” *The Rise of Fully Homomorphic Encryption - ACM Queue*, Sep 2022.

[50] O. Regev, “Lattice-based cryptography,” *Lecture Notes in Computer Science*, p. 131–141, 2006.

[51] R. L. Rivest, L. Adleman, and M. L. Dertouzos, “On data banks and privacy homomorphisms,” *Foundations of secure computation*, vol. 4, no. 11, p. 169–180, 1978.

[52] C. Gentry, "Fully homomorphic encryption using ideal lattices," *Proceedings of the 41st annual ACM symposium on Symposium on theory of computing - STOC '09*, 2009.

[53] C. Gentry and S. Halevi, "Implementing Gentry's fully-homomorphic encryption scheme," *Advances in Cryptology – EUROCRYPT 2011*, p. 129–148, 2011.

[54] M. van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan, "Fully homomorphic encryption over the integers," *https://eprint.iacr.org/*, 2010.

[55] I. Chillotti, M. Joye, and P. Paillier, "Programmable bootstrapping enables efficient homomorphic inference of deep neural networks," *whitepaper.zama.ai*, 2020.

[56] Zama-AI, "Linear models," *Concrete ML.* Available at `https://docs.zama.ai/concrete-ml/built-in-models/linear`.

[57] Zama-AI, "Tree-based models," *Concrete ML.* Available at `https://docs.zama.ai/concrete-ml/built-in-models/tree`.

[58] Zama-AI, "Neural networks," *Concrete ML.* Available at `https://docs.zama.ai/concrete-ml/built-in-models/neural-networks`.

[59] Zama-AI, "Key concepts," *Concrete ML.* Available at `https://docs.zama.ai/concrete-ml/getting-started/concepts`.

[60] Zama-AI, "Quantization," *Concrete ML.* Available at `https://docs.zama.ai/concrete-ml/advanced-topics/quantization`.

[61] Zama-AI, "Compilation," *Concrete ML.* Available at `https://docs.zama.ai/concrete-ml/advanced-topics/compilation`.

[62] Zama-AI, "Quick start," *Concrete ML.* Available at `https://docs.zama.ai/concrete-numpy/getting-started/quick_start`.

[63] Zama-AI, *Summary of the overall communications protocol to enable cloud deployment of machine learning services.* Zama-AI, Dec 2022.

[64] Zama-AI, "Production deployment," *Concrete ML.* Available at `https://docs.zama.ai/concrete-ml/advanced-topics/client_server`.

[65] Zama-AI, "Concrete.ml.deployment.fhe_client_server," *Concrete ML.* Available at `https://docs.zama.ai/concrete-ml/developer-guide/api/concrete.ml.deployment.fhe\_client\_server`.

[66] Zama-AI, "Concrete-ml/clientserver.ipynb at release/0.5.x · zama-ai/concrete-ml," *GitHub*, Sep 2022. Available at `https://github.com/zama-ai/concrete-ml/blob/release/0.5.x/docs/advanced_examples/ClientServer.ipynb`.

[67] P. Flajolet, Fusy, O. Gandouet, and F. Meunier, "Hyperloglog: The analysis of a near-optimal cardinality estimation algorithm," *Discrete Mathematics amp;amp; Theoretical Computer Science*, vol. DMTCS Proceedings vol. AH,..., no. Proceedings, 2007.

[68] IBM, "What is logistic regression?," *IBM.* Available at `https://www.ibm.com/topics/logistic-regression`.

[69] 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," *MDPI*, Jul 2022.

[70] A. C. Chang, "Machine and deep learning," *Intelligence-Based Medicine*, Aug 2020.

[71] M. A. Zeller, Z. W. Arendsee, G. J. Smith, and T. K. Anderson, "Classlog: Logistic regression for the classification of genetic sequences," *bioRxiv*, Jan 2022.

[72] S. Khare, C. Gurry, L. Freitas, M. B. Schultz, G. Bach, A. Diallo, N. Akite, J. Ho, R. TC Lee, W. Yeo, and et al., "GISAID's role in pandemic response," *China CDC Weekly*, vol. 3, no. 49, p. 1049–1051, 2021.

[73] S. Elbe and G. Buckland-Merrett, "Data, disease and diplomacy: GISAID's innovative contribution to global health," *Global Challenges*, vol. 1, no. 1, p. 33–46, 2017.

[74] Y. Shu and J. McCauley, "GISAID: Global initiative on sharing all influenza data – from vision to reality," *Eurosurveillance*, vol. 22, no. 13, 2017.

[75] W. Wirth and S. Duchene, "Wytamma/gisaidr: Programmatically interact with the gisaid database.," 2022.

[76] Feb 2021.

[77] 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.

[78] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux, "API design for machine learning software: experiences from the scikit-learn project," in *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pp. 108–122, 2013.

[79] J. K. E. Yee, "Brief introduction to four categories of feature selection techniques," Jan 2021.