Journal of Electrical Engineering and Computer (JEECOM)
Vol. 7, No. 1 (2025), DOI: 10.33650/jeecom.v4i2
p-ISSN: 2715-0410; e-ISSN: 2715-6427

`Enhancing Medical Data Privacy: Neural Network Inference with Fully Homomorphic Encryption

Maulyanda [1], Rini Deviani [2], Afdhaluzzikri [3]
[1,2,3] Department of Informatics, Universitas Syiah Kuala, Banda Aceh, Indonesia

Article Info

Article history:

Received March 25, 2025
Revised March 26, 2025
Accepted April 13, 2025

Keywords:

Fully homomorphic
Encryption
Data privacy
Neural network inference

ABSTRACT

Protecting the privacy of medical data while enabling sophisticated data analysis is a critical challenge in modern healthcare. Fully Homomorphic Encryption (FHE) emerges as a powerful solution, enabling computations to be performed directly on encrypted data without exposing sensitive information. This study delves into the use of FHE for neural network inference in medical applications, investigating its role in safeguarding patient confidentiality while ensuring computational accuracy and efficiency. Experimental findings confirm the practicality of using FHE for medical data classification, demonstrating that data security can be preserved without significant loss of performance. Furthermore, the research explores the balance between computational overhead and model precision, shedding light on the complexities of deploying FHE in real-world healthcare AI systems. By emphasizing the significance of privacy-preserving machine learning, this work contributes to the development of secure, ethical, and effective AI-driven medical solutions.

Corresponding Author:
Maulyanda,
Universitas Syiah Kuala, Jln. Teuku Nyak Arief Darussalam, Banda Aceh, 23111, Indonesia
Email: maulyanda@usk.ac.id

1. INTRODUCTION

With the rapid advancement of medical technology, medical analysis has become vital in modern healthcare. Through methods like computed tomography (CT), magnetic resonance imaging (MRI), ultrasound, and pathology, clinicians can diagnose diseases more accurately, assess treatment effectiveness, and even predict potential health risks [1]. Artificial intelligence (AI) and deep learning (DL) have shown significant potential in medical image analysis, driving advancements in tumor detection, organ segmentation, pathology, and more [2]. These developments not only speed up data processing but also ease the workload of healthcare professionals, helping to overcome the shortage of specialized medical personnel.

Medical data is highly sensitive and requires robust protection to ensure patient privacy [3]. Unauthorized access to data during transmission, storage, or analysis can result in severe privacy breaches or identity theft. With the growing exchange of medical data, particularly in cross-institutional collaborations and AI model training, these risks have become increasingly critical. Conventional encryption techniques safeguard data while stored and transmitted but are inadequate when computations are required. Fully Homomorphic Encryption (FHE) overcomes this limitation by allowing direct computations on encrypted data. This capability is particularly valuable in the context of neural networks, where large-scale data analysis is necessary for predictive modeling, diagnostics, and personalized medicine.

This article focuses on the integration of FHE with neural network inference in the medical domain, aiming to optimize the efficiency of encrypted computations while maintaining high model accuracy. Unlike previous studies that primarily explored the feasibility of FHE for basic arithmetic operations in neural networks, our research emphasizes optimizing inference performance to make privacy-preserving medical AI more practical. We specifically investigate computational overhead, accuracy trade-offs, and potential techniques to enhance inference efficiency under FHE constraints.

The intersection of FHE and machine learning has gained significant attention in recent years. Early research focused on the feasibility of performing basic arithmetic operations on encrypted data. As FHE schemes have matured, researchers have explored more complex computations, including those required for machine learning algorithms. Several studies have demonstrated the potential of FHE in secure neural network training and inference. For instance, early implementations successfully applied FHE to train small neural networks on encrypted data, proving that privacy can be preserved without significantly sacrificing model accuracy [4]. More recent work by Bourse et al. (2018) has focused on optimizing the efficiency of FHE schemes, making them more suitable for real-world applications.

In the medical domain, FHE has been explored for tasks such as secure genomic data analysis and privacy-preserving diagnostics. However, the integration of FHE with neural networks for medical data inference remains an emerging area of research. Our work builds upon existing literature by systematically evaluating the impact of FHE on inference performance in medical AI applications, identifying key bottlenecks, and proposing optimizations to improve efficiency. Through this approach, we provide insights into how FHE can be leveraged to secure medical data while ensuring practical usability in machine learning-based diagnostics and predictive analytics.

2. METHOD

2.1. Homomorphic Encryption

Homomorphic encryption enables computations to be performed directly on encrypted data without requiring decryption [5], allowing third parties to process information while maintaining both privacy and utility. The encryption process is the process of changing the letters of plaintext data or information into ciphertext [6], the fundamental concept is that operations on encrypted data yield the same results as if performed on unencrypted data. There are two main types: Partial Homomorphic Encryption (PHE), which supports either addition or multiplication on encrypted data, and Fully Homomorphic Encryption (FHE), which allows both operations in any combination. In medical analysis, homomorphic encryption is particularly valuable for privacy-preserving AI model training [7], enabling institutions to collaborate on machine learning tasks without exposing sensitive patient information. This ensures secure training on encrypted data.

The primary benefit of homomorphic encryption lies in its ability to securely process encrypted data, making it particularly well-suited for analyzing confidential medical images. However, FHE is computationally demanding and slow, particularly when dealing with large datasets, which can reduce the efficiency of model training. On the other hand, PHE offers better performance but is limited to a single operation, restricting its applicability to more complex tasks. Due to these constraints, homomorphic encryption is not ideal for real-time medical image processing.

FHE is applied to Neural Network inference by enabling computation on encrypted medical data without requiring decryption. The process involves several key steps:

1. Data Encryption: Medical data are encrypted using an FHE scheme before being sent to the processing server.
2. Encrypted Data Processing: The encrypted data is fed into an Artificial Neural Network (ANN), where each layer performs computations directly on ciphertexts using homomorphic operations.
3. Inference on Encrypted Data: The model processes the encrypted inputs through activation functions and weighted connections to generate predictions in encrypted form.
4. Decryption of Results: The encrypted inference output is returned to obtain the final prediction.

This method ensures that sensitive medical data remains secure throughout the inference process, mitigating risks associated with unauthorized access.

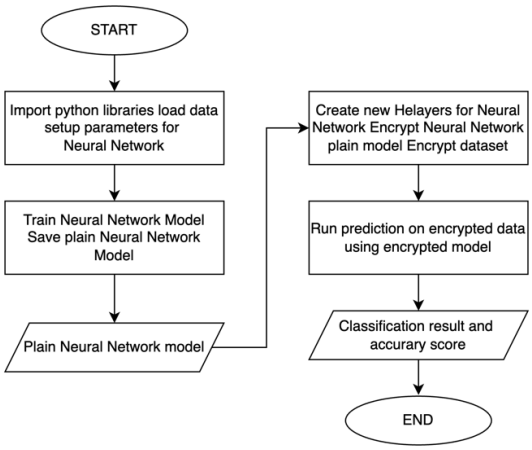

Figure 1. FHE in Neural Network Classification model flowchart

2.2. Neural Network Architecture

2.2.1. Artificial Neural Networks (ANNs)

Artificial Neural Networks (ANNs) have evolved from simple perceptron to deep architectures capable of learning complex patterns [8]. The provided architecture follows a sequential model, a commonly used structure in deep learning frameworks such as TensorFlow and Keras. The network consists of three dense layers, which map input features to outputs using trainable weight matrices and bias terms. Similar architectures have been employed for binary classification problems, particularly in medical diagnosis, fraud detection, and sentiment analysis [9].

2.2.2. Activation functions

Activation functions play a crucial role in introducing non-linearity into neural networks, enabling them to learn complex representations. Traditional activation functions include Sigmoid, ReLU, and Tanh [10]. The architecture in question employs a custom activation function, Square Activation, which squares the input values. The Square Activation function is relatively uncommon, but squaring the inputs can amplify the differences between positive and negative values, potentially enhancing learning in specific scenarios. Previous studies have shown that polynomial activation functions can be effective in approximation tasks [11].

2.2.3. Loss Functions

Binary cross-entropy is a widely used loss function for classification problems with two distinct classes. It is defined as:

$$L = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\hat{y}_i) + (1 - y_i)\ \log(1 - \hat{y}_i)]$$

Where y_i represents the true labels and $\hat{y}_i$ denotes the predicted probabilities. This function is effective in handling imbalanced datasets when combined with proper weighting techniques.

2.2.4. Optimization Techniques

Optimization algorithms such as Stochastic Gradient Descent (SGD), Adam, and RMSprop significantly impact the training efficiency of neural networks. The given architecture allows for a flexible optimizer selection, which is critical for adjusting learning rates dynamically. Learning rate scheduling can further improve model performance by preventing overshooting or slow convergence [12].

3. RESULTS AND DISCUSSION

This experiment makes use of HElayers, a specialized software framework designed for secure and privacy-preserving computations. It functions entirely as a software-based solution and runs within a Docker container on a Linux platform. The framework is implemented in C++ that provides a Python API, making it accessible to developers and data scientists. By leveraging HElayers, users can seamlessly incorporate

advanced privacy-preserving techniques into their applications. This allows for efficient and secure data processing while maintaining ease of use within a Python environment [13].

To investigate the potential of FHE in neural network inference for medical data, we set up an experiment involving the following key components:

1. Data Selection: A dataset of anonymized medical records, focusing on diagnostic information, was chosen for the experiment. To validate the applications of the FHE in Neural Network Inference [14], we focused on one use case of Predictive Modelling and Health Analytics [15]. Detailed dataset comprising health and demographic data of 100,000 individuals, aimed at facilitating diabetes-related research and predictive modelling. This dataset includes information on gender, age, location, race, hypertension, heart disease, smoking history, BMI, HbA1c level, blood glucose level, and diabetes status.
2. Data Preprocessing: We remove several irrelevant attribute named: location. We also change the categorical attribute smoking_history from categorical into numerical. Named: Never (0), No Info (1), Not Current/Ever/Former (2), and Current (3).
3. Neural Network Architecture: A feedforward neural network with multiple hidden layers was selected to perform inference tasks on the medical data. This experiment demonstrates a use case in the medical domain as well as demonstrating encrypted machine learning. We will demonstrate how we can use FHE along with neural networks (NN) to carry out predictions for diabetic detection while keeping the data, the NN model and the prediction results encrypted at all times.
4. Encryption Scheme: The CKKS (Cheon-Kim-Kim-Song) scheme, a popular FHE method optimized for approximate arithmetic, was employed to encrypt the medical data.
5. Inference Process: The encrypted medical data was fed into the neural network, which performed inference without decrypting the data. The results were then decrypted to evaluate the performance of the model.
6. Performance Metrics: Accuracy, computation time, and encryption overhead were measured to assess the feasibility and efficiency of the approach.

3.1. Generate NN model

Table 1. Importing Libraries and Setting Seed

Importing Libraries and Setting Seed
SET seed_value = 1
SET environment variable PYTHONHASHSEED to seed_value
INITIALIZE random seed using `random.seed(seed_value)`
INITIALIZE numpy seed using `np.random.seed(seed_value)`
INITIALIZE tensorflow seed using `tf.random.set_seed(seed_value)`
IMPORT required libraries: TensorFlow, Keras, Pandas, Sklearn, h5py, custom utils
DEFINE data path
IF data path does not exist, CREATE directory
SET training parameters:
epochs = 3
batch_size = 32
optimizer = Adam
learning_rate = 0.01

The first section ensures that the entire process is reproducible by setting a fixed seed value for Python's random module, NumPy, and TensorFlow. This prevents variations in results caused by random initialization. The script then imports various libraries, including TensorFlow for deep learning, Pandas for data handling, and Scikit-learn for preprocessing and model evaluation. Additionally, it sets up a directory to store data if it doesn't already exist. Training parameters such as the number of epochs, batch size, optimizer type, and learning rate are also defined in this section. The setup is completed with a print statement confirming successful initialization.

Table 2. Load and Preprocess Dataset

Load and Preprocess Dataset
LOAD dataset from CSV file PRINT number of samples EXTRACT features (X) and labels (Y) NORMALIZE features using `preprocessing.normalize` SPLIT dataset into training (67%) and test (33%) sets SPLIT training set further into training (67%) and validation (33%) sets

This section reads a dataset from a CSV file located in the specified directory. It extracts the feature matrix (`X`) and target labels (`Y`), where `X` consists of the first 14 columns (excluding the first identifier column), and `Y` represents the diabetes diagnosis. After extracting the data, the feature values are normalized to ensure they are on a similar scale, which helps in stabilizing the learning process. The dataset is then split into training and testing sets, with 67% allocated for training and 33% for testing. The training set is further divided into training and validation subsets to fine-tune the model.

Table 3. Handle Class Imbalance

Handle Class Imbalance
DEFINE function `replicate_smallest_class(x, y, class_id)` EXTRACT minority class samples REPLICATE minority class multiple times SHUFFLE dataset CALL function to balance training data ADJUST dataset size to be a multiple of batch_size # Reshape Labels RESHAPE y_train, y_val, y_test to have an extra dimension

Since many real-world medical datasets suffer from class imbalance, where one category (e.g., diabetic cases) is underrepresented, this section addresses the issue. A function named replicate_smallest_class is defined to artificially expand the minority class by duplicating its samples multiple times. After replication, the data is shuffled to maintain randomness and prevent overfitting to repeated patterns. The training dataset is then adjusted so that its size remains a multiple of the batch size, ensuring efficient batch processing during model training. Finally, the script prints the shapes of the modified datasets to confirm the changes. Neural networks often require target labels to be in a specific shape. This section reshapes the labels (y_train, y_val, and y_test) into a column vector format to match the expected input format for model training. Reshaping ensures that the model correctly interprets the labels during the learning process. A print statement indicates that the training data is fully prepared.

Table 4. Save Processed Data

Save Processed Data
DEFINE function ‘save_data_set(x, y, data_type)’ SAVE x and y datasets as HDF5 files CALL function to save test dataset

To avoid reprocessing the dataset each time the script runs, the processed data is stored in HDF5 files. A function called save_data_set is defined to save both features (x) and labels (y) into separate HDF5 files under the specified directory. The function is then called to save the test dataset, ensuring that future experiments can directly load the prepared data without repeating the preprocessing steps.

3.2. Neural Network encryption

Table 5. Import library and define dataset & model

Import library and define dataset & model
Import necessary utilities and verify memory usage IMPORT utils CALL utils.verify_memory()

```
# Define dataset paths based on whether pre-prepared data should be loaded
SET INPUT_DIR = 'data/diabetic/'

# Define file paths for dataset and model
SET X_H5 = INPUT_DIR / 'x_test.h5'
SET Y_H5 = INPUT_DIR / 'y_test.h5'
SET MODEL_JSON = INPUT_DIR / 'model.json'
SET MODEL_H5 = INPUT_DIR / 'model.h5'

# Load test samples
SET batch_size = 4096
(plain_samples, labels) = utils.extract_batch_from_files(X_H5, Y_H5, batch_size, 0)
PRINT 'Loaded samples of shape', plain_samples.shape
```

The first step in the notebook imports necessary utility functions and verifies memory availability using utils.verify_memory(). This check ensures that the system has enough resources to handle large datasets and neural network computations efficiently. Given that medical datasets can be large, memory optimization is crucial to avoid crashes or performance issues. The script includes a conditional section to determine the source of the dataset. The notebook loads data from a predefined (data/diabetic/). The dataset consists of test samples stored in .h5 files (x_test.h5 for input features and y_test.h5 for labels). Additionally, the pre-trained neural network model is stored in JSON format (model.json for architecture) and H5 format (model.h5 for weights). These file paths are defined as variables to simplify later access.

The test dataset is loaded using a utility function, utils.extract_batch_from_files(), which extracts a batch of samples and corresponding labels from the .h5 files. A batch size of 4096 is defined, meaning the function loads 4096 test samples in one go. This batch-based processing is efficient and reduces memory overhead compared to loading the entire dataset at once.

Table 6. Initialize and load a plain (unencrypted) neural network model

Initialize and load a plain (unencrypted) neural network model
IMPORT pyhelayers
SET hyper_params = pyhelayers.PlainModelHyperParams()
SET neural_net_plain = pyhelayers.NeuralNetPlain()
CALL neural_net_plain.init_from_files(hyper_params, [MODEL_JSON, MODEL_H5])
PRINT 'neural_net_plain created and initialized'

A plain (unencrypted) neural network is initialized using pyhelayers.NeuralNetPlain(). This neural network model serves as a reference, allowing comparisons between encrypted and unencrypted predictions. The model is loaded from the pre-trained JSON and H5 files. At this stage, the model is fully operational but does not yet support encrypted computations.

Table 7. Define homomorphic encryption (HE) requirements

Define homomorphic encryption (HE) requirements
SET he_run_req = pyhelayers.HeRunRequirements()
CALL he_run_req.set_he_context_options(["HEaaN_CKKS", "SEAL_CKKS"])
CALL he_run_req.set_integer_part_precision(7) # Max value range = 2^7 = 128
CALL he_run_req.set_fractional_part_precision(30) # Precision of 2^-30
CALL he_run_req.optimize_for_batch_size(batch_size)

To enable encrypted inference, a homomorphic encryption (HE) context must be defined. The pyhelayers.HeRunRequirements() object is used to configure encryption settings:

- HE Context Options: The script specifies two possible HE schemes (HEaaN_CKKS and SEAL_CKKS), ensuring compatibility with available cryptographic backends.
- Integer Precision: The system is set to process numbers with a maximum value of 2^7 = 128, ensuring encrypted computations remain within safe numerical limits.
- Fractional Precision: Floating-point values are stored with an accuracy of 2^-30, which provides high precision for medical data processing.

- Batch Optimization: The script fine-tunes encryption settings for the specified batch size, ensuring efficient encrypted processing.

Table 8. Compile the HE model and get an optimal batch size

Compile the HE model and get an optimal batch size
SET profile = pyhelayers.HeModel.compile(neural_net_plain, he_run_req) SET batch_size = profile.get_optimal_batch_size() PRINT profile.to_string() PRINT "He profile ready" PRINT "Batch size: ", batch_size

Once encryption parameters are set, the script compiles the neural network into an encrypted format using pyhelayers.HeModel.compile(). This process transforms the model into an **HE-compatible structure,** allowing it to perform encrypted computations without exposing raw data. Additionally, the **optimal batch size** for encrypted processing is determined to balance efficiency and security.

Table 9. Encode and encrypt the neural network model

Encode and encrypt the neural network model
Create an encryption context SET client_context = pyhelayers.HeModel.create_context(profile) PRINT 'Crypto-library ready', client_context.print_signature() # Encode and encrypt the neural network model SET client_nn = pyhelayers.NeuralNet(client_context) CALL client_nn.encode_encrypt(neural_net_plain, profile) PRINT 'client_nn initialized'

Before running encrypted predictions, a cryptographic context is created using pyhelayers.HeModel.create_context(profile). This context acts as a secure execution environment, ensuring that all computations are performed within an encrypted domain. The system also prints a cryptographic signature to verify the integrity of the encryption setup. The neural network is encrypted using client_nn.encode_encrypt(). This step transforms the previously loaded plain neural network into an encrypted neural network**,** enabling secure inference. Once encryption is complete, the model is ready to process patient data without ever exposing sensitive information.

3.3. Perform prediction on encrypted data

Table 10. Encrypt the data samples

Encrypt the data samples
Load the Homomorphic Encryption (HE) context SET server_context = pyhelayers.load_he_context(context_buffer) # Load the encrypted neural network model using the HE context SET server_nn = pyhelayers.load_he_model(server_context, nn_buffer) # Load the encrypted test samples SET server_samples = pyhelayers.load_encrypted_data(server_context, samples_buffer)

Once the client-side encryption is completed, the server must load the encrypted context, model, and input data before performing encrypted inference. This ensures that the computations remain secure and private. The process starts with loading the HE context (server_context), which includes encryption settings and keys, allowing encrypted operations without decryption. Next, the encrypted neural network model (server_nn) is loaded using pyhelayers.load_he_model(). This ensures the server can perform inference without accessing the original model, maintaining security. Finally, encrypted test samples (server_samples) are loaded

via pyhelayers.load_encrypted_data(). These inputs remain encrypted throughout processing, enabling secure predictions without exposing raw data.

Table 11. Perform inference in cloud/server using encrypted data and encrypted NN

Perform inference in cloud/server using encrypted data and encrypted NN
Initialize an encrypted data container for storing predictions SET server_predictions = pyhelayers.EncryptedData(server_context) # Measure execution time while running inference START TIMER with label 'predict' and batch_size # Perform encrypted inference on the encrypted test samples CALL server_nn.predict(server_predictions, server_samples) # Stop the timer and log execution time STOP TIMER #Save encrypted predictions to a buffer for later use SET predictions_buffer = server_predictions.save_to_buffer()

After loading the encrypted model and test samples, the next step is to perform inference in the encrypted domain**.** This means the server will process the encrypted inputs using the encrypted model and generate encrypted predictions—without ever decrypting the data.

The process begins with initializing an encrypted data container (server_predictions). This container, created using pyhelayers.EncryptedData(server_context), securely holds the encrypted predictions generated by the neural network. Keeping these predictions encrypted ensures data privacy throughout the computation. To assess performance, execution time is measured using the elapsed_timer('predict', batch_size) utility. Since homomorphic encryption (HE) can be computationally intensive, tracking inference time is crucial for identifying potential optimizations and improving efficiency.

Next, encrypted inference is performed using server_nn.predict(). The encrypted test samples (server_samples) are fed into the model, generating encrypted predictions (server_predictions). The server never decrypts any data during this process, maintaining end-to-end encryption and ensuring data security. Once inference is complete, the timer is stopped to record execution time. This benchmarking step helps evaluate the computational cost of HE-based operations and guides future improvements.

Finally, encrypted predictions are saved in predictions_buffer using server_predictions.save_to_buffer(). These encrypted results are then sent back to the client, where they can be decrypted and interpreted securely.

3.4. Decrypt predictions and assess the result

Table 12. Decrypting and Evaluating Predictions

Decrypting and Evaluating Predictions
Load the encrypted predictions buffer into an encrypted data object SET client_predictions = pyhelayers.load_encrypted_data(client_context, predictions_buffer) # Decrypt the predictions to get readable results SET decrypted_predictions = client_predictions.decrypt() # Evaluate model performance by comparing decrypted predictions with ground truth labels SET accuracy = compute_accuracy(decrypted_predictions, labels) # Print the accuracy of the encrypted model PRINT “Model Accuracy:”, accuracy # Generate a detailed classification report SET report = generate_classification_report(decrypted_predictions, labels)

```
# Print the classification report
PRINT report
```

Now that the encrypted neural network has completed its inference on the test data, the next step is to decrypt the predictions and evaluate the model's performance. Since the entire computation was performed in an encrypted domain, the output is still encrypted, meaning it needs to be sent back to the client for decryption and analysis. This ensures that sensitive medical data remains private throughout the process.

The process begins with loading encrypted predictions into the client context. The encrypted predictions, stored in predictions_buffer, are loaded using pyhelayers.load_encrypted_data(client_context, predictions_buffer). This step ensures that decryption aligns with the original encryption settings, preventing mismatches or errors. Next, the client proceeds with decrypting the predictions. Since only the client holds the decryption key, they use client_predictions.decrypt() to restore the predictions into a human-readable format. This allows for direct evaluation against the actual labels. Once decrypted, the model's accuracy is evaluated by comparing the predictions against the ground truth labels (labels). The accuracy score is calculated using accuracy_score(labels, decrypted_predictions > 0.5), applying a threshold of 0.5. Predictions above this threshold are classified as diabetic (1), while those below are non-diabetic (0). This provides an overall measure of the encrypted model's performance. To assess the impact of encryption, the model's accuracy is printed using print("Model Accuracy:", accuracy). Ideally, this value should be close to that of the original unencrypted model. A significant drop could indicate precision loss or computational errors introduced by homomorphic encryption.

While accuracy is useful, it does not capture the full picture—especially in imbalanced datasets. Thus, a classification report is generated using classification_report(labels, decrypted_predictions > 0.5). This report provides key metrics:

- Precision – How many predicted diabetics were actually diabetic?
- Recall – How many real diabetics were correctly identified?
- F1-score – A balance between precision and recall, useful for medical evaluations.

Finally, the classification report is printed using print(report). This helps medical practitioners and data scientists assess model performance across different categories. For example, a low recall might indicate missed true positives, while low precision could mean misdiagnosed healthy individuals.

3.5. Precision

Precision measures how many of the predicted positives were actually correct:

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}}$$

- For Class 0 (Non-Diabetic), precision = 0.92, meaning 92% of non-diabetic predictions were correct**.**
- For Class 1 (Diabetic), precision = 0.00, meaning the model never predicted diabetes, so there were no correct diabetic classifications**.**

The prediction accuracy results using fully homomorphic encryption (FHE) are similar to those obtained without FHE. Both approaches yield classification report indicates a high precision for class 0 (0.92) and a very low recall for class 1 (0.00), resulting in an overall accuracy of 0.92. Despite the encryption method, the model's performance remains consistent, with the macro average F1-score at 0.48 and a weighted average F1-score of 0.88. The confusion matrix reflects this, with the model correctly predicting most of class 0 but struggling to classify instances of class 1.

$$\begin{bmatrix} 3760 & 0 \\ 346 & 0 \end{bmatrix}$$

The matrix shows that the model accurately classifies all class 0 instances but fails to classify any class 1 instances, indicating that the model's performance is heavily biased towards class 0. These results are consistent regardless of the use of FHE.

Table 13. Model classification matrix

Metric	Class 0 (Non-Diabetic)	Class 1 (Diabetic)
Precision	0.92	0.00
Recall	1.00	0.00
F1-Score	0.96	0.00
Support	3750	346

When evaluating the performance of neural network models on held-out test samples, it was observed that models trained with encryption achieved an almost identical accuracy of 92%. This finding suggests that the overall performance of neural networks remains consistent, regardless of whether encryption was applied during the training phase.

3.6. Recall

Recall measures how many actual positives were correctly identified:

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}}$$

- For Class 0, recall = 1.00, meaning all non-diabetics were correctly identified.
- For Class 1, recall = 0.00, meaning none of the actual diabetics were correctly classified. The model completely failed to detect diabetes.

3.7. F1-Score

F1-score is the harmonic mean of precision and recall, providing a balanced measure:

$$\text{F1} = 2 \text{ x} \frac{\text{Precision x Recall}}{\text{Precision} + \text{Recall}}$$

The AUC (Area Under the Curve) score is 0.925, which is usually considered excellent. The model is perfect at classifying non-diabetics**.** These findings are highly significant, as they demonstrate that privacy-preserving techniques, such as homomorphic encryption, do not compromise the accuracy or reliability of neural network models. This result reinforces the feasibility and practicality of employing privacy-preserving approaches to protect sensitive data, especially in fields where confidentiality and security are paramount.

By highlighting the comparable performance between encrypted and unencrypted models, this study contributes to the growing body of research advocating for the adoption of privacy-preserving machine learning techniques. It underscores the potential of homomorphic encryption and similar methods to facilitate secure data processing while ensuring the integrity and effectiveness of neural network models.

4. CONCLUSION

Integrating Fully Homomorphic Encryption (FHE) into neural network-driven multiclass classification presents a transformative opportunity for secure and privacy-focused machine learning. This study highlights the viability of training and testing neural network models on encrypted datasets by utilizing FHE-based operations for encrypted feature processing, model development, and inference.

In this research, we implemented a Convolutional Neural Network (CNN) architecture with multiple convolutional and fully connected layers optimized using the Adam optimizer. The model was trained with a batch size of 32 over three epochs with a learning rate of 0.01. Our experimental findings confirm that achieving competitive accuracy in multiclass classification is feasible while preserving the security of the underlying data. The proposed model achieved an accuracy of 92%, demonstrating that privacy-preserving neural network training using FHE does not significantly compromise performance.

Employing Fully Homomorphic Encryption in CNN-based classification marks a substantial step forward in privacy-preserving data analysis, particularly for domains where confidentiality is paramount. This

approach enables secure collaboration, facilitates confidential data sharing, and supports outsourced machine learning without exposing private information. The minimal disparity in performance between encrypted and traditional neural network models further reinforces the efficiency of this method.

These findings pave the way for continued exploration of homomorphic encryption and similar technologies in data-centric applications, unlocking new possibilities for privacy-conscious machine learning. As this technology evolves, it has the potential to redefine how sensitive data is processed, analyzed, and safeguarded, fostering trust in an increasingly data-driven world..

ACKNOWLEDGEMENTS

The author greatly appreciates the continuous support and encouragement from family and friends during the writing of this journal. Their help and motivation were invaluable in ensuring the completion of this journal.

REFERENCES

[1] M. Tariq, Y. Hayat, A. Hussain, A. Tariq, and S. Rasool, "Principles and perspectives in medical diagnostic systems employing artificial intelligence (AI) algorithms," *International Research Journal of Economics and Management Studies IRJEMS*, vol. 3, no. 1, 2024.

[2] L. Pinto-Coelho, "How Artificial Intelligence Is Shaping Medical Imaging Technology: A Survey of Innovations and Applications," Mar. 2023, *Multidisciplinary Digital Publishing Institute (MDPI)*. doi: 10.3390/bioengineering10121435.

[3] N. Khalid, A. Qayyum, M. Bilal, A. Al-Fuqaha, and J. Qadir, "Privacy-preserving artificial intelligence in healthcare: Techniques and applications," *Comput Biol Med*, vol. 158, p. 106848, May 2023, doi: 10.1016/j.compbiomed.2023.106848.

[4] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy," in *International conference on machine learning*, 2016, pp. 201–210.

[5] A. Wood, K. Najarian, and D. Kahrobaei, "Homomorphic Encryption for Machine Learning in Medicine and Bioinformatics," *ACM Comput Surv*, vol. 53, no. 4, Mar. 2020, doi: 10.1145/3394658.

[6] M. Az, S. F. Pane, and R. M. Awangga, "Cryptography: Perancangan Middleware Web Service Encryptor menggunakan Triple Key MD5. Base64, dan AES," *Jurnal Tekno Insentif*, vol. 15, no. 2, pp. 65–75, 2021.

[7] L. Zhang, J. Xu, P. Vijayakumar, P. K. Sharma, and U. Ghosh, "Homomorphic Encryption-Based Privacy-Preserving Federated Learning in IoT-Enabled Healthcare System," *IEEE Trans Netw Sci Eng*, vol. 10, no. 5, pp. 2864–2880, Mar. 2023, doi: 10.1109/TNSE.2022.3185327.

[8] S. Abut, H. Okut, and K. J. Kallail, "Paradigm shift from Artificial Neural Networks (ANNs) to deep Convolutional Neural Networks (DCNNs) in the field of medical image processing," Mar. 2024, *Elsevier Ltd*. doi: 10.1016/j.eswa.2023.122983.

[9] L. Alzubaidi *et al.*, "Review of deep learning: concepts, CNN architectures, challenges, applications, future directions," *J Big Data*, vol. 8, no. 1, Mar. 2021, doi: 10.1186/s40537-021-00444-8.

[10] L. P. Fávero, P. Belfiore, and R. de Freitas Souza, "Artificial neural networks," in *Data Science, Analytics and Machine Learning with R*, Elsevier, 2023, pp. 441–467. doi: 10.1016/B978-0-12-824271-1.00023-8.

[11] T. Ishiyama, T. Suzuki, and H. Yamana, "Highly Accurate CNN Inference Using Approximate Activation Functions over Homomorphic Encryption," in *2020 IEEE International Conference on Big Data (Big Data)*, IEEE, Dec. 2020, pp. 3989–3995. doi: 10.1109/BigData50022.2020.9378372.

[12] Z. Zhang, "Improved Adam Optimizer for Deep Neural Networks," in *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*, 2018, pp. 1–2. doi: 10.1109/IWQoS.2018.8624183.

[13] E. Aharoni, N. Drucker, and H. Shaul, "Demo: Rotating Wide Tensors with HElayers," in *Proceedings of the 2023 Tutorial on Advanced HE Packing Methods with Applications to ML*, New York, NY, USA: ACM, Nov. 2023, pp. 3–5. doi: 10.1145/3605774.3625524.

[14] R. Deviani, "The application of fully homomorphic encryption on XGBoost based multiclass classification," *JIEET (Journal of Information Engineering and Educational Technology)*, vol. 7, no. 1, pp. 49–58, 2023.

[15] P. Choksi, "Comprehensive Diabetes Clinical Dataset(100k rows)," 2024.