Hana Hasan, Hallel Weinberg, Tidhar Rettig

Privacy-Preserving Distributed Expectation Maximization for Gaussian Mixture Models

Prof. Adi Akavia's Secure Cloud Computing Laboratory, Fall 2022-2023

Introduction

Background

Proposed Approach

Conclusion

Introduction

Background

Proposed Approach

Conclusion

Introduction

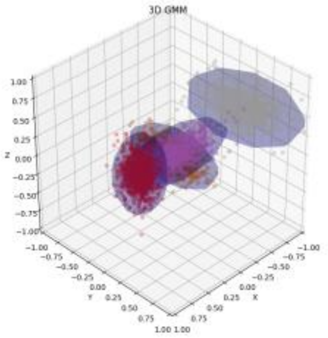
In recent years, cloud computing has emerged as a ubiquitous and cost-effective solution for storing and processing large volumes of data. However, the outsourcing of data to remote servers in the cloud raises concerns about data privacy and security. As data breaches and privacy violations become increasingly common, there is a growing need for secure cloud computing solutions that can ensure the confidentiality, integrity, and availability of data.

Introduction

One of the major challenges in secure cloud computing is the need to preserve the privacy of sensitive data while allowing for meaningful analysis. In this paper, we propose a privacy-preserving distributed expectation-maximization algorithm for Gaussian mixture models. Our method involves utilizing fully homomorphic encryption to facilitate a privacy-preserving centralized federated learning approach.

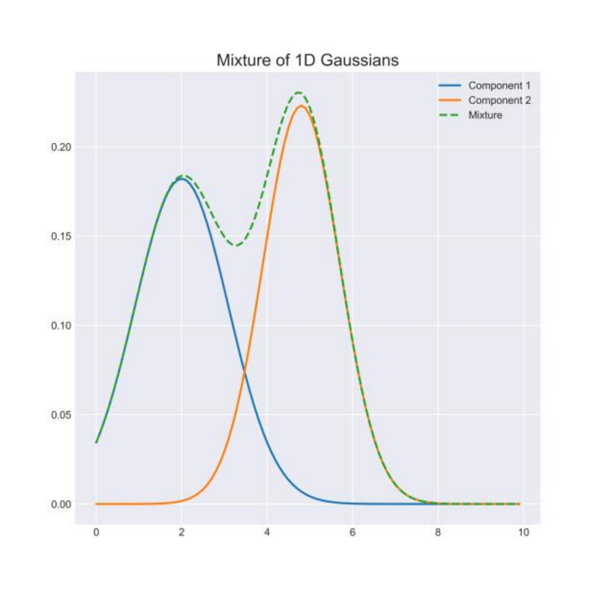

Introduction
Background
Proposed Approach
Conclusion

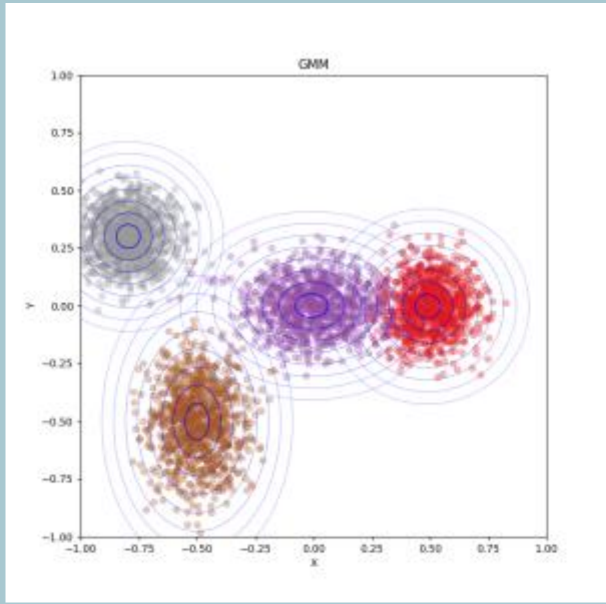
Gaussian Mixture Models (GMM)

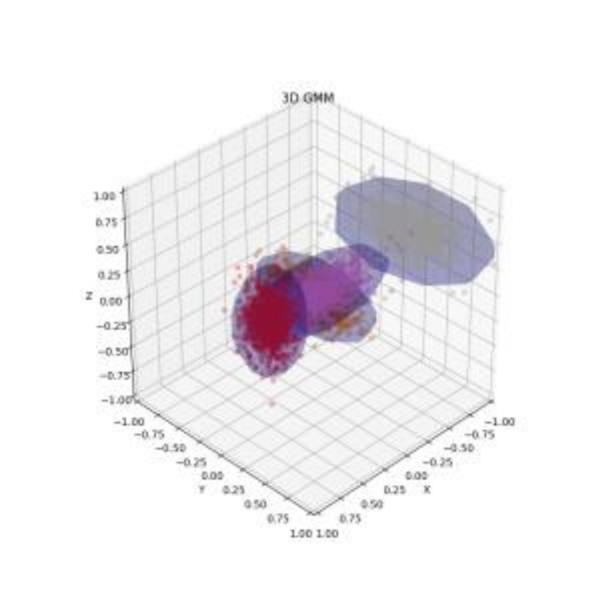


Gaussian Mixture Models (GMM)

A Gaussian mixture model is a probabilistic model that assumes all the data points are generated from a mixture of a finite number of Gaussian distributions with unknown parameters.





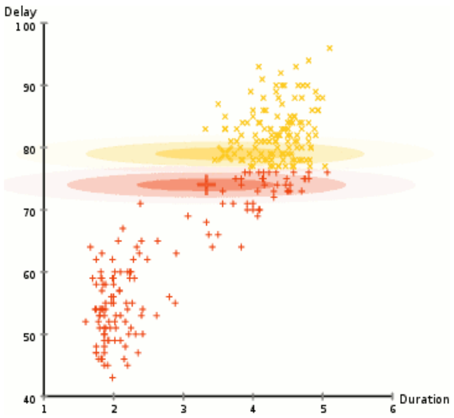


Gaussian Mixture Models (GMM)

Gaussian mixture models are very useful clustering models. Note that in traditional clustering algorithms such as k-means or DBSCAN, each data point belongs to exactly one cluster (hard clustering).

Gaussian mixture models, on the other hand, use soft clustering where each data point may belong to several clusters with a fractional degree of membership in each.In the GMM framework, each Gaussian component is characterized by its mean μ, covariance matrix σ, and mixture coefficient (weight) β.

Expectation Maximization (EM)



Expectation Maximization (EM)

Expectation maximization is a clustering-based machine learning algorithm that is widely used in many areas of science, such as bio-informatics and computer vision, to perform maximum likelihood estimation (MLE) estimation for models with latent (hidden, unobserved) variables.

EM is an iterative algorithm that starts with an initial guess of the model's parameters (All the parameters are collectively denoted as θ), and then proceeds to iteratively update θ_i until convergence:

E-Step

For the i^{th} step, we calculate how likely it is to observe the data, as a function of θ. The equation for this calculation is:

$$E[l(\theta; X, \Delta)|X, \theta_i]$$

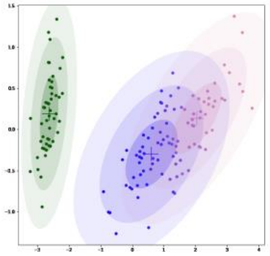
such that X is the data, Δ is the latent data, θ_i are the parameter-estimates of the previous iteration (or initial guess), and we define l as the log-likelihood function.

M-Step

We compute parameters maximizing the expected log-likelihood found on the expectation step and call the result θ_{i+1} , this is the new estimate which will be used in the next iteration of the algorithm. The equation for this calculation is:

$$\theta_{i+1} = argmax_0(E = [l(\theta; X, \Delta)|X, \theta_i])$$

Expectation Maximization for GMM



Expectation-Maximization for GMM

The expectation maximization algorithm can be applied to estimate the parameters of a G

In the case of a GMM, the E-step involves computing the posterior probabilities of each data point belonging to each of the Gaussian components. These probabilities are used to update the estimates of the mixture weights and the means and covariances of each Gaussian component in the M-step. Gaussian mixture model given a set of observed data.

Distributed Expectation Maximization for GMM

Distributed EM for GMM

Consider the scenario where n parties each has its own data x_i and these parties would like to collaborate to learn a GMM based on the full dataset $\{x_1, x_2, \ldots, x_n\}$.

Assuming there are c Gaussian components, the GMM density is given by:

$$p(x) = \Sigma_{j=1}^{c} \beta_j p\left(x \middle| \mu_i, \Sigma_j\right)$$

where β_j is the mixing coefficient of the j^{th} Gaussian component, and μ_j and Σ_j are the mean and covariance, respectively, of the j^{th} Gaussian component.

Distributed EM for GMM

The EM algorithm is iterative and for each iteration t, the following steps are taken for all Gaussian components:

E-Step

$$P\left(x_i \middle| N_j^t\right) = \frac{p\left(x_i \middle| \mu_j, \Sigma_j\right)\beta_j^t}{\Sigma_{k=1}^{c} p(x_i|\mu_k, \Sigma_k)\beta_k^t}$$

where $p(x_i|\mu_k, \Sigma_k)$ is the pdf for a Gaussian distribution with mean μ_k and covariance matrix Σ_k, and β_k is the mixing coefficient of the k^{th} Gaussian component.

M-Step

$$\beta_j^{t+1} = \frac{\Sigma_{i=1}^{n} P\left(x_i \middle| N_j^t\right)}{n}$$

$$\mu_j^{t+1} = \frac{\Sigma_{i=1}^{n} P\left(x_i \middle| N_j^t\right) x_i}{\Sigma_{i=1}^{n} P\left(x_i \middle| N_j^t\right)}$$

$$\Sigma_j^{t+1} = \frac{\Sigma_{i=1}^{n} P\left(x_i \middle| N_j^t\right)\left(x_i - \mu_j^t\right)\left(x_i - \mu_j^t\right)^{\mathsf{T}}}{\Sigma_{\{i=1\}}^{n} P\left(x_i \middle| N_j^t\right)}$$

where $P\left(x_i \middle| N_j^t\right)$ denotes the conditional probability that data x_i belongs to Gaussian model j (result of the E-Step).

Privacy-Preserving Expectation Maximization (PPEM)

Privacy-Preserving EM (PPEM)

To deploy such an algorithm in cloud environments, security and privacy issues need be considered to avoid data breaches or abuses by external malicious parties or even by cloud service providers.

Existing approaches for PPEM:

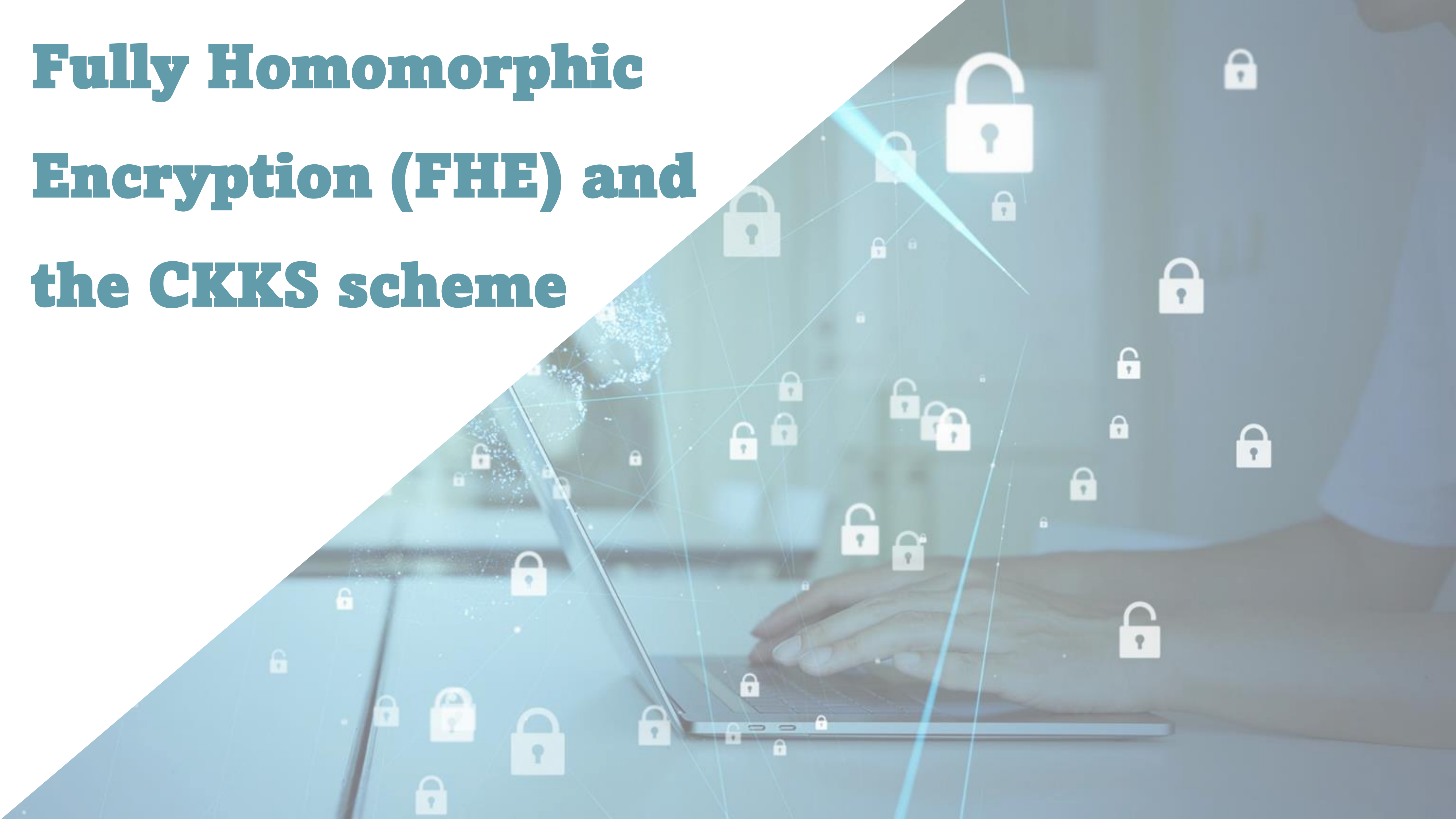
- **Differential Privacy (DP) based PPEM:** The DP-based PPEM approaches perturb the data to prevent sensitive information from being leaked. One such approach is the DP-EM algorithm that adds noise to the EM algorithm's update steps to ensure differential privacy.
- **Homomorphic Encryption (HE) based PPEM:** Homomorphic encryption allows computation on encrypted data without decryption, enabling privacy-preserving computation. The HE-based PPEM approaches encrypt the data before using the EM algorithm, ensuring that sensitive information is not leaked.

Existing approaches for PPEM:

- **Federated Learning based PPEM:** Federated Learning is a machine learning technique that enables training on decentralized data. The Federated EM algorithm is a variant of the EM algorithm that uses Federated Learning to train a model on multiple devices without centralizing the data.

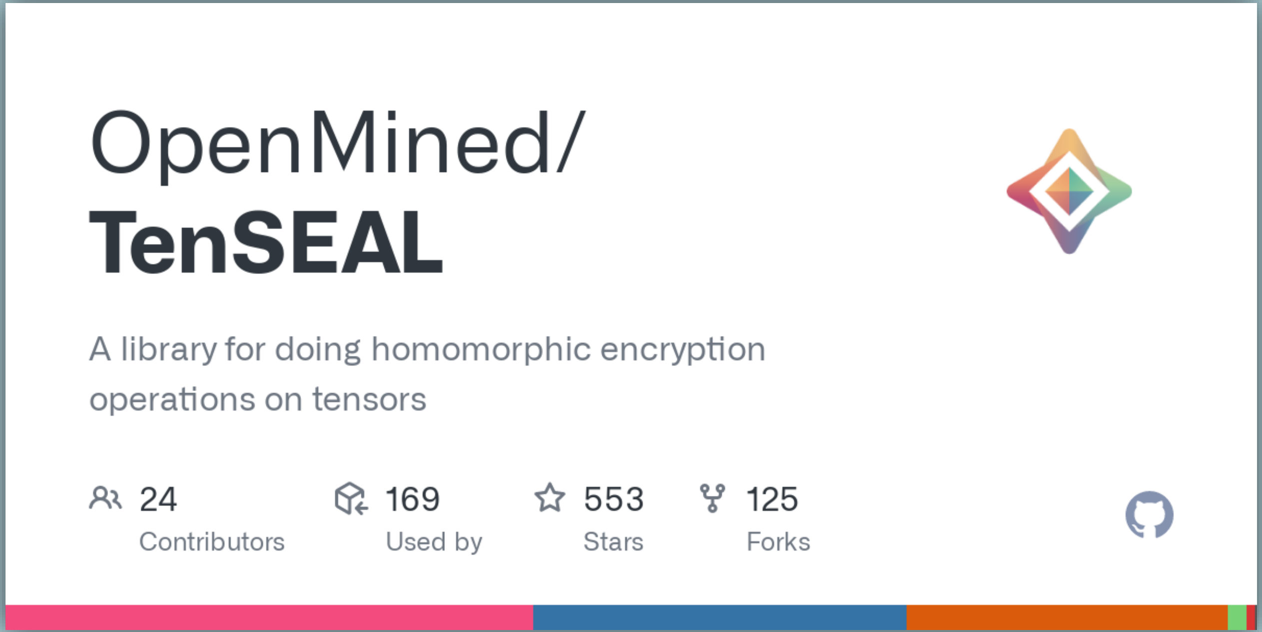
- **Secure Multi-Party Computation (SMPC) based PPEM:** Secure Multi-Party Computation (SMPC) is a cryptographic protocol that allows multiple parties to compute a function while keeping their inputs private. SMPC-based PPEM approaches enable multiple parties to run the EM algorithm on their local data without sharing it, ensuring privacy.

Fully Homomorphic Encryption (FHE) and the CKKS scheme



Fully Homomorphic Encryption (FHE) and the CKKS scheme

Acknowledgement: Our project uses the TenSEAL library, which is an open-source library for homomorphic encryption in Python.



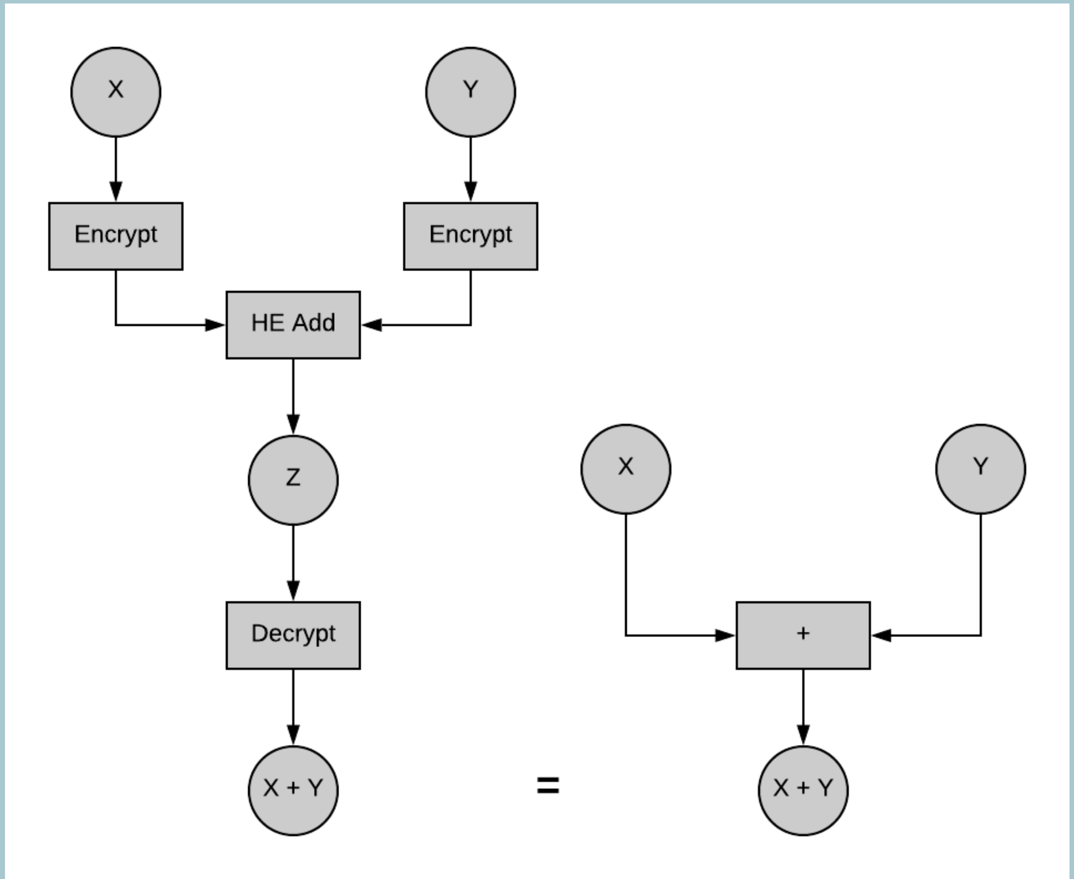
Fully Homomorphic Encryption (FHE)

Definition: Fully Homomorphic Encryption (FHE) is an encryption technique that allows computations to be made on ciphertexts and generates results that when decrypted, correspond to the results of the same computations made on plaintexts.

In practice, for an application that needs to perform some computation F on data that is encrypted, the FHE scheme would provide some alternative computation F' which when applied directly over the encrypted data will result in the encryption of the application of F over the data in the clear. More formally:

$$F(unencrypted\_data) = Decrypt(F'(encrypted\_data))$$

Fully Homomorphic Encryption (FHE)



Fully Homomorphic Encryption (FHE)

Formally, an FHE scheme is a tuple of four algorithms $(Gen, Enc, Dec, Eval)$ that satisfy the following properties:

***Gen*:**
The key generation algorithm takes a security parameter k as input and outputs a public key pk and a secret key sk.

***Enc*:**
The encryption algorithm takes a public key pk and a message m as input and outputs a ciphertext c.

***Dec*:**
The decryption algorithm takes a secret key sk and a ciphertext c as input and outputs the original message m.

***Eval*:**
The evaluation algorithm takes a function f and a set of ciphertexts $\{c_1, c_2, \dots, c_n\}$ as input and outputs a new ciphertext c_f that represents the result of applying the function f to the plaintexts corresponding to the input ciphertexts.

Fully Homomorphic Encryption (FHE)

For an FHE scheme to be fully homomorphic, the Eval algorithm must satisfy two additional properties:

Correctness:

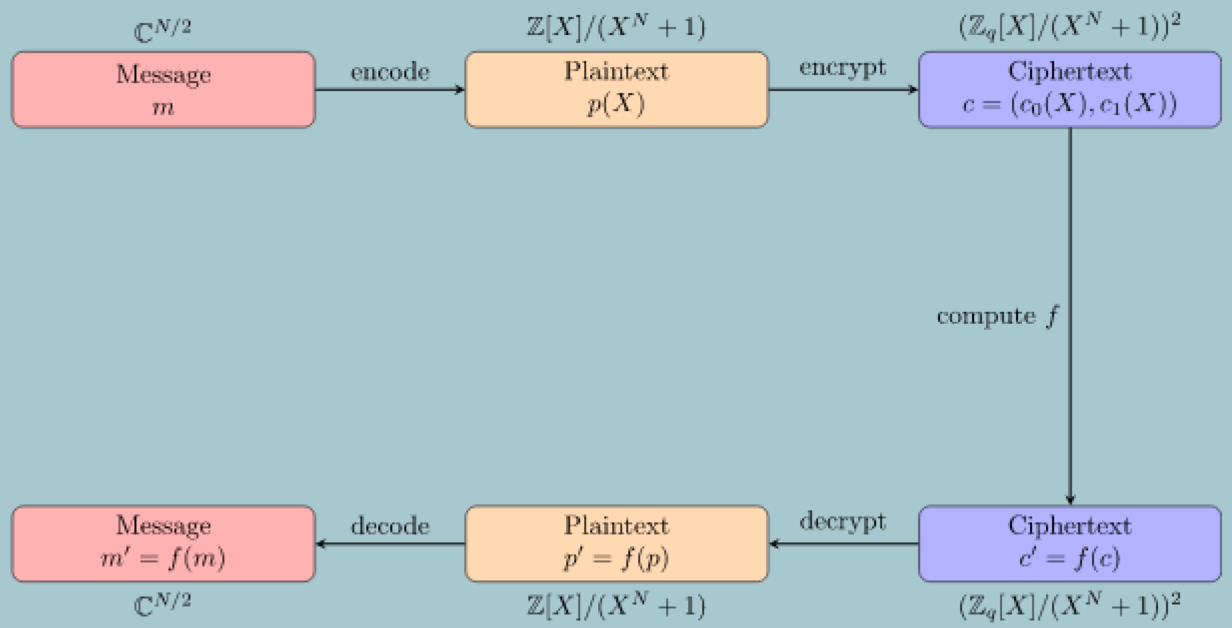
For any function f and any set of ciphertexts $\{c_1, c_2, \ldots, c_n\}$ corresponding to plaintexts $\{m_1, m_2, \ldots, m_n$, the ciphertext c_f output by $Eval(pk, f, \{c_1, c_2, \ldots, c_n\})$ must decrypt to the correct result $f(m_1, m_2, \ldots, m_n)$.

Security:

The scheme must provide a level of security that makes it infeasible for an attacker to learn any information about the plaintexts from the ciphertexts or the public key.

CKKS scheme

Definition: Cheon-Kim-Kim-Song(CKKS) is a scheme for Leveled Homomorphic Encryption that supports approximate arithmetics over complex numbers (hence, real numbers).



CKKS scheme

CKKS keys:

- **The secret key:** The secret key is used for decryption. DO NOT SHARE IT.
- **The public encryption key:** The key is used for encryption in the public key encryption setup.
- **The relinearization keys:** Every new ciphertext has a size of 2, and multiplying ciphertexts of sizes K and L results in a ciphertext of size $K + L - 1$. Unfortunately, this growth in size slows down further multiplications and increases noise growth.
- **The Galois Keys(optional):** Galois keys are another type of public keys needed to perform encrypted vector rotation operations on batched ciphertexts. One use case for vector rotations is summing the batched vector that is encrypted.

CKKS scheme

Note:

Relinearization is the operation that reduces the size of ciphertexts back to 2. This operation requires another type of public keys, the relinearization keys created by the secret key owner.

The operation is needed for encrypted multiplications. The plain multiplication is fundamentally different from normal multiplication and does not result in ciphertext size growth.

Adversary Models

Adversary models are used to describe the capabilities and goals of attackers who may attempt to compromise the security of the system, such as:

- **Semi-Honest ("honest, but curious"):** All parties follow protocol instructions, but dishonest parties may be curious to violate privacy of others when possible.
- **Fully Malicious Model:** Adversarial Parties may deviate from the protocol arbitrarily (quit unexpectedly, send different messages etc). It is much harder to achieve security in the fully malicious model.


Introduction
Background
Proposed Approach
Conclusion

Motivation

Motivation

All proposed approaches in prior work on PPEM involve a trade-off between accuracy, privacy, and performance.

For example, Fully Homomorphic Encryption (FHE) based algorithms can be time consuming and computationally heavy as computing over encrypted data incurs a high computational overhead, however privacy will be maintained as FHE allows for secure computations on encrypted data without requiring access to the plaintext.

Motivation

Another example is Secure Multi-Party Computation (SMPC) based PPEM: maintaining privacy is a primary goal of SMPC. However, achieving perfect privacy comes at the cost of computational complexity and performance, and the efficiency of SMPC can be impacted by the number of parties involved in the computation.

Our Contribution

Our Contribution

We propose a protocol for privacy-preserving expectation maximization that attains the desirable properties of FHE while simplifying calculations to only require addition operations on encrypted data. This approach allows us to achieve a high level of privacy while also improving performance and reducing computational overhead.

By utilizing the strengths of FHE while streamlining the computation process, we are able to strike a balance between accuracy, privacy, and performance in our proposed approach.

Settings

Settings

We address the following scenario:

- Data is distributed and private: there are n parties, each holding its own private data x_i (a two-dimensional point) and we wish to fit a GMM to the full dataset, without revealing the private data of each party.
- We propose a client-server model, where the cloud service is an untrusted third party and acts as the central server, providing a service to the clients who are the owners of the private data.
- We assume an Honest-but-Curious adversary and do not consider the Malicious case, thus we can assume that the server (the untrusted third party, the "cloud") is not mailcious.
- To generate, distribute, and manage cryptographic keys for the CKKS scheme via TenSEAL library, we utilize a Key Management Service (KMS).

Reminder

Reminder: Distributed EM for GMM

Consider the scenario where n parties each has its own data x_i and these parties would like to collaborate to learn a GMM based on the full dataset $\{x_1, x_2, \ldots, x_n\}$.

Assuming there are c Gaussian components, the GMM density is given by:

$$p(x) = \Sigma_{j=1}^{c} \beta_j p\left(x \middle| \mu_i, \Sigma_j\right)$$

where β_j is the mixing coefficient of the j^{th} Gaussian component, and μ_j and Σ_j are the mean and covariance, respectively, of the j^{th} Gaussian component.

Reminder: Distributed EM for GMM

The EM algorithm is iterative and for each iteration t, the following steps are taken for all Gaussian components:

E-Step

$$P\left(x_i \middle| N_j^t\right) = \frac{p\left(x_i \middle| \mu_j, \Sigma_j\right)\beta_j^t}{\Sigma_{k=1}^{c} p(x_i | \mu_k, \Sigma_k)\beta_k^t}$$

where $p(x_i|\mu_k, \Sigma_k)$ is the pdf for a Gaussian distribution with mean μ_k and covariance matrix Σ_k, and β_k is the mixing coefficient of the k^{th} Gaussian component.

M-Step

$$\beta_j^{t+1} = \frac{\sum_{i=1}^{n} P\left(x_i \middle| N_j^t\right)}{n}$$

$$\mu_j^{t+1} = \frac{\sum_{i=1}^{n} P\left(x_i \middle| N_j^t\right) x_i}{\sum_{i=1}^{n} P\left(x_i \middle| N_j^t\right)}$$

$$\Sigma_j^{t+1} = \frac{\sum_{i=1}^{n} P\left(x_i \middle| N_j^t\right)\left(x_i - \mu_j^t\right)\left(x_i - \mu_j^t\right)^{\mathsf{T}}}{\sum_{\{i=1\}}^{n} P\left(x_i \middle| N_j^t\right)}$$

where $P\left(x_i \middle| N_j^t\right)$ denotes the conditional probability that data x_i belongs to Gaussian model j (result of the E-Step).

Our Approach

Our Approach

It's worth noting that in the distributed version of the Expectation Maximization algorithm, the $E - Step$ can be computed locally. This means that each party can perform the E-step on its own data, thus preserving data privacy.

However, the $M - Step$ requires data aggregation to compute global updates of the Gaussian components' parameters, which raises privacy concerns.

To address this, we simplify the M-step by breaking it down into intermediate updates that are also computed locally and will periodically be communicated with the central server.

The Algorithm

For each iteration t, the KMS generates a pair of public and secret keys

(pk_t, sk_t)

and distributes both keys to all parties involved in the computation. Therefore, each party will hold both the public key pk_t and the corresponding secret key sk_t for the current iteration. Furthermore, the server gets access only to the public key pk_t.

The Algorithm

For every Gaussian component j, each node i computes the following intermediate updates:

$$a_{ij}^{t} = P\left(x_i \middle| N_j^t\right)$$

$$b_{ij}^{t} = P\left(x_i \middle| N_j^t\right) x_i$$

$$c_{ij}^{t} = P\left(x_i \middle| N_j^t\right)\left(x_i - \mu_j^t\right)\left(x_i - \mu_j^t\right)^{\top}$$

The Algorithm

All the above updates can also be computed locally at node i, and then the node can compute its intermediate updates vector:

$$v_{ij}^t = \left[a_{ij}^t, b_{ij_0}^t, b_{ij_1}^t, c_{ij_{00}}^t, c_{ij_{01}}^t, c_{ij_{10}}^t, c_{ij_{11}}^t\right]$$

x_i is a two-dimensional point, a_{ij} is a scalar, b_{ij} is a two-dimensional vector, and c_{ij} is a $2x2$ matrix. Thus, v_{ij}^t includes all the entries of a, b, c.

The Algorithm

Node i then encrypts this vector and sends the ciphertext $v_{ij}^{\hat{t}} \leftarrow Enc_{pk_t}(v_{ij}^{t})$ to the server.

After receiving these intermediate updates from all nodes, the server computes the sum of all the vectors (for a specific Gaussian component j):

$$v_{ij}^{\hat{t}} \leftarrow Eval_{pk_t}\left(C, v_{i_1 j}^{\hat{t}}, \ldots, v_{i_n j}^{\hat{t}}\right)$$

where C is a homomorphic addition circuit.

The Algorithm

The server sends back the result $s_{ij}^{\hat{t}}$ to all the nodes, and then every node \$i\$ decrypts it to obtain the sum $v_j^t \leftarrow Dec_{sk_t}\left(v_j^{\hat{t}}\right)$ which is $v_j^t = \Sigma_{i=1}^n v_{ij}^t$ from which we can obtain the sums $\Sigma_{i=1}^n a_{ij}^t, \Sigma_{i=1}^n b_{ij}^t, \Sigma_{i=1}^n c_{ij}^t$.

The Algorithm

These sums are used to update the global estimates of the mixture weights and the means and covariances of each Gaussian component (global updates):

$$\beta_j^{t+1} = \frac{\sum_{i=1}^{n} P\left(x_i \middle| N_j^t\right)}{n} = \frac{\sum_{i=1}^{n} a_{ij}^t}{n}$$

$$\mu_j^{t+1} = \frac{\sum_{i=1}^{n} P\left(x_i \middle| N_j^t\right) x_i}{\sum_{i=1}^{n} P\left(x_i \middle| N_j^t\right)} = \frac{\sum_{i=1}^{n} b_{ij}^t}{\sum_{i=1}^{n} a_{ij}^t}$$

$$\Sigma_j^{t+1} = \frac{\sum_{i=1}^{n} P\left(x_i \middle| N_j^t\right)\left(x_i - \mu_j^t\right)\left(x_i - \mu_j^t\right)^{\top}}{\sum_{i=1}^{n} P\left(x_i \middle| N_j^t\right)} = \frac{\sum_{i=1}^{n} c_{ij}^t}{\sum_{i=1}^{n} a_{ij}^t}$$

Privacy

Privacy

Let us quantify the individual privacy of each node's private data using mutual information $I(\bar{X};\bar{Y})$ which measures the mutual dependence between two random variables $\bar{X},\bar{Y}$. We have that $0 \leq I(\bar{X};\bar{Y})$, with equality if and only if $\bar{X}$ is independent of $\bar{Y}$. On the other hand, if $\bar{Y}$ uniquely determines $\bar{X}$ we have $I(\bar{X};\bar{Y}) = I(\bar{X};\bar{X})$which is maximal.

Privacy

Suppose each node shared its intermediate updates vector without encrypting it first, then although the node does not share the private date directly, it is still revealed to the server. This is because with the intermediate updates a_{ij}^t and b_{ij}^t, the server is able to determine the private data x_i of each node i since $b_{ij}^t = a_{ij}^t x_i$.

That is, at each iteration the server has the following mutual information

$$I\left(\overline{X_i}; \overline{A_{ij}^t}, \overline{B_{ij}^t}\right) = I\left(\overline{X_i}; \overline{X_i}\right)$$

which is maximal. This means that every node's private data x_i is completely revealed to the server. Hence, the algorithm would not be privacy-preserving at all.

Privacy

In our approach, the server is unable to determine the private data x_i of each node as it only receives the encrypted intermediate updates a_{ij}^t and b_{ij}^t.

Moreover, at the end of each iteration, each node solely acquires the global updates, which do not expose any information regarding the private data of other nodes. Specifically, from the global updates, nodes can acquire $\Sigma_{i=1}^n a_{ij}^t, \Sigma_{i=1}^n b_{ij}^t, \Sigma_{i=1}^n c_{ij}^t$, from which they can compute the sum of the intermediate updates of other nodes. Given the large number of participants, this does not raise any privacy concerns.

Correctness

Correctness

The fitted model, i.e., the estimated parameters of GMM, should be the same as using non-privacy preserving counterparts. Namely, the performance of the GMM should not be compromised by considering privacy.

Since we're using the CKKS scheme, the results obtained through this scheme may be approximate, but this does not significantly compromise the accuracy of the model.

Results

Results

In this section, we demonstrate numerical results to validate the comparison between the non-privacy preserving algorithm and our privacy preserving approach.

To achieve this, we utilize a data generation function that takes three input parameters: num_of_gaussians, which determines the number of Gaussian components to simulate, points_per_gaussian, which determines the number of data points in each Gaussian component, and mean_range, which specifies the range of mean values for the data points.

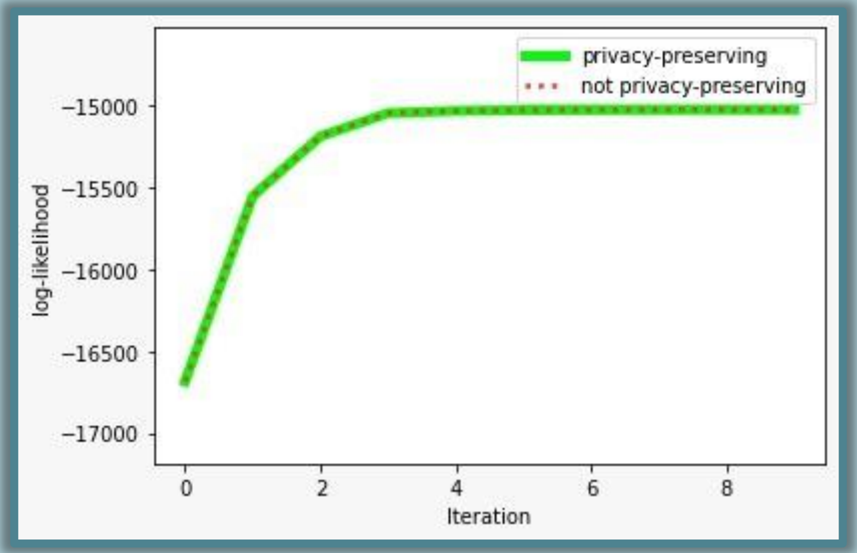
We simulate several GMMs using the data generation function and compare the two algorithms using the log-likelihood of the GMMs. We plot the log-likelihood as a function of the iteration number of our proposed privacy preserving approach and the existing non-privacy preserving approach.

The log-likelihood of a GMM with data points and Gaussian components is defined by:

$$\mathcal{L} = \sum_{i=1}^{n} log\left(\sum_{j=1}^{c} \beta_j \cdot \mathcal{N}\left(x_i \middle| \mu_j, \Sigma_j\right)\right)$$

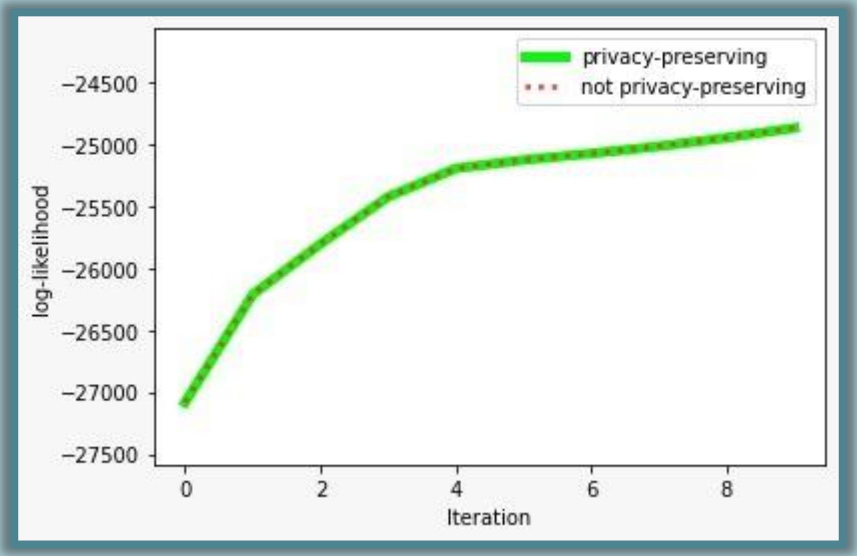
Results

Experiment 1: $num\_of\_gaussians = 3, points\_per\_gaussian = 1000, mean\_range = [-20, 20]$



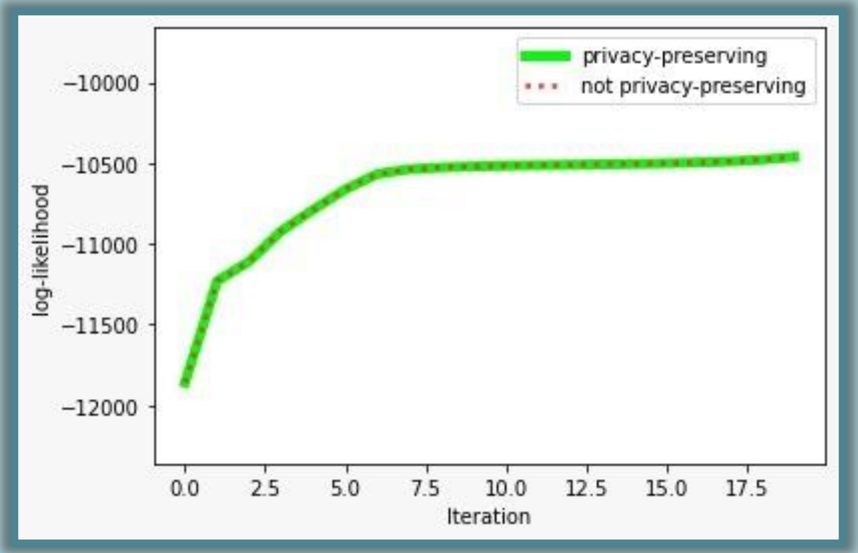
Results

Experiment 2: $num\_of\_gaussians = 5, points\_per\_gaussian = 900, mean\_range = [-10, 10]$



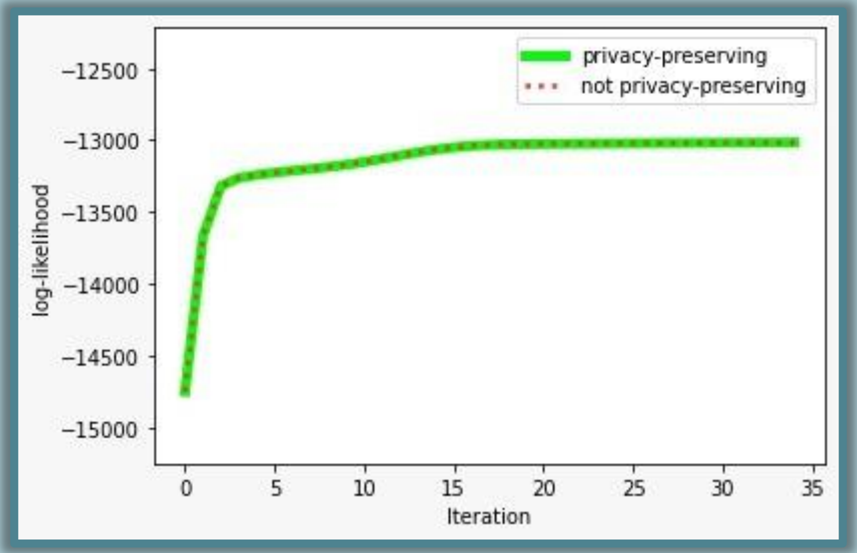
Results

Experiment 3: $num\_of\_gaussians = 4, points\_per\_gaussian = 500, mean\_range = [-10, 10]$



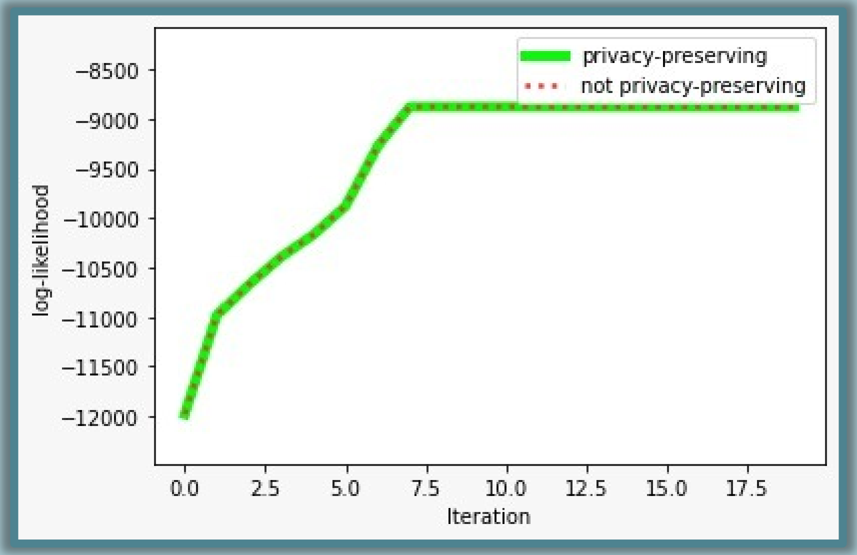
Results

Experiment 4: $num\_of\_gaussians = 5, points\_per\_gaussian = 500, mean\_range = [-10, 10]$



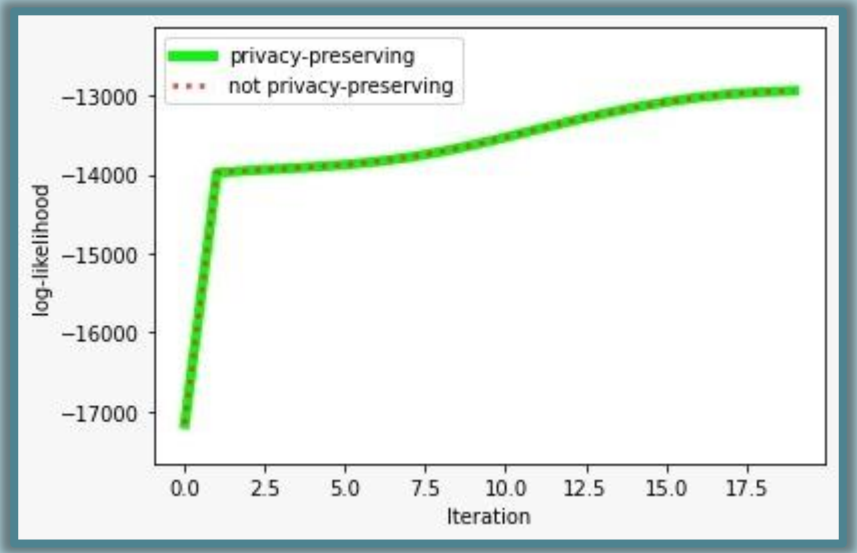
Results

Experiment 5: $num\_of\_gaussians = 2, points\_per\_gaussian = 1000, mean\_range = [-10, 10]$



Results

Experiment 6: $num\_of\_gaussians = 3, points\_per\_gaussian = 1000, mean\_range = [-10, 10]$



Introduction → Background → Proposed Approach → **Conclusion**

Conclusion and Further Questions

As shown in the figures in the previous seciton, we see that the proposed approach yields parameter estimations for GMMs that are indistinguishable from those of the non-privacy preserving approach. Hence, the output correctness of the proposed approach is guaranteed and not compromised by considering privacy.

By using fully homomorphic encryption, we were able to maintain participants' individual privacy because the private data did not need to be disclosed for computations.

Regarding performance, applying parallel computing could significantly enhance it. Instead of having the server wait for all participants to send intermediate updates, the process can be done simultaneously, resulting in a faster runtime.