IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 27 June 2024, accepted 28 July 2024, date of publication 5 August 2024, date of current version 20 August 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3438161

RESEARCH ARTICLE

HEaaN-NB: Non-Interactive Privacy-Preserving Naive Bayes Using CKKS for Secure Outsourced Cloud Computing

BOYOUNG HAN[1,*], HOJUNE SHIN[1,*], YEONGHYEON KIM[2], JINA CHOI[1], AND YOUNHO LEE[1,3], (Member, IEEE)

[1]Department of Data Science, Seoul National University of Science and Technology, Seoul 01811, South Korea
[2]CryptoLab, Seoul 08826, South Korea
[3]Department of Industrial Engineering, Seoul National University of Science and Technology, Seoul 01811, South Korea

Corresponding author: Younho Lee (younholee@seoultech.ac.kr)

This study was supported by the Research Program funded by the SeoulTech (Seoul National University of Science and Technology).

*Boyoung Han and Hojune Shin are co-first authors.

ABSTRACT Although there has been significant progress in homomorphic encryption (HE) technology, a fully homomorphic Naive Bayes (NB) classifier capable of training on HE-encrypted data without decryption has not yet been efficiently developed. This research introduces a new method for approximating homomorphic logarithm calculations with an average relative error under 0.01%. Leveraging the SIMD functionality of the HE framework and a GPU, this technique can compute logarithm values for thousands of encrypted probabilities in about 2.5 seconds. Building upon this, we present a more efficient fully homomorphic NB classifier. Our method can train on a breast cancer dataset in roughly 14.3 seconds and perform query inferences in 0.84 seconds. Compared to the recent privacy-protecting NB classifier from Liu et al. in 2017, which offers a similar security level, our method is estimated to be about 28 times faster.

INDEX TERMS Naive Bayes classifier, privacy-preserving machine learning, CKKS, fully homomorphic encryption.

I. INTRODUCTION

The swift advancement in artificial intelligence (AI) technology has spurred a growing need for AI applications in sectors like healthcare and finance, where user data is highly sensitive and private. Due to stringent legal regulations [1], [2], this type of information cannot be utilized unless its privacy is safeguarded. Furthermore, models developed from training with such sensitive data hold substantial commercial value, making the security of the model information crucial. This paper explores how the Naive Bayes (NB) machine learning algorithm can be applied to privacy-sensitive data in a real-world system context, ensuring data privacy throughout the process.

The NB classifier is renowned for its simplicity and accuracy, and is recognized for its scalability. Its ease of implementation and effectiveness with limited data make it a popular choice for applications such as text classification, sentiment analysis, system performance management, collaborative filtering, and clinical decision support systems [3], [4], [5], [6], [7], [8], [9], [10], [11].

Our research is centered on the privacy-preserving Naive Bayes (PPNB) classifier, which employs homomorphic encryption (HE). In this setup, the training process does not necessitate decrypting the intermediate data at any point. As a result, the cloud server can independently complete the training once it has received the ciphertexts containing the training data. Additionally, the inference process is entirely conducted by the cloud server using just the encrypted input data, thereby obviating the need for decrypting any intermediate ciphertexts before arriving at the inference result.

The associate editor coordinating the review of this manuscript and approving it for publication was Sedat Akleylek.

Existing research has faced challenges in achieving this goal due to two main reasons. First, there has been no efficient method to implement complex functions, such as the homomorphic logarithm function [12], [13], [14], [15], [16]. Second, certain homomorphic operations, like division or multiplication over encrypted multi-precision integer values, are prohibitively expensive [17]. To address these issues, the costly operations are carried out after decrypting the data. Following this, the computation results are re-encrypted and sent back to the cloud to resume training or inference processes.

Contribution: This paper introduces a novel approach enabling a server to construct a training model exclusively through homomorphic operations on encrypted training data, without requiring interaction with the data owner or a trusted party possessing decryption keys.

The data owner encrypts her data using the system's public key and transmits it to the server. Subsequently, the server independently generates and stores the model using solely homomorphic operations. Notably, this study marks the first implementation of a homomorphic encryption-based PPNB method that avoids intermediate plaintext computation post-encryption of training data.

Our method efficiently approximates the natural logarithm of a probability value encrypted by CKKS, termed ApproxLog(), and requires about 2.5 seconds in a GPU-assisted desktop environment. Utilizing the SIMD function, we can compute over 1000 logarithmic values simultaneously, achieving an accuracy with a relative error of less than 0.01% for most of the input range (0,1].

Our PPNB method is notably efficient for practical applications. It processes encrypted models and data, requiring around 14.3 seconds for training on the Breast Cancer dataset [18] and just 0.84 seconds for a single inference. The amortized time is further reduced to about 4.65ms when considering that 180 inputs can be encrypted together for inference. This represents a substantial speed increase compared to Liu et al.'s [17], which took 2951.8 minutes for training and 348.8 minutes for classification, making our method about 12,000 times faster for training and 24,900 times faster for classification.

To accommodate different computational environments, such as GPU utilization and the use of Residue Number System (RNS), we adjusted our comparison of multiplication operation speeds between our proposed PPNB method and [17]. Specifically, we found that the multiplication operation in [17] is approximately 430.77 times slower compared to our PPNB implementation under similar conditions.

To assess the performance improvements of our method in terms of computational complexity, we conducted a comparative analysis with [17] focusing on homomorphic addition, multiplication, and slot rotation operations—fundamental operations provided by homomorphic encryption (HE). Specifically, we examined the multiplicative depth of these operations, which directly impacts bootstrapping requirements. Our findings highlight that our proposed method demonstrates superior performance in multiplication operations relative to the number of slots accommodated in a ciphertext. Particularly, we observed a substantial difference in the multiplication operation depth between our method and [17] during the training phase. In our method, the multiplication depth remains constant (O(1)), whereas in [17], it scales with $O(l_{data} \cdot n_{max})$, dependent on the size of the data and the maximum number of categories (n_{max}).

The remainder of this paper is structured as follows. Section II presents the necessary background information for understanding this work. Section III details the system and security models of the proposed method. This is followed by the proposed homomorphic logarithm technique in Section IV and the proposed PPNB training and inference methods in Section V. The experimental results and computational complexity analysis are provided in Sections VI and VII, respectively. Section VIII offers a comprehensive noise analysis of the proposed method and discusses its extension when the Key Manager is semi-honest. Section IX reviews related work. Finally, the conclusion is presented in Section X.

II. BACKGROUNDS

A. NOTATION

The notations used in this paper are listed in **Table 1**.

B. NAIVE BAYES CLASSIFIER

Consider a vector of values $\vec{x} = (x_0, \cdots, x_{d-1})$, which belongs to a set of categorical variables $X(= X_0 \times X_1 \times \cdots \times X_{d-1})$, and a target categorical variable $Y = \{y_1, \ldots, y_u\}$. The NB classifier determines the classification result $y_r \in Y$ in the following manner:

$$\begin{aligned} y_r &:= \underset{1\le i\le u}{\text{argmax}}\, Pr[Y=y_i|X=\vec{x}] \\ &= \underset{1\le i\le u}{\text{argmax}}\, \frac{Pr[Y=y_i]\cdot Pr[X=\vec{x}|Y=y_i]}{Pr[X=\vec{x}]} \\ &= \underset{1\le i\le u}{\text{argmax}}\, Pr[Y=y_i]\cdot Pr[X=\vec{x}|Y=y_i] \\ &= \underset{1\le i\le u}{\text{argmax}}\, Pr[Y=y_i]\cdot \prod_{j=0}^{d-1} Pr[X_j=x_j|Y=y_i] \\ &= \underset{1\le i\le u}{\text{argmax}}\, logPr[Y=y_i] + \Sigma_{j=0}^{d-1} logPr[X_j=x_j|Y=y_i] \end{aligned} \tag{1}$$

Among the equations mentioned earlier, (1) arises from assuming that each element in the input vector is independently sampled. This allows the application of Bayes' rule.

C. CKKS (CHEON-KIM-KIM-SONG) HE

CKKS encryption does not eliminate the noise associated with the message, which persists as an error in the decrypted output [19]. This error is treated as a computational cost, enabling efficient multiplication operations on encrypted complex numbers. Despite introducing small errors during

TABLE 1. Notations and conventions.

Symbol	Meaning
[a,b]	The collection of integers ranging from a to b, inclusive of both endpoints.
X_i	Categorical variables that are independent of each other (features) ($i \in \mathbb{N}$)
d	Count of distinct independent categorical variables (features)
n_i,n_{max}	n_i: # of categories in X_i, $n_{max} = \max_i(n_i)(i \in [0, d-1])$
Y, y_t	Target categorical variable and # of categories of Y
$\vec{z}$, $[\![z]\!]$	$\vec{z} = (z_0, z_1, \cdots, z_{M-1}) \in \mathbb{R}^M$ and $[\![z]\!]$ is its encryption. $z \in \{x, y\}$ (M is in Table 2.)
$[\![x]\!][i](c[i])$	The value in i-th slot in $[\![x]\!](c)$ ($i \in [0, M-1]$)
evk,sk,pk	evaluation key, secret key and public key
(sk_s, pk_s), (sk_i, pk_i)	System's key pair and User i's key pair
$c_{\mathbf{X}}^{(\mathbf{Y})}$,$\bar{c}_{\mathbf{X}}^{(\mathbf{Y})}$,$\tilde{c}_{\mathbf{X}}^{(\mathbf{Y})}$, $\hat{c}_{\mathbf{X}}^{(\mathbf{Y})}$	A ciphertext. $\mathbf{X}$ represents indices and $\mathbf{Y}$ is an additional symbol offering further details about the ciphertext.
$\vec{b}_{X_i,j}([\![b_{X_i,j}]\!])$, l_{data}	(encrypted) bin mask vector (BMV) for value j in the categorical variable X_i. Its length is l_{data}, i.e. the number of rows in the training data. $\vec{b}_{X_i,j}$ can be encrypted to multiple ciphertexts if $l_{data} > M$.
$\vec{b}_{Y,k}([\![b_{Y,k}]\!])$	(encrypted) bin mask vector (BMV) for value k in the target variable Y.
S_X, S_{n_i}, S_Y, S_{Xn_iY}, S_u	$S_X = [0, d-1]$, $S_{n_i} = [1, n_i]$, $S_Y = [1, y_t]$, $S_{Xn_iY} = S_X \times S_{n_i} \times S_Y$,$S_u = [0, m-1]$
m	# of users in the system
N,N_{i1},N_i	$N = \sum_{i \in S_X} n_i$, $N_{i1} = N + 1$, $N_i = \sum_{j \in [0, i-1]} n_j$
t,α	2^{-t}: the smallest number that **ApproxLog()** works. α: Laplace smoothing factor (ref. Section V-C)
N_{tot}, s_{ctxt}, $s_{\mathbf{ID}}$	$N_{tot} = N_{i1} * y_t$, s_{ctxt}=(size of a ciphertext) (14MB in our parameter), $s_{\mathbf{ID}}$=(size to represent an ID)
$\vec{1}^{(i)}$	A vector where i-th slot (component) has 1 and the other slots are zero. It can be used for representing a ciphertext that contains it.
$\vec{1}^{[a,b]}$	A vector where the slots between a-th and b-th have 1 including the border and the other slots are zero. It can be used to represent the ciphertext containing it. If $(a > b)$, it means the zero vector where all slot values are zero.
$(\vec{1}^a \| \vec{0}^b)^f$	A vector where $(i*(a+b))$-th$\sim$ $(i*(a+b)+a-1)$-th slots are 1 and $(i*(a+b)+a)$-th$\sim$ $(i*(a+b)+b-1)$-th slots are 0 for all $i \in [0, f-1]$. It also can be used to represent the ciphertext of the vector.
$b \leftarrow_R S$	b is uniformly chosen from S
ns	Number of slots of valid values ($ns \leq M$)
$\overrightarrow{input}$	The set of the values in the independent variables as an input for inference in Fig. 10

operations in an encrypted state, CKKS remains viable for implementing privacy-preserving versions of various applications, especially in fields like machine learning where minor calculation discrepancies using IEEE floating point representation do not significantly affect overall performance.

CKKS supports the following algorithms:

- $\mathsf{KeyGen}(1^\lambda)$ takes a security parameter λ as input and returns pk, sk, and evk.
- $\mathsf{Enc}_{pk}(\vec{x})$ outputs $[\![x]\!]$ that maintains the vector structure as $\vec{x}$.
- $\mathsf{Dec}_{sk}([\![x]\!])$ returns $\vec{x}$ if $[\![x]\!]$ is a valid encryption of $\vec{x}$, resulting from Enc or generated through operations on valid ciphertexts using the correct pk, evk, and sk. Otherwise, it returns $\perp$.

TABLE 2. Parameters of RNS-CKKS used.

	Description	Values
M	Number of slots	32768
$log_2(PQ)$	Maximum modulus bit of a ciphertext	1555
$log_2 p$	Quantization bit size	42
Supported multiplication level per a bootstrapping		9

- $\mathsf{Add}([\![x]\!], [\![y]\!])(\mathsf{Sub}([\![x]\!], [\![y]\!]))$ outputs a new ciphertext $[\![\vec{x} + \vec{y}]\!]$ ($[\![\vec{x} - \vec{y}]\!]$).
- $\mathsf{Add}([\![x]\!], k)(\mathsf{Sub}([\![x]\!], k))$ outputs a new ciphertext that is an encryption of $(x_0 + k, \cdots, x_{M-1} + k)$ $((x_0 - k, \cdots, x_{M-1} - k))$ for given $k \in \mathbb{C}$.
- $\mathsf{Level}([\![x]\!])$ returns $[\![x]\!]$'s level l, the number of further possible multiplication with $[\![x]\!]$.
- $\mathsf{Mult}_{evk}([\![x]\!], [\![y]\!])$ returns an (approximate) encryption of $(x_0 * y_0, \cdots, x_{M-1} * y_{M-1})$ whose level is $min(\mathsf{Level}([\![x]\!]), \mathsf{Level}([\![y]\!])) - 1$. ($min(A, B)$ returns the minimum value among A and B.)
- $\mathsf{Mult}_{evk}([\![x]\!], k)$ outputs an encryption of $(kx_0, \cdots, kx_{M-1})$ where $k \in \mathbb{C}$. The level of resultant ciphertext is decremented from the level of $[\![x]\!]$ by 1.
- $\mathsf{Rot}_{evk}([\![x]\!], i)$ returns an encryption of $(x_{M-i}, x_{M-(i-1)}, \cdots, x_{M-1}, x_0, \cdots, x_{M-(i+1)})$, if $i \in [0, M-1]$. If i is negative, it refers to $M + i$.
- $\mathsf{Boot}_{evk}([\![x]\!])$ returns a fresh ciphertext $[\![x']\!]$ that has approximation of $\vec{x}$ if $\mathsf{Level}([\![x]\!]) \geq l_{min_{boot}}$, the number of required multiplication level to perform $\mathsf{Boot}()$. $l_{min_{boot}}$ depends on what bootstrapping algorithm is used and parameter.

We assume that the rescaling algorithm [19] operates within the $\mathsf{Mult}()$ function. Additionally, for simplicity, we have omitted details about the keys used. Our implementation employs RNS-CKKS, leveraging GPU acceleration to enhance performance [20], [21], [22], [23].

Table 2 shows the parameters for setting up CKKS.

D. APPROXIMATE FUNCTIONS ON CKKS CIPHERTEXTS

We employ $\mathsf{ApproxSign}([\![x]\!])$, which takes a ciphertext $[\![x]\!]$ and outputs an encrypted vector $(a_0, \cdots, a_{M-1})$, where $a_i = 1$ if $[\![x]\!][i] > 0$, and $a_i = -1$ if $[\![x]\!][i] < 0$ for $i \in [0, M-1]$. This is implemented using the method described by [24] in the HEAAN library [25]. Additionally, we utilize $\mathsf{ApproxInv}([\![x]\!])$, which produces an encryption of the multiplicative inverse of the values in $[\![x]\!]$. This function is implemented using the Newton approximation method [25].

III. SYSTEM MODEL

This section outlines the system setup and presents an overview of the proposed PPNB protocol.

A. PLAYERS AND PROTOCOL OVERVIEW

An overview of the protocol is illustrated in Fig. 1. Below is a description of the three participants in the protocol:

- Data Owners (DO_i $(i \in S_u)$) encrypt their data using the system public key (pk_s) and send the encrypted

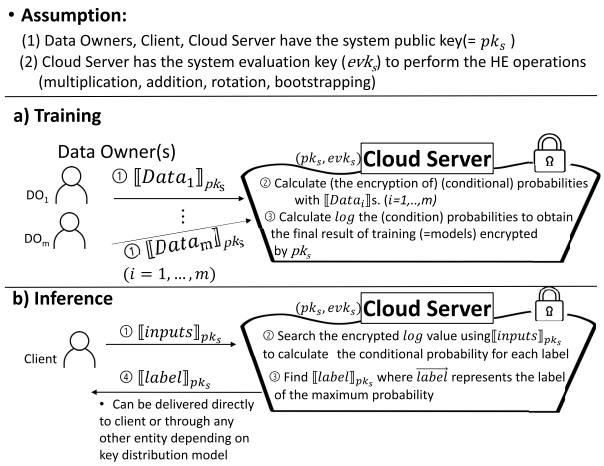

FIGURE 1. Protocol overview and assumption.

data ($[\![Data_i]\!]_{pk_s}$ in Fig. 1) to the cloud server (CS) for training.

- CS holds both the system public key (pk_s) and the evaluation key (evk_s). It carries out training on the encrypted data received from the data owners. Since the data is encrypted with pk_s, CS can perform CKKS HE operations without needing key switching. Once the model is trained, CS can process encrypted inference queries from clients and provide the encrypted results using pk_s, tagged with the requesting user's ID.
- Client encrypts their input data for inference with pk_s (represented as $[\![inputs]\!]pk_s$ in Fig. 1) and sends it to CS. The client then receives the inference result, encrypted with pk_s (shown as $[\![label]\!]pk_s$ in Fig. 1).

B. ASSUMPTION AND PROBLEM DEFINITION

Assumptions: Initially, we assume that a common system public key is shared among all participants. And the second assumption is that the evaluation key for performing homomorphic operations are provided to the CS, as shown in Fig. 1. This key distribution can be facilitated by a trusted third party, as described in [26], or in a secure outsourcing context where the data owner, who delegates their data to the CS for processing, generates the keys and supplies both the public and evaluation keys to the CS. In the environment of [26], data collected from various organizations is combined and analyzed, and in this case, due to legal constraints, the analysis results must be reviewed by a trusted third party before being sent to Client. We provide two instances of the system model, where the entity responsible for key generation and distribution can either be a trusted third party or a semi-honest, described in Section VIII-B and Section VIII-C, respectively.

Problem Definition: This research introduces a protocol for privacy-preserving NB training and inference, which needs to satisfy the following criteria.

1) The CS receives encrypted training data using the system public key and independently generates an encrypted model (the result of training). This process does not require assistance from other participants or the use of decryption keys, enabling the model to be used for inference.
2) The CS can compute encrypted inference results using the system public key without decryption or assistance from other entities. This is based on input data encrypted with the system public key received from the Client.
3) Under the honest-but-curious (HBC) model, the CS, Client, and Data Owners do not have access to information regarding the training data, inference results, or the model itself. The training model remains concealed from all involved parties.
4) The CS operates exclusively within a single security domain. In scenarios involving multiple CSs, there exists a risk of potential malicious cooperation if they act adversarially.

According to discussions in [26], 1) is crucial in scenarios where multiple owners of encrypted training data need to collaborate for training purposes. In such cases, restrictions on the use of decryption keys during training or inference processes may apply. Moreover, legal frameworks often impose constraints on the utilization of decryption keys during training, as highlighted in [26]. In the context of secure outsourcing, 1) offers advantages by reducing the resource requirements of Data Owners, enabling them to operate offline after transferring their data to the CS.

2) relates to the efficiency of communication costs and reducing security risks by limiting the use of encryption keys, as mentioned in [27]. 3) focuses on security, ensuring the protection of valuable model information while maintaining the privacy of participants' data used in training and inference. Lastly, 4) addresses scenarios where using CSs from different security domains is not feasible. Using multiple CSs not only incurs high costs but also heightens the potential for undetectable collusion between them. Therefore, this study assumes the CS operates within a single security domain.

C. SECURITY MODEL

Similar to many prior studies, our assumption is that each participant in the protocol might act as an adversary, following the behavior defined in the honest-but-curious (HBC) model, with the exception of KM. Assuming KM is trustworthy as described in Section V-B, there is limited concern because CKKS is already established as CPA-secure, CS handles only encrypted data, and Client receives solely the inference result. However, if KM operates under semi-honest assumption, it becomes necessary to verify the privacy of the model and the inference data in relation to KM. This concern is addressed in Section VIII-C.

IV. HOMOMORPHIC LOGARITHM FUNCTION OVER ENCRYPTED DATA

The core of homomorphic PPNB classification involves multiplying probabilities and performing the argmax operation on the resulting values. However, many homomorphic encryption (HE) schemes that support precise calculation results incur high computational costs for high-precision homomorphic multiplications [17]. Additionally, using approximate HE schemes for homomorphic multiplication on very small encrypted data does not achieve high accuracy due to noise introduced during encryption.

To address this, recent homomorphic PPNB methods [16], [27] transform all probability values into their logarithmic form, replacing multiplication with addition for classification. However, logarithmic operations on HE data are not well-studied, leading many existing approaches to compute logarithms of probabilities only after decryption. This presents challenges when decryption keys are restricted during training. In this study, we propose a homomorphic logarithm calculation method (ApproxLog()) that ensures high accuracy and manages encrypted values within the range $(\epsilon, 1]$, where ϵ is a very small positive number. This method can be applied to any approximate HE scheme with SIMD functions, such as CKKS [19], allowing for polynomial computations over encrypted x.

To our knowledge, no existing literature addresses the homomorphic logarithm operation on encrypted data within the range (0,1]. The open left range causes polynomial approximation methods like Remez [28] to have high approximation errors when applied directly. Additionally, using Newton's approximation [29] directly results in substantial errors when approximating the log function with inputs in the range (0,1].

Our ApproxLog() utilizes a well-established polynomial approximation method for logarithmic operations as proposed in [30]. It first computes $s = (x-1)/(x+1)$ on the input x, followed by evaluating $s * (L_1 s^2 + L_2 s^4 + L_3 s^7 + L_4 s^8 + L_5 s^{10} + L_6 s^{12} + L_7 s^{14})$ to derive the result. The coefficients $L_1 \sim L_7$ are specifically chosen for computations with encrypted inputs using CKKS, characterized by their small absolute values, all less than 0.1 (exact values are detailed in Section XI). This method demonstrates a high level of accuracy, with a maximum approximation error reaching up to $2^{-58.45} \approx 10^{-17.6}$; however, such precise accuracy is attainable only when the input falls within the range $[1/\sqrt{2}, 2/\sqrt{2}]$. We denote this approach as HermesLog() [30].

The contribution of this research is to develop a method for transforming encrypted data from an original range of (0,1] into a new encrypted range $[1/\sqrt{2}, 2/\sqrt{2}]$ so that the transformed data can be an input for HermesLog() [30]. After applying our proposed transformation to the encrypted data, we then use HermesLog() on this transformed data. Finally, we complete the process by subtracting the outcome obtained from HermesLog(), applied to the transformed data, from the logarithm of the multiplier used in the initial conversion of the original input.

The challenge with this issue lies in the complexity of determining the scaling factor required to transform the input into the range $[1/\sqrt{2}, 2/\sqrt{2}]$, especially since the input value is in an encrypted state.

Algorithm 1 Proposed Approximate Logarithm Algorithm

procedure ApproxLog($[\![input]\!]$) $\vec{input} = (\overbrace{x, x, \ldots, x}^{t+2}, 0, \ldots, 0)$
  $c_{\mathbf{inp}} \leftarrow [\![input]\!]$
  $c_{\mathbf{sign}} \leftarrow \mathbf{ApproxSign}(\mathbf{Sub}(T_1, c_{\mathbf{inp}}))$
  $c_0 \leftarrow \mathbf{Mult}(\mathbf{Mult}(\mathbf{Add}(c_{\mathbf{sign}}, \mathbf{Rot}(c_{\mathbf{sign}}, 1)), 0.5), 1^{[0,t+1]})$
  $c_1 \leftarrow \mathbf{Sub}(\vec{1}^{[0,t+1]}, \mathbf{Mult}(c_0, c_0))$
  $c_{\mathbf{h}} \leftarrow \mathbf{Mult}(\mathbf{Mult}(c_1, T_2), c_{\mathbf{inp}})$
  $c_{\mathbf{r}} \leftarrow \mathbf{Mult}(c_1, \mathbf{HermesLog}(c_{\mathbf{h}}))$
  $c_{\mathbf{u}} \leftarrow \mathbf{Sub}(c_{\mathbf{r}}, \mathbf{Mult}(c_1, T_3))$
  $c_{\mathbf{ret}} \leftarrow \mathbf{SumGroup}(c_{\mathbf{u}}, t+2, t+2)$
  return $c_{\mathbf{ret}}$ // ($c_{\mathbf{ret}}$ has $log_e(x)$ in 0th slot)
end procedure

A. PROPOSED METHOD

The proposed approach computes the natural logarithm of the encrypted input $x \in (0, 1]$ using the following steps: 1) Identify a such that $x \times 2^a$ falls within the interval $[1/\sqrt{2}, 2/\sqrt{2}]$. 2) Apply the function HermesLog($x \times 2^a$), yielding the result $r - a \times log_e(2)$ to obtain the final logarithmic value.

On encrypted x, we can determine a as follows.

$$\text{If } 2^{-b} < x \leq 2^{-(b-1)} \text{ Then } a \leftarrow b - 1/2, \tag{2}$$

If a is computed as described earlier, $x \times 2^a = x \times 2^{(b-1/2)} \in (1/\sqrt{2}, 2/\sqrt{2}]$. To verify if x falls within a specific range for b as mentioned previously, we utilize ApproxSign(). The challenge arises from the need to perform this comparison operation multiple times in an encrypted state to determine the appropriate b, since x remains encrypted. To efficiently execute these comparisons, we replicate x across multiple slots and compare each instance with 2^{-b} values for various b. An example of implementing this method is illustrated in Fig. 2. Assuming the minimum value of x is 2^{-t}, the core of the method involves Steps 2∼5 in Fig. 2. By utilizing ApproxSign(), we ascertain that x lies within the range $(2^b, 2^{b-1}]$ for a specific b.[1] Consequently, we can apply HermesLog() after scaling x appropriately and feeding the result of this multiplication as input.

Moreover, when t is significantly smaller than M within a ciphertext, we can concurrently compute logarithmic values for $\lfloor M/(t+2) \rfloor$ numbers using the aforementioned approach by performing ApproxSign() only once after consolidating all inputs into a single ciphertext. Setting t to approximately 20 allows us to compute the logarithm for over a thousand values simultaneously, considering the total number of slots (= 32768) in our configuration. The

[1]This does not imply that we know b in plaintext; rather, we possess a ciphertext indicating the correct b

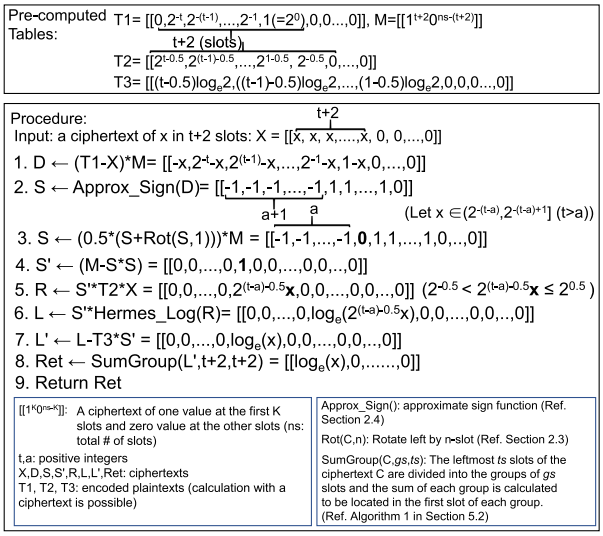

FIGURE 2. **Proposed** ApproxLog**()- An example of execution.**

algorithmic description of **ApproxLog()** is provided in Alg. 1. Note that pre-computed ciphertexts T_1, T_2, T_3 are depicted in Fig. 2.

V. PRIVACY-PRESERVING NAIVE BAYESIAN TRAINING AND CLASSIFICATION ALGORITHMS

A. DATA ORGANIZATION AND REPRESENTATION

We represent the training data as a table, where each column corresponds to a categorical variable. Each value in a variable is binary-encoded using one-hot encoding. For instance, if X_1 has a value of two and $n_1 = 3$, it is encoded as $(0, 1, 0)$. We aggregate the component values from these encoded vectors at the same position across all variables to form a vector of collected values. As an example, we can create a vector specifically for the first component of the one-hot encoded vectors from X_1's values. This vector is referred to as a binary mask vector (BMV), denoted as $\vec{b}_{X_1,1}$, which manages the values in the 1st component of one-hot encoded vectors corresponding to X_1. To encompass values across all features $X_0, \ldots, X_{d-1}$, we need $(n_0 + \ldots + n_{d-1})$ BMVs.

B. SUBROUTINES

The subroutines utilized are **SumGroup()** and **FindMaxPos()**. **SumGroup**($[\![a]\!]$, gs, ts) groups the values in $[\![a]\!]$ into segments of gs slots to compute the sum of values within each segment. When considering cases where only a subset of slots in $[\![a]\!]$ are active, the number of slots to be utilized is specified by ts. The computed sum is stored in the first slot of each segment, starting from the leftmost slot in a ciphertext. With slight modifications, we can implement **SumGroup**$_z$(), where the spacing between adjacent elements is not necessarily one but can be an arbitrary number z slots apart.

FindMaxPos(c,ns) returns a new ciphertext where the slot position containing the maximum value among the first ns slots in c is set to one, while all other slots are set to zero. The execution of **FindMaxPos**(c,ns) proceeds as follows: it identifies the largest value among the inputs using **FindMax**(c,ns), then constructs a ciphertext (c_{max}) that fills the first ns slots exclusively with this maximum value. Subsequently, c_{max} is subtracted from the input ciphertext, followed by adding a very small ϵ value to each slot in the resulting subtraction (c_{in}). The **ApproxSign**(c_{in}) function is then applied to obtain c_{out}. Finally, the resulting ciphertext c_{out} is adjusted by adding 1 to each slot and then multiplying every slot by 0.5 to yield the final result. Algorithms 2 and 3 provide detailed descriptions of **SumGroup()** and **FindMaxPos()**, respectively.

The following algorithms are the subroutines to implement FindMaxPos().

1) FINDMAX() ALGORITHM

The function **FindMax**($[\![input]\!]$, ns) returns a ciphertext where the first slot contains the maximum value among the leftmost ns slots of $[\![input]\!]$. The number of calls to **ApproxSign()** is $\lceil \log_2(ns) \rceil$. Detailed steps are provided in Algorithm 4.

2) COPYSLOT() ALGORITHM

This function generates a new ciphertext where the first j slots are filled with the value from the i-th slot of the input ciphertext. It uses **Copy2PSlot()** as a subroutine. The **Copy2PSlot**(c, i, n) function creates a new ciphertext where the first i slots contain the same value as c, and the value from the i-th slot in c is copied to the $(i + 1)$ to $(i + 2^n)$ slots in the new ciphertext. The **CopySlot** algorithm is detailed in Algorithm 5, and **Copy2PSlot()** is described in Algorithm 6.

Algorithm 2 SumGroup Algorithm

```
procedure SumGroup([[input]], gs, ts)
    [[a]] ← [[input]]
    while gs > 1 do
        if gs is even then
            m⃗ ← (1⃗^(gs/2) || 0⃗^(gs/2))^⌈ts/gs⌉
            [[r]] ← Rot([[a]], gs/2)
            [[t1]] ← Mult(m⃗, [[r]]), [[t2]] ← Mult(m⃗, [[a]])
            [[a]] ← Add([[t1]], [[t2]]), gs ← gs/2
        else
            m⃗_1 ← (1⃗^((gs-1)/2) || 0⃗^((gs+1)/2))^⌈ts/gs⌉
            m⃗_2 ← (1⃗^((gs+1)/2) || 0⃗^((gs-1)/2))^⌈ts/gs⌉
            [[r]] ← Rot([[a]], (gs+1)/2)
            [[t1]] ← Mult(m⃗_1, [[r]]), [[t2]] ← Mult(m⃗_2, [[a]])
            [[a]] ← Add([[t1]], [[t2]]), gs ← (gs+1)/2
        end if
    end while
    return [[a]]
end procedure
```

C. TRAINING

The training process of the plaintext version of the Naive Bayes (NB) protocol involves tallying the number of rows

Algorithm 3 FindMaxPos Algorithm
(ns: the number of slots having input)

procedure FindMaxPos($[\![input]\!]$, ns)
  $c_{\mathbf{max}} \leftarrow$ **FindMax**($[\![input]\!]$,ns) %max value in 0th slot
  $c_{\mathbf{copy}} \leftarrow$ **CopySlot**($c_{\mathbf{max}}$,0,ns) %max value is copied to the leftmost ns slots
  $c_{\mathbf{in}} \leftarrow$ **Add**(**Sub**(c, $c_{\mathbf{copy}}$), ϵ) %ϵ: tiny positive value
  $c_{\mathbf{out}} \leftarrow$ **ApproxSign**($c_{\mathbf{in}}$)
  $c_{\mathbf{out}} \leftarrow$ **Mult**(**Add**($c_{\mathbf{out}}$, $\vec{1}^{ns}$), 0.5)
  return $c_{\mathbf{out}}$
end procedure

Algorithm 4 FindMax Algorithm

procedure FindMax($[\![input]\!]$, ns)
  if $ns == 1$ **then**
    return $[\![input]\!]$
  end if
  if ns is odd **then**
    Assume we know the minimum value that can be in $[\![input]\!]$, put it into $(ns+1)$-th slot in $[\![input]\!]$.
    $ns \leftarrow ns + 1$
  end if
  $c_0 \leftarrow$ **Mult**($[\![input]\!]$, $\vec{1}^{[0,\frac{ns}{2}-1]}$)
  $c_1 \leftarrow$ **Rot**(**Mult**($[\![input]\!]$, $\vec{1}^{[\frac{ns}{2},ns-1]}$), $\frac{ns}{2}$)
  $c_{\mathbf{cmp}} \leftarrow$ **ApproxSign**(**Sub**(c_0, c_1))
  $c_{0>1} \leftarrow$ **Mult**(**Mult**(**Add**($\vec{1}^{[0,\frac{ns}{2}-1]}$, $c_{\mathbf{cmp}}$), 0.5), c_0)
  $c_{1>0} \leftarrow$ **Mult**(**Mult**(**Sub**($\vec{1}^{[0,\frac{ns}{2}-1]}$, $c_{\mathbf{cmp}}$), 0.5), c_1)
  $c_{\mathbf{out}} \leftarrow$ **Add**($c_{0>1}$, $c_{1>0}$)
  return FindMax($c_{\mathbf{out}}$, $ns/2$)
end procedure

for each combination of values in X_i and Y, as well as counting the occurrences of each category value in Y from the dataset. This allows for straightforward computation of $Pr[X_i = j|Y = k]$ and $Pr[Y = k]$, where $(i, j, k) \in S_{Xn_iY}$. These computations collectively form the model for a Naive Bayes classifier. Fig. 3 provides a detailed description of the proposed PPNB training algorithm. In Step 1-(1), $\mathsf{cnt}i, j, k[0]$ stores the count of rows where $X_i = j$ and $Y = k$. Step 2 updates $\mathsf{cnt}Y_k[0]$ with the count of rows where $Y = k$. In Step 7, the inverse of this count is computed. These counts are then multiplied in Step 8 to derive the conditional probability $Pr[X_i = j|Y = k]$. Steps 9 to 11 outline the process for computing the logarithmic values of these probabilities.

Algorithm 5 CopySlot Algorithm

procedure CopySlot($[\![input]\!]$, i, j)
  $c_1 \leftarrow$ **Rot**(**Mult**($[\![input]\!]$, $\vec{1}^{(i)}$), i)
  $ind \leftarrow j - 1$, $cp \leftarrow 0$
  while $ind > 0$ **do**
    $c_1 \leftarrow$ **Copy2PSlot**(c_1, cp, $\lfloor log_2(ind) \rfloor$)
    $cp \leftarrow cp + 2^{\lfloor log_2(ind) \rfloor}$
    $ind \leftarrow ind - 2^{\lfloor log_2(ind) \rfloor}$
  end while
  return c_1
end procedure

Algorithm 6 Copy2PSlot Algorithm

procedure Copy2PSlot($[\![input]\!]$, i, n)
  $c_1 \leftarrow$ **Rot**(**Mult**($[\![input]\!]$, $\vec{1}^{(i)}$), -1)
  for $j = 1$ to n **do**
    $c_1 \leftarrow$ **Add**(**Rot**(c_1, -2^{j-1}), c_1)
  end for
  $c_{\mathbf{ret}} \leftarrow$ **Add**(**Mult**(c, $\vec{1}^{[0,i-1]}$), c_1)
  return $c_{\mathbf{ret}}$
end procedure

Input:
• **CS** *has* $[\![b_{X_i,j}]\!]$, $[\![b_{Y,k}]\!]$ where $\overrightarrow{b_{X_i,j}}$, $\overrightarrow{b_{Y,k}}$ for all $(i,j,k) \in S_{Xn_iY}$.
• **CS** has a set of ciphertext $_v$ where $\hat{c}_v[w] = 0$ if $\mathsf{ind}_{v,w} \bmod N_{t1} = 0$ else $\hat{c}_v[w] = \alpha * n_i$ where $N_i + 1 \leq \mathsf{ind}_{v,w} \bmod N_{t1} < N_{i+1} + 1$, $\mathsf{ind}_{v,w} = (v * M + w)$, for all $(v, w) \in [0, \lfloor N_{tot}/M \rfloor] \times [0, M - 1]$,
Output:
• **CS** obtains the ciphertexts of models $\mathsf{c}_i^{(model)}$ for all $i \in [0, \lfloor N_{tot}/M \rfloor]$, which are the encrypted logarithm values of all (conditional) probabilities $\{Pr[Y = k], \{Pr[X_i = j|Y = k]\}_{i\in[0,d-1],j\in[1,n_i]}\}_{k\in[1,y_t]}$
CS:
1. For all $(i,j,k) \in S_X \times S_{n_i} \times S_Y$ nosep,label=(0)
  1) $c_{i,j,k}^{(cnt)} \leftarrow$ **SumGroup**(Mult($[\![b_{X_i,j}]\!]$, $[\![b_{Y,k}]\!]$), M, M)
  2) $c_{i,j,k}^{(cnt)} \leftarrow \mathsf{Add}(\alpha, c_{i,j,k}^{(cnt)})$
  3) Copy $\mathbf{cnt}_{i,j,k}[0]$ to the $\bar{c}_{p(i,j,k)}[\mathbf{ind}(i,j,k)]$ where $p(i,j,k) = \lfloor \beta_{i,j,k}/M \rfloor$, $\mathbf{ind}(i,j,k) = \beta_{i,j,k} \bmod M$, $\beta_{i,j,k} = N_{t1} \times (k-1) + (N_i + j - 1)$.
2. $c_k^{(cntY)} =$ **SumGroup**($[\![b_{Y,k}]\!]$, M, M) for all $k \in S_Y$
3. $c^{(sumY)} \leftarrow \sum_{k\in S_Y} c_k^{(cntY)}$.
4. Copy $c_k^{(cntY)}[0]$ into every slot in $\tilde{c}_{q(k)}[\mathsf{ind}_{Y_k}(k)] \sim \tilde{c}_{q'(k)}[\mathsf{ind}_{Y'_k}(k)]$ respectively, where $q(k) = \lfloor \gamma_k/M \rfloor$, $\mathsf{ind}_{Y_k}(k) = \gamma_k \bmod M$, $q'(k) = \lfloor (\gamma_k + N_{t1} - 1)/M \rfloor$, $\mathsf{ind}_{Y'_k}(k) = (\gamma_k + N_{t1} - 1) \bmod M$, $\gamma_k = N_{t1} \times (k-1) + 1$. for all $k \in S_Y$.
5. Copy $c^{(sumY)}[0]$ into $\tilde{c}_{r(k)}[\mathsf{ind}_Y(k)]$ and Copy $c_k^{(cntY)}[0]$ into $\bar{c}_{r(k)}[\mathsf{ind}_Y(k)]$. where $r(k) = \lfloor (\gamma_k - 1)/M \rfloor$ and $\mathsf{ind}_Y(k) = (\gamma_k - 1) \bmod M$ for all $k \in S_Y$.
6. $\tilde{c}_i \leftarrow$ **Add**($\hat{c}_i$, $\tilde{c}_i$) ($i \in [0, \lfloor N_{tot}/M \rfloor]$)
7. $\tilde{c}_i \leftarrow$ **ApproxInv**($\tilde{c}_i$) ($i \in [0, \lfloor N_{tot}/M \rfloor]$)
8. $\bar{c}_i^{(model)} \leftarrow \mathsf{Mult}(\bar{c}_i, \tilde{c}_i)$ for all $i \in [0, \lfloor N_{tot}/M \rfloor]$
9. Copy $\bar{c}_j^{(model)}[i]$ to every slot in $(\tau(i,j) * (t+2))$th $\sim (\tau(i,j) * (t+2) + t + 1)$th slots in $\hat{c}_{\rho(i,j)}^{(model)}$ where $\tau(i,j) = (\mu(i,j) \bmod z)$ and $\rho(i,j) = \lfloor \mu(i,j)/z \rfloor$, $\mu(i,j) = (M * j + i)$, $z = \lfloor (M/(t+2)) \rfloor$ for all $(i,j) \in [0, M-1] \times [0, \lfloor N_{tot}/M \rfloor]$
10. $\hat{c}_i^{(model)} \leftarrow$ **ApproxLog**($\hat{c}_i^{(model)}$) for all $i \in [0, \lfloor \lfloor N_{tot}/M \rfloor \times M/z \rfloor]$
11. Copy $\hat{c}_{\rho(i,j)}^{(model)}[\tau(i,j) * (t+2)]$ into $c_j^{(model)}[i]$ if $j \times M + i < N_{t1} \times y_t$, for all $(j,i) \in [0, \lfloor N_{tot}/M \rfloor] \times [0, M-1]$.

FIGURE 3. The proposed PPNB training algorithm.

In the training protocol, it is assumed that the **CS** has received the encrypted BMVs for all values across all variables. To compute the conditional probability $Pr[X_i = j|Y = k]$, we incorporate α as the Laplace smoothing factor in both the numerator and $n_i \times \alpha$ in the denominator. This approach is necessary to address cases where the numerator is zero and to ensure that the probability does not exceed one even with the addition of α in the numerator. We set $\alpha = 0.01$. Detailed steps of the training procedure are

Input:
• $User u_\ell$ has an input vector $(x_0, x_1, \cdots, x_{d-1})$ where x_i is a categorical value of X_i $(i \in S_X)$.
• **CS** prepares for the model $\{c_i^{(model)}\}_{i\in[0,\lfloor N_{tot}/M\rfloor]}$ which is the result of training.
Output:
• $User u_\ell$ obtains a ciphertext $c^{(\mathbf{ret})}$ that can be decrypted using sk_{u_ℓ}, where $c^{(\mathbf{ret})}$ contains the inference result.
User u_ℓ:
1. One-hot encode x_i. Let $\vec{h}_{x_i}$ is the result of encoding $(i \in S_X)$.
2. $\vec{h}_{(x_0,\cdots,x_{d-1})} \leftarrow 1||\vec{h}_{x_0}||\cdots||\vec{h}_{x_{d-1}}$
3. $\vec{inp} \leftarrow$ (concatenation result of $(\vec{h}_{(x_0,\cdots,x_{d-1})})$ by y_t times) $(|\vec{inp}| = N_{tot})$
4. Encrypt $\vec{inp}$ with pk_s to obtain $\{c_w^{(inp)}\}$ where $w \in [0, \lfloor N_{tot}/M\rfloor](= S_w)$. Send $\{c_w^{(inp)}\}$ to **CS** with the ID of u_ℓ.
CS: $(N_{t1} = |\vec{h}_{(x_0,\cdots,x_{d-1})}|$, for every $w \in S_w$, perform step 1 and 2.)
1. $c_w^{(mult)} \leftarrow \mathbf{Mult}(c_w^{(model)}, c_w^{(inp)})$.
2. $c_w^{(out)} \leftarrow \mathbf{SumGroup}(c_w^{(mult)}, N_{t1}, \min(\lfloor M/N_{t1}\rfloor \times N_{t1}, (y_t - w) \times N_{t1}))$, assuming $\lfloor M/N_{t1}\rfloor \geq 1$
3. Compact the result of **SumGroup** $(\{c_w^{(out)}\}_{w\in S_w})$ in Step 2 into a single ciphertext where the first y_t slots have the result of compaction $(=c^{(\mathbf{ret})})$.
4. $c^{(\mathbf{ret})} \leftarrow \mathbf{FindMaxPos}(c^{(\mathbf{ret})})$
5. CS obtains $c^{(\mathbf{ret})}$ as a result of the inference.

FIGURE 4. The proposed PPNB inference algorithm.

depicted in the 'Training' figure above. In Step 9, the value from one slot is duplicated into $t + 2$ consecutive slots for use with the proposed ApproxLog(). In Step 11, the result of the logarithmic operation is moved back to the original slot position from where the input value was taken before Step 9.

D. INFERENCE

The proposed PPNB inference algorithm is detailed in Fig. 4. To illustrate the inference process clearly, we provide an execution example in Fig. 5, assuming the existence of three variables: X_0, X_1, Y. Each variable has 2, 3, and 2 categories, respectively. Initially, as depicted in Fig. 5, the CS possesses the model obtained from the training process, represented as $c_i^{(model)}$.

The algorithm does not specify how the result is delivered to the user requested for inference. Depending on if KM is trusted or semi-honest, the methods become different. Please refer to Section VIII-B and VIII-C for each case.

First, the input data required for inference is encrypted and sent to the CS after pre-processing. The pre-processing steps are illustrated in the fourth line of Fig. 5. For example, if the input values are $X_0 = 1$ and $X_1 = 2$, each value is one-hot encoded, resulting in $\vec{h}x_0 = (1, 0)$ and $\vec{h}x_1 = (0, 1, 0)$. These are combined into a vector $\vec{h}(x_0, x_1) = 1||hx_0||h_{x_1}$, forming the final input $\vec{inp} = (\vec{h}(x_0, x_1))^{y_t}$. This vector inp is then

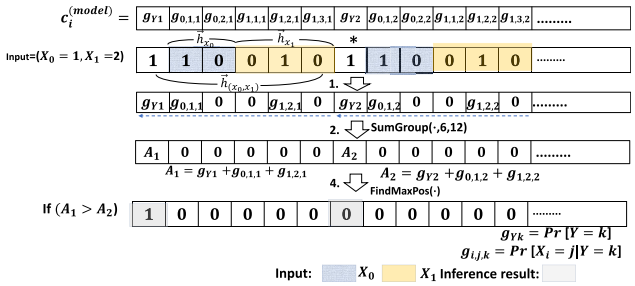

FIGURE 5. Inference example.

encrypted, and the ciphertext $[\![inp]\!]$ is sent to CS. In the fourth line of Fig. 5, the blue section represents $\vec{h}x_0$, and the yellow section represents $\vec{h}_{x_1}$.

After receiving $[\![inp]\!]$, CS multiplies it with $c_{model(i)}$ and performs the SumGroup() operation on the result, producing a ciphertext that contains A_1 and A_2, as shown in Step 2) of Fig. 5. Here, $A_1 = \log_e(Pr[Y = 1] \times Pr[X_0 = 1|Y = 1] \times Pr[X_1 = 2|Y = 1])$ and $A_2 = \log_e(Pr[Y = 2] \times Pr[X_0 = 1|Y = 2] \times Pr[X_1 = 2|Y = 2])$. In the final step, the FindMaxPos() function determines which of A_1 or A_2 is larger, placing a one in that position and zero in the other. The final result is then sent to the KM, which converts the ciphertext into a form that can be decrypted by the client and delivers it to the user. Note that if y_t is two, the ApproxSign() function suffices to identify the largest value's position. However, for $y_t > 2$, the more complex FindMaxPos() function is required.

VI. EXPERIMENTAL RESULTS

This section presents the experimental results of the proposed NB algorithms on various data sets. The experiment environment comprised an AMD RYZEN 5950X CPU, NVIDIA Quadro RTX A6000 48GB GPU, and 128GB RAM using ubuntu 20.04LTS. We performed timing measurements for operations, algorithms, and protocols across 30 trials.

A. CKKS AND SUBROUTINES

Table 3 presents performance metrics for CKKS unit operations and subroutines. Boot operations require only 152ms thanks to GPU acceleration [20]. Approximately 2s are consumed by ApproxSign(), used in both ApproxLog() and other subroutines. The relative error of ApproxInv() measures $9.052E-04 \pm 3.13E-06\%$. Errors for other unit operations are under $1E-06\%$.

We measured ApproxLog() averaging 2525±1.14ms. Accuracy of ApproxLog() was assessed by comparing results to traditional logarithm computations on plaintexts derived from decrypted ciphertexts. Both approaches show non-zero relative errors due to encryption noise.

Figure 6-(A) shows results for inputs at 2^{-t}, with t from 1 to 20. It indicates minimal relative error differences between our method and 'decrypt-then-log' for inputs $\leq 2^{-14}$, confirming robust performance near zero. Figure 6-(B) depicts relative errors for logarithm calculations across

TABLE 3. Average Time (ms) of CKKS operations and subroutines.

Add	**Mult**(lv. 9)	**Mult**(lv. 1)	**Rot**
0.045±0.0024	0.77±0.0055	0.40±0.0053	0.66±0.0086
Boot	**ApproxSign**	**ApproxInv**	**SumGroup**
152.0±4.85	2072±6.67	352.7±0.28	55.10±0.48

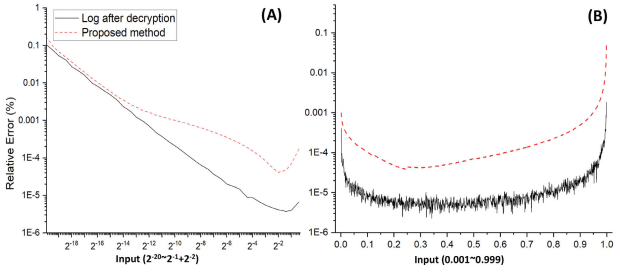

FIGURE 6. Accuracy of the proposed ApproxLog() comparing to 'decrypt-then-log' case.

intervals 0.001 to 0.999. Although about ten times larger than 'decrypt-then-log', errors remain < 0.01%, supporting high accuracy except near 1.0.

B. PERFORMANCE OF THE PROPOSED PPNB ALGORITHMS

In our experiments, we employed the following datasets: Acute Inflammation IUB (D1) and NRPO (D2) [31], Breast Cancer (D3) [18], and Car Evaluation (D4) [32]. D1 and D2 each contain six independent variables with categories consisting of 72, 2, 2, 2, 2, and 2, respectively. D3 includes nine independent variables, each with 10 categories, while D4 comprises six independent variables with categories numbered 4, 4, 5, 5, 3, and 3. The target variables in D1, D2, and D3 have two categories each, whereas D4 has four categories. All datasets have undergone preprocessing to convert each category into positive integers. The dataset sizes are 120 instances for D1 and D2, 699 instances for D3, and 1728 instances for D4.

Table 4 presents an analysis of the communication costs associated with our proposed protocol. During inference, if $\lceil y_t(N+1)\rceil \ll M$, multiple input data for inference can be encapsulated within a single ciphertext, facilitating parallel inference on multiple inputs. Moreover, given that $y_t \ll M$ typically, multiple inference results can often be accommodated within a single ciphertext. Regarding training, we observe that the amount of data transmitted to the **CS** scales with the number of rows in the dataset (l_{data}). Additionally, the sum of categories for variables ($N+y_t$) also influences the communication costs.

1) ACCURACY ANALYSIS

Table 5 presents a comparison of accuracy between the proposed PPNB method and alternative approaches. Our method demonstrates superior performance compared to others, particularly evident in D3. For D4, the accuracy achieved by our method closely approaches that of scikit-learn.

TABLE 4. Communication cost.

	User → CS	**CS → KM** (**KM**→u_ℓ)
Inference	$\lceil y_t(N+1)/M\rceil \times s_{\mathbf{ctxt}}$	$\lceil y_t/M\rceil \times s_{\mathbf{ctxt}}+s_{\mathbf{ID}}$
Training	$(N+y_t)\lceil l_{data}/M\rceil \times s_{\mathbf{ctxt}}$	N/A

TABLE 5. Accuracy (scikit:scikit-learn).

	Proposed	[16]	[27]	[17]	scikit
D1	100%	100%	-	-	100%
D2	100%	100%	-	-	100%
D3	97.8%	97.42%	95%	96%	97.8%
D4	74.9%	-	-	-	75.2%

TABLE 6. Average time for training and inference.

Training (seconds)			
D1	**D2**	**D3**	**D4**
12.73±0.87	12.87±0.06	14.20±0.11	9.97±0.11
Inference (milliseconds)			
D1	**D2**	**D3**	**D4**
835 ±0.7	835 ±0.3	838 ±0.2	3180 ±4.0

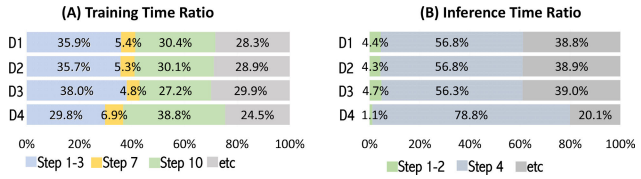

FIGURE 7. Training and inference time ratio.

2) EXECUTION TIME ANALYSIS

Table 6 displays the elapsed times for training and inference across datasets D1 to D4. During inference, D3 requires the most time due to its larger N compared to the other datasets, despite all datasets having sizes smaller than M. On the other hand, D4 performs the least effectively during inference due to the intensive use of FindMaxPos() with y_t set to four. Figure 7-(A) illustrates the distribution of execution times for training steps. Key operations such as calculating the distributions of joint variables X_i and Y (Steps 1 to 3), performing the inversion (Step 7), and executing ApproxLog() (Step 10) collectively account for about 70.1% to 75.5% of the overall execution time. The remaining time is spent on placing the computed values into their designated slots. Figure 7-(B) illustrates the distribution of execution time for each step during the inference phase, with Step 4 (FindMaxPos()) taking up the majority of the time. However, removing this step is not feasible as it relates to concealing conditional probabilities essential for maintaining model privacy.

3) SOURCE CODE

The source code working on top of the HEAAN SDK running on CPU is available. The HEAAN library can be obtained from https://heaan.it. Please visit https://gitlab.com/islab-seoultech/homomorphic-encryption-ml/naive-bayes-classifier

to access the code. We are not allowed to release the GPU implementation version of the proposed method because the GPU version of HEAAN library is proprietary to CryptoLab.

VII. COMPUTATION COMPLEXITY ANALYSIS OF THE PROPOSED METHOD AND [17]

This section presents a complexity analysis comparing the proposed method with the approach described in [17], highlighting the performance improvements of the proposed method in terms of computational complexity.

A. ANALYSIS OF THE PROPOSED METHOD

Table 7 presents a detailed analysis of the computational complexity associated with the proposed PPNB training and inference protocols, along with the relevant algorithms. The final column specifies the multiplication depth used.

During training, Mult and Rot operations are performed in proportion to N_{tot}, which is $\sum_{i \in X} n_i \times y_t$, typically a very large value. In contrast, inference and other algorithms require Mult or Rot operations proportional to N_{tot}/M. Because M is generally large in conventional settings, this difference leads to varying performance between training and other protocols and algorithms.

Moreover, the FindMaxPos() algorithm employs $(log_2 y_t + 1)$ApproxSign() to determine the computation needed for the FindMax() function, which significantly impacts performance when y_t exceeds 2.

The exact computational demands and multiplication depth for ApproxSign() and ApproxInv() are undisclosed. However, based on existing literature [19], [24], it can be inferred that these methods likely use polynomials of high degrees. Consequently, the depth required is approximately the logarithm of the polynomial's degree, and the number of multiplications is estimated to be in the tens.

To sum up, the disparity in operations between training and other processes, especially due to the high N_{tot} value and the polynomial approximations in ApproxSign() and ApproxInv(), leads to notable performance variations.

B. COMPLEXITY ANALYSIS OF [17]

Based on the analysis from Table 8, we have conducted a detailed assessment of the homomorphic encryption (HE) operations required for training and inference in the method proposed by [17]. To ensure a fair comparison, we assume a single target variable ($\beta = 1$ in [17]) and $y_t = 2$. The outcomes of this evaluation are documented in Table 9 (training) and Table 10 (inference).

Upon reviewing the training phase, it becomes apparent that the method proposed in [17] necessitates a significantly deeper level of multiplicative operations compared to our proposed approach, as delineated in Table 7 and Table 9. In our method, the multiplicative depth remains constant because operations like ApproxInv() and ApproxLog() are independent of variables such as dataset size and the number of variables. Conversely, the training depth in [17] scales as $O(l_{data} \cdot n_{max})$, indicating potential extensive use of bootstrapping operations proportional to dataset size and the number of independent variable categories.

TABLE 7. Computation complexity analysis ($n_{ctxt} := \lceil l_{data}/M \rceil$, d(A): required multiplicative depth to run algorithm A, cMult: Mult where the multiplier is not a ciphertext.).

Operations	Complexity	Mult. Depth
Training		
Mult	$N_{tot} \cdot n_{ctxt} + \lceil N_{tot}/M \rceil$	d(**ApproxInv**()) +d(**ApproxLog**()) +6
cMult	$2*N_{tot} + 4\lceil N_{tot}/M \rceil$	
Rot	$2log_2 M + 2N_{tot} + \lceil log_2(t+1) \rceil$	
Add	$N_{tot}(n_{ctxt} - 1)$	
ApproxInv()	$\lceil N_{tot}/M \rceil$	
Etc.	$\lceil N_{tot}/M \rceil$ **ApproxLog()**	
Inference		
Mult	$\lceil N_{tot}/M \rceil$	d(**FindMaxPos**()) +2
cMult	y_t	
Rot	$\lceil N_{tot}/M \rceil \ log_2 N_{t1} + y_t$	
Add	0	
ApproxInv()	0	
Etc.	1×**FindMaxPos**$(\cdot, y_t)$	
FindMaxPos$(\cdot, ns)$		
Mult	1	d(**ApproxSign()**) +d(**FindMax**()) +3+$log_2 ns$
cMult	$(2 + log_2 ns)$	
Rot	$(2 + log_2 ns)$	
Add	3	
ApproxInv()	0	
Etc.	$1 \times$ **FindMax**$(\cdot, ns) + 1 \times$**ApproxSign()**	
FindMax$(\cdot, ns)$		
Mult	$5log_2 ns$	$(log_2 ns)\times$ (d(**ApproxSign**()) +3)
cMult	0	
Rot	$log_2 ns$	
Add	$3log_2 ns$	
ApproxInv()	0	
Etc.	$log_2 ns$ of **ApproxSign()**	
ApproxLog()		
Mult	13	d(**ApproxSign**()) +d(**ApproxInv**()) +1
cMult	11	
Rot	1	
Add	3	
ApproxInv()	1	
Etc.	1×**ApproxSign()**	

Regarding Mult and cMult, comparing the required quantities, our proposed method requires $O(N_{tot} \cdot n_{ctxt}) = O(dn_{max} l_{data}/M)$, whereas [17] needs $O(l_{data}(n_{max} d + \mu^2))$. Our method benefits from the factor $1/M$, as it fully utilizes SIMD. Additionally, since [17] stores only one bit value per ciphertext slot, it requires μ slots to represent a single integer, resulting in an additional $O(l_{data}\mu^2)$ computation. In contrast, our method can store one number per slot, eliminating this additional computation.

It's important to note that, as mentioned in [17], the process of calculating probabilities and conditional probabilities using the distribution of each variable value is performed after decryption. Therefore, no homomorphic operations are needed for this process in [17]. Conversely, our method performs this process in an encrypted state and additionally performs the log calculation homomorphically with encrypted probability values. Hence, Table 7 includes the necessary HE operations for these processes. In conclusion, our method achieves more functions in an encrypted state with significantly fewer computations compared to [17] during training.

The inference performance of our proposed method significantly outperforms that of [17], primarily due to the

TABLE 8. Computation complexity analysis of the various basic operations in [17] (μ: bit precision of a plaintext value).

Operations	HE operations				Mult. Depth × Width	Etc.
	Add	**Rot**	**Mult**	**cMult**		
l.equ	2	$\lfloor \log_2 \mu \rfloor$+1	$\lfloor \log_2 \mu \rfloor$	1	$\lfloor \log_2 \mu \rfloor \times 1$	
l.cmp	$\mu + \lfloor \log_2 \mu \rfloor$	$\mu + \lfloor \log_2 \mu \rfloor$	μ	$\lfloor \log_2 \mu \rfloor$+1	$(2\mu\text{-}1)\times 1$	
Scpy	$k-1$	$k-1$	0	0	0	k: # of slots copied
l.mul	$2\mu - 2$	$2\mu - 2$	μ	μ	1×1	
l.add	$\mu^2 + 2\mu + 2$	$\frac{\mu^2+3\mu-1}{2}$	$2\mu - 1$	$\frac{\mu^2+\mu}{2}$	$(\mu+2)\times O(\mu^2)$	
CPack	m	0	0	m	1×1	$GF(2^m)$ is used
CUPack	m^2	0	0	m^2	$1 \times d$	
SPack	$\lfloor M/\mu \rfloor$ -1	$\lfloor M/\mu \rfloor$ -1	0	0	0	
SUPack	0	$\lfloor M/\mu \rfloor$ -1	0	$\lfloor M/\mu \rfloor$	$1\times \lfloor M/\mu \rfloor$	

TABLE 9. Computation complexity analysis of training (Max: max function, $n_{max} = \mathsf{Max}(n_0, \cdots, n_{d-1})$) in [17].

HE operations	Required numbers
Add	$l_{data}(2n_{max} + 3\mu^2 + 6\mu + 6)$
Rot	$l_{data}(n_{max}(3d - 2 + \lfloor \log_2 \mu \rfloor) + \frac{3\mu^2+9\mu-3}{2}) + 2n_{max}(d-1)$
cMult	$l_{data}(n_{max}(3d+1) + \frac{3\mu^2+9\mu+2}{2}) + 2n_{max}d$
Mult	$l_{data}(n_{max}\lfloor \log_2 \mu \rfloor + 6\mu - 3)$
Mult. Depth	$(l_{data}(1 + 3n_{max} + 3\mu + 6) + 2n_{max} + \lfloor \log_2 \mu \rfloor) \times \mathsf{Max}(O(\mu^2), d, l_{data} \cdot n_{max})$

TABLE 10. Computation complexity analysis of inference in [17].

HE operations	Required numbers
Add	$n_{max}(d + 2\mu^2 + 5\mu + 1) + 16\mu + \lfloor \log_2 \mu \rfloor - 14$
Rot	$n_{max}(d + \lfloor \log_2 \mu \rfloor + \frac{\mu^2+5\mu-5}{2}) + 2d + \lfloor \log_2 \mu \rfloor + 14\mu - 16$
cMult	$n_{max}(\mu^2 + 2\mu) + 2\lfloor M/\mu \rfloor + 2\mu + \lfloor \log_2 \mu \rfloor + 1$
Mult	$n_{max}(2\mu + \lfloor \log_2 \mu \rfloor - 2) + 2\mu$
Mult. Depth	$(2n_{max}(\mu+2) + 6\mu + \lfloor \log_2 \mu \rfloor - 4) \times \mathsf{Max}(O(\mu^2), n_{max}, \lfloor M/\mu \rfloor)$

lower multiplicative depth. In [17], the multiplicative depth is $O(n_{max}\mu)$, which can reach several thousand when μ is 40, as in our method. This necessitates numerous bootstrapping operations, greatly increasing inference time. Conversely, in our method, if y_t is set to 2 as in [17], the FindMaxPos() operation at the end of the inference can be replaced by a single multiplication, resulting in a multiplicative depth of 3. This allows inference to be performed without bootstrapping.

In terms of the number of multiplication operations, our method requires $N_{tot}/M + 1 (\leq dn_{max}/M + 1)$, whereas [17] needs $O(n_{max} \cdot \mu^2)$. If d (the number of independent variables) is small, our method benefits from the $1/M$ factor, requiring fewer operations. However, if d is very large such that d/M exceeds μ^2, our method may require more computation. Fortunately, since our method uses $M = 32768$, it remains more efficient unless the number of independent variables is exceptionally large, in the hundreds of thousands.

In summary, our method offers significant improvements in inference performance due to its lower multiplicative depth and more efficient computation, making it superior to [17] under typical conditions.

VIII. DISCUSSION AND EXTENSION

A. NOISE ANALYSIS OF THE PROPOSED HOMOMORPHIC LOGARITHM ALGORITHM

In this subsection, we examine the noise generated by the ApproxLog() algorithm, which is a key factor influencing the accuracy of the algorithm's results. The aim is to validate the correctness of the algorithm's outcomes. The noise in ApproxLog() originates from two primary sources. Firstly, there is noise resulting from the homomorphic operations. Secondly, there is the approximation error that arises from ApproxLog() using a polynomial function to approximate the logarithmic operation. Our analysis begins with an exploration of the noise stemming from homomorphic operations. Following this, we evaluate the overall error by integrating the approximation error with the error from homomorphic operations.

1) THE ANALYSIS OF THE NOISE BY HOMOMORPHIC OPERATIONS WITHOUT CONSIDERING BOOTSTRAPPING

According to [33], if the noise generated by decrypting the ciphertext c_1 is called ϵ_1 and c_2 is called ϵ_2, the noise generated by each operation is described as follows.

- $\mathsf{Mult}(c_1, c_2) \rightarrow \epsilon_1 m_2 + \epsilon_2 m_1 + \epsilon_{rs} \leq \epsilon_1 + \epsilon_2 + \epsilon_{rs}$ ($\because m_1, m_2 \leq 1$)
- $\mathsf{Add}(c_1, c_2)$ (or $\mathsf{Sub}(c_1, c_2)$) $\rightarrow \epsilon_1 + \epsilon_2$
- $\mathsf{Rot}(c_1, r) \rightarrow \epsilon_1 + \epsilon_{ks}$
- $\mathsf{Boot}(c_1) \rightarrow \epsilon_1 + \epsilon_{\mathsf{boot}}$

where m_1 and m_2 are the plaintext used to create c_1 and c_2 by encryption, respectively. Since we suppose the plaintext values are probabilities, m_1,$m_2 \in [0, 1]$. ϵ_{rs} is the noise by the rescaling operation and ϵ_{ks} is the noise by key-switching.

Fig. 8-(A), (B), and (C) show the circuits of the proposed ApproxLog(), HermesLog(), and SumGroup() function, respectively. We suppose 'Square' operation is the same as Mult where two identical ciphertexts are provided as inputs. Also, 'Inverse' refers to the ApproxInverse() algorithm. The noise from this function is described as ϵ_{inv}. Note that in Fig. 8-(C), the circuits in the area identified by the blue solid line is repeated $\lceil log_2 gs \rceil$ times. Then, the result $[\![ret]\!]$ is obtained.

We calculate the error introduced by each circuit in Fig. 8-(A), (B), and (C) based on the noise of each operation presented above.

Let x be the noise in the input $[\![input]\!]$ of each algorithm. If we exclude the noise added by bootstrapping, the noise $\epsilon_{HM}(x)$ from circuit (B) (HermesLog()), which is a function of x, can be calculated as $47x+22\epsilon_{inv}+46\epsilon_{rs}$. We can also compute that the noise $\epsilon_{\mathsf{SumGroup}}(x)$ generated by circuit (C) (SumGroup) is $2^{\lceil log_2 gs\rceil}(x)+(2^{\lceil log_2 gs\rceil}-1)(2\epsilon_{rs}+\epsilon_{ks})$. This is a result of considering that the area surrounded by the blue line in (C) is repeated by $\lceil log_2 gs\rceil$ times.

The noise of the proposed ApproxLog() ($\epsilon_{HL}(x)$), can be calculated based on $\epsilon_{HM}(x)$ and $\epsilon_{\mathsf{SumGroup}}(x)$, where x is the noise of $[\![input]\!]$ of the corresponding circuit. We can see that if we can evaluate $\epsilon_{\mathsf{SumGroup}}(x)$ with the correct input that is mapped to the noise of the input of the SumGroup circuit and calculate the noise added during performing SumGroup in (A), we can finally calculate the ϵ_{HL} (ϵ_{inp}), where ϵ_{inp} is the noise of $[\![input]\!]$ in (A).

From the circuit description of (A), we can calculate the noise of the input for the HM circuit in Fig. 8-(A) (β_{HM}) as $4\epsilon_{inp}+2\epsilon_{ks}+4\epsilon_{rs}$.

However, to calculate the noise of the input of the HM circuit precisely, we need to add the noise caused by bootstrapping. This will be addressed in the next part.

The noise of the input of SumGroup ($\beta_{SumGroup}$) in (A) is $8\epsilon_{inp}+4\epsilon_{ks}+8_{rs}+\epsilon_{HM}(\beta_{HM})$. This also does not take the noise by bootstrapping into account. We consider the noise by bootstrapping in the next part.

2) THE ANALYSIS OF THE NOISE BY BOOTSTRAPPING

The bootstrapping algorithm is composed of a specific, unvarying quantity of homomorphic operations. As a result, the noise generated by the bootstrapping process can be characterized as a constant, denoted as ϵ_{boot}. This noise encompasses two main components: firstly, the noise arising from the homomorphic operations themselves, and secondly, the approximation error that is inherent to the CKKS-bootstrapping operation. This approximation error is due to the process of approximating the plaintext value in the input ciphertext of the bootstrapping algorithm, and then incorporating this approximated plaintext into the output ciphertext.

We model where the bootstrapping operation takes place in a circuit with a binary variable. If a multiplication or a squaring operation needs to be evaluated in a circuit but it is difficult to proceed without bootstrapping, we assume the bootstrapping operation is inevitably performed. In this case, the noise caused by bootstrapping is also added in the input of the multiplication or square, thus the noise of the result of the operation is also affected. In Fig. 8-(A)∼(C), each multiplication or squaring operation is numbered with a parenthesized number. Note that bootstrapping can occur before any of these operations. However, once bootstrapping occurs, it will not occur again until all available multiplication depth is consumed. To reflect this situation, we define a binary variable $b_{(i)}(\in \{0,1\})$ where $i \in [1,19]$ as a value set to 1 if bootstrapping occurs before the operation denoted by (i), and to 0 if it does not. We then can denote the noise generated with respect to bootstrapping before the operation (i) as $b_{(i)}\epsilon_{\mathsf{boot}}$. With this, we can compute the noise introduced by bootstrapping in the circuits (A), (B), and (C) in the Fig. 8. Note that in (C), operations (18) and (19) are repeated $\lceil log_2 gs\rceil$ times, so we use $b_{(18),j}$ to represent if bootstrapping occurs in the j-th iteration before the circuit (18). Note that $b_{(19),j}=b_{(18),j}$ as they share the same input.

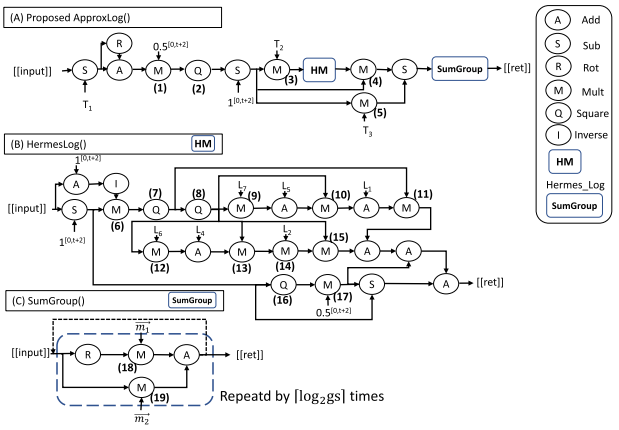

FIGURE 8. Circuit description of the proposed ApproxLog(), HermesLog(), and SumGroup() algorithm.

The final aspect to examine is the analysis of the error induced by the 'Inverse' operation in circuit (B), specifically related to bootstrapping. The Inverse operation is significant because it utilizes multiple multiplicative depths internally. Understanding the exact number of multiplicative depths consumed before bootstrapping becomes necessary is crucial for accurate analysis. To address this, we define the noise value associated with the Inverse operation as $\rho_{\mathsf{inv},k}$, where k represents the number of multiplication depths completed prior to the occurrence of bootstrapping. This definition allows for a precise quantification of the noise generated as a result of bootstrapping at various stages of the Inverse operation's execution within circuit (B).

Based on the definitions so far, the noise due to bootstrapping for each circuit can be calculated as follows. First, we calculate the noise (ρ_{SumGroup}) generated by circuit (C). To do this, we need to check the level of the ciphertext to see if the multiplications (18) and (19) are possible before the rotation operation. We can denote it as $b_{(18),i}$ whether bootstrapping is required at that position at the ith iteration, so we have $\rho_{\mathsf{SumGroup}}=\Sigma_{i=1}^{\lceil log_2 gs\rceil}2^{\lceil log_2 gs\rceil+1-i}*b_{((18),i)}\epsilon_{\mathsf{boot}}$.

We calculate the noise ρ_{HM} caused by bootstrapping on Hermes_Log. From the circuit (B), it can be seen that the result of the operation in (8) is used in many different places. Therefore, we can denote as $b_{(9)}$ whether bootstrapping is necessary for further operations after (8). From the structure of the circuit, $b_{(9)}=b_{(12)}$. Also, since operations (13) and (15) require the same inputs, we can use $b_{(9)}$ to denote whether a bootstrapping is required. Taking these into account, we can derive that $\rho_{HM}=48b_{(6)}\epsilon_{\mathsf{boot}}+22(\rho_{\mathsf{inv},k}+b_{(7)}\epsilon_{\mathsf{boot}})+(5b_{(8)}+4b_{(9)}+\Sigma_{i=10}^{1}5(b_{(i)})+2*b_{(17)})\epsilon_{\mathsf{boot}}$.

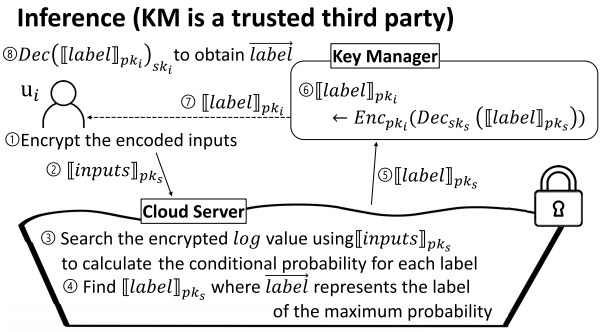

FIGURE 9. System Model if KM is a trusted party.

Finally, we compute the noise due to bootstrapping that occurs within the proposed homomorphic logarithmic algorithm. Our interest is the amount of the noise (γ_{HM}) added to the input of the HM circuit and that ($\gamma_{\mathsf{SumGroup}}$) to the input of the SumGroup() in (A), respectively, as they are needed to calculate the noise of the proposed ApproxLog(). It can be seen that $\gamma_{HM} = 2(b_{(1)} + b_{(2)}) + b_{(3)}$ if we carefully analyze (A). Note that, since the operation (2) is a squaring, the noise is doubled. Finally, we calculate $\gamma_{\mathsf{SumGroup}}$. Because the operations (3), (4), and (5) have the same input as one of their operands, the noise due to bootstrapping caused by that input can be denoted as $b_{(3)}\epsilon_{\mathsf{boot}}$. Then, we can derive $\gamma_{\mathsf{SumGroup}} = \rho_{HM} + (4(b_{(1)} + b_{(2)}) + 2b_{(3)})\epsilon_{\mathsf{boot}}$.

3) ANALYSIS RESULT

The final noise calculation, which combines the contents of the two subsections above, is as follows: From the circuit (A), we can derive $\epsilon_{HL}(\epsilon_{inp}) = \epsilon_{\mathsf{SumGroup}}(\beta_{\mathsf{SumGroup}} + \gamma_{\mathsf{SumGroup}}) + \rho_{\mathsf{SumGroup}}$. The previous part showed that $\beta_{\mathsf{SumGroup}} = 8\epsilon_{inp} + 4\epsilon_{ks} + 8\epsilon_{rs} + \epsilon_{HM}(\beta_{HM} + \gamma_{HM})$, $\gamma_{\mathsf{SumGroup}} = \rho_{HM} + (4(b_{(1)} + b_{(2)} + 2b_{(3)})\epsilon_{\mathsf{boot}}$, $\beta_{HM} = 4\epsilon_{inp} + 2\epsilon_{ks} + 4\epsilon_{rs}$ and finally $\rho_{\mathsf{SumGroup}} = \Sigma_{i=1}^{\lceil log_2 gs \rceil} 2^{\lceil log_2 gs \rceil + 1 - i} * b_{(18),i}$. Using them, we can compute that $\epsilon_{HL}(\epsilon_{inp}) = 2^{\lceil log_2 gs \rceil}(196\epsilon_{inp} + 97\epsilon_{ks} + 244\epsilon_{rs} + 22\epsilon_{inv} + (98b_{(1)} + 98b_{(2)} + 49b_{(3)} + 48b_{(6)} + 22(f_{inv,k} + b_{(7)}) + 5b_{(8)} + 4b_{(7)} + \Sigma_{i=10}^{1} 5b_{(i)} + 2b_{(17)})\epsilon_{boot}) - 2\epsilon_{rs} - 2\epsilon_{ks} + \Sigma_{i=1}^{\lceil log_2 gs \rceil}(2^{(\lceil log_2 gs \rceil + 1 - i)} b_{(18),i}\epsilon_{boot}$.

Through experimentation, we have found that $\epsilon_{inp} \sim 10^{-9}$, $\epsilon_{rs} \sim 10^{-8}$, $\epsilon_{ks} \sim 10^{-7}$, $\epsilon_{boot} \sim 3 \times 10^{-8}$, $\epsilon_{inv} + \rho_{\mathsf{inv},k} \sim 2.5 * 10^{-7}$, and as a result, the magnitude of the noise generated by the analysis can be found to be around 10^{-4} considering $Pr[b_{(i)} = 1] = 1/9$ for $i \in [1, 19]$, as 1 bootstrapping is required for every 9 multiplicative depth consumption. This is larger than the actual size of the noise. The reason for this is the inequality in the calculation of multiplicative noise. If the size of the actual plaintext is less than 1, the magnitude of the noise generated by multiplication will be smaller than the value used in our calculation.

The error we have not considered is the approximation error. The proposed method approximates the logarithmic function as a polynomial function using the Hermes method. This error should also be considered. [30] shows that the maximum error is about 2^{-58}, which is less than 10^{-16}. This results in a very small error compared to the error produced by homomorphic operations.

Theorem 1: The noise of the output of the proposed logarithm algorithm is at most 10^{-4} on the parameter used in the paper.

Proof: The theorem is concluded from the previous analysis, with its inequality rooted in the presumed inequality of noise generated by Mult. ■

B. AN INSTANCE OF SYSTEM MODEL WHEN KM IS A TRUSTED PARTY

We introduce a plausible instance of the system model where the proposed PPNB can be utilized, which was suggested in [26]. There are three protocols: key distribution, training and inference. Initially, in the key distribution protocol, users generate public key/private key pairs and sent the public keys to the key manager (KM) which is a trusted party. KM generates a public key/private key pair and evaluation keys. KM's public key is distributed to all the other players, and the evaluation keys are delivered to the cloud server (CS).

In the training protocol, the users, who act as data owners in Fig. 1, deliver the encrypted data with some parameters such as the variable names, the number of categories in each variable, and the size of data, to CS. The data is encrypted by the system public key. CS performs training with the received ciphertexts and the parameters. The training is solely performed by CS, without the help of the other players. Therefore, CS can calculate the model which is the encryption of the logarithm of probabilities and conditional probabilities of variables in the data. This is depicted in Fig. 1-a).

The inference process is as follows. A user (u_i), who also act as a client, first creates a query that is an encrypted vector of the input variables, then delivers the query to CS along with her ID, CS performs the inference and sends the result to KM. KM decrypts the received ciphertext using sk_s then re-encrypt the result using the user's public key after checking if the inference result violates the privacy policy. The re-encrypted inference result is sent to the user. In the case of a secure outsourcing scenario, the system public key is eventually the public key of the Data Owner, so decryption is possible because the Data Owner, which is equal to Client in the setting, has a corresponding private key. This process is depicted in Fig. 9.

Please be aware that this system model assumes the data owners and the clients are the same set of entities. For example, a small number of companies want to share their data to create models with which each company can perform the desired inference through CS, and the companies perform inference on their desired inputs through CS to receive the inference results. Because multiple companies' data are used to generate models, an individual company cannot access the generated models thus can only perform inferences with them

through CS. In this scenario, the companies can act as both the data owner and the client running inferences.

Another thing we would like to mention in this model is the role of KM. As mentioned in [26], it could be a legal requirement in certain countries that the inference results must be verified if the inference results contain any privacy-sensitive information related to the people who are related to the training data. This verification must be done by a trusted government agency. In a privacy preserving service utilizing homomorphic encryption, one of the ways to satisfy this legal requirement is for a trusted entity to decrypt the inference result, verify its contents, and deliver it to the client who requested inference. Therefore, in this case, it is inevitable that a third party entity such as KM decrypts the inference result.

C. THE INFERENCE PROTOCOL WHEN KM IS SEMI-HONEST

We discuss the inference protocol if KM is semi-honest. We remind the inference protocol when KM is a trusted third party, as depicted in Fig. 10-(A) in the previous section. In the protocol, the client encrypts her input with the system public key and sends it to CS with his ID. Then, CS performs the inference protocol described in Fig. 10-(A) (details given in Fig. 4 of Section V-D) and sends the result to KM with the client's ID (ID_u). KM decrypts the result to obtain the classification result ($\vec{ret}$), re-encrypts it with the client's public key, and sends it to the client. The client decrypts the result to obtain the classification result.

Assuming KM is semi-honest, we can design an inference protocol as shown in Fig. 10-(B). The difference from (A) is that CS encrypts an element $\vec{r}$ selected from a uniform distribution in the plaintext space ($\mathbb{Z}_{2^\Delta}[X]/(X^N+1)$) with the system public key, add it to the ciphertext of the inference result then sends the addition result to KM, and simultaneously encrypts $\vec{r}$ with the client's public key and sends it to the client. KM receives the ciphertext that contains the inference result plus $\vec{r}$. It decrypts the ciphertext and re-encrypts the decryption result with the client's public key. Finally, it sends the re-encrypted ciphertext to the client. The client decrypts all the received ciphertexts and subtracts the decryption result received from the KM from that received from the CS. The result of the subtraction is the final inference result.

1) SECURITY ANALYSIS

We discuss the privacy of the proposed inference protocol of semi-honest setting. We show that in the protocol in Fig. 10-(B), it should be difficult for KM to obtain any information about the classification results and the model used from the information it receives. To this end, we define the following experiment in Alg. 7, where *Adv* refers to an adversary.

In the experiment, TrainingOracle is defined in Alg. 8 as an oracle that generates the model that is the result of the training, to be ready to execute InferenceOracle, which is defined in Alg. 9. To execute TrainingOracle, *Adv* provides it with the training data needed to generate the model. TrainingOracle runs the training algorithm and returns the generated model. Later, *Adv* can call InferenceOracle to obtain information given to KM when the proposed inference protocol runs. The goal of the experiment is to check if *Adv* can distinguish whether InferenceOracle operates in the World-0 setting or in the World-1 setting, given in Fig. 10. *Adv* runs its algorithm arbitrarily and returns his guess as b'. The final part of the experiment is to compare it to the actual value of b and return 1 if the guess is correct and 0 otherwise.

Algorithm 7 Experiment: $\mathbf{Exp}(1^k, \mathsf{HE})$

1: **procedure** $\mathbf{Exp}(1^k, \mathsf{HE})$
2: $sk, pk, evk \leftarrow \mathsf{HE.KeyGen}(1^k)$
3: $\mathsf{state}, \{c_i^{(model)}\}_{i\in[0,\lfloor N_{tot}/M\rfloor]} \leftarrow Adv^{\mathsf{TrainingOracle}(\cdot,pk,evk)}$
4: $b \leftarrow^{\$} \{0,1\}$
5: $b' \leftarrow Adv^{\mathsf{InferenceOracle}_{sk}(\cdot,\mathsf{state},\{c_i^{(model)}\}_{i\in[0,\lfloor N_{tot}/M\rfloor]},pk,evk)}$
6: **return** 1 if $b = b'$ else 0

Based on the experiment, we define the privacy of the proposed inference protocol with semi-honest KM as follows.

Definition 1: (KM-privacy) We say that an inference protocol supports KM-privacy if $Pr[\mathbf{Exp}(1^k, \mathsf{HE}) \rightarrow 1] < 1/2 + \epsilon$ (where ϵ is negligible over security parameter k) when performing the experiment of Alg. 7, i.e., KM cannot computationally distinguish between model information and random information using the information it has obtained.

The above definition can be justified by the difference between the World-0 and World-1 in Fig. 10. In World-1, KM acquires information that is completely unrelated to the model and inference inputs. The indistinguishability between World-0 and World-1 by the adversary means that KM cannot extract any meaningful information from what it obtains even it runs the real inference protocol in World-0.

Algorithm 8 TrainingOracle$(data, pk, evk)$

1: **procedure:** TrainingOracle$data, pk, evk$
2: Preprocess *data* to make BMVs and encrypt them with pk.
3: Run the training protocol with the encrypted input in the previous step and pk, evk.
4: Obtain $\{c_i^{(model)}\}_{i\in[0,\lfloor N_{tot}/M\rfloor]}$ as an output of the training protocol.
5: Set state to any information needed to run the inference protocol.
6: **return** state, $\{c_i^{(model)}\}_{i\in[0,\lfloor N_{tot}/M\rfloor]}$

Based on these definitions, we prove the following theorem.

Theorem 2: The inference protocol for the semi-honest KM described in Fig. 10-(B) satisfies KM-privacy.

Proof: In Fig. 10, the only difference between World-0 (B) and World-1 (C) is Step ③-4), i.e., whether the ciphertext c'_{ret} delivered to the KM is simply the ciphertext of a random vector or a random vector plus the inference result. Therefore, in both worlds, the result obtained by KM from decrypting c'_{ret} is eventually a uniformly randomly selected value from the possible plaintext space, and the probability of the decryption

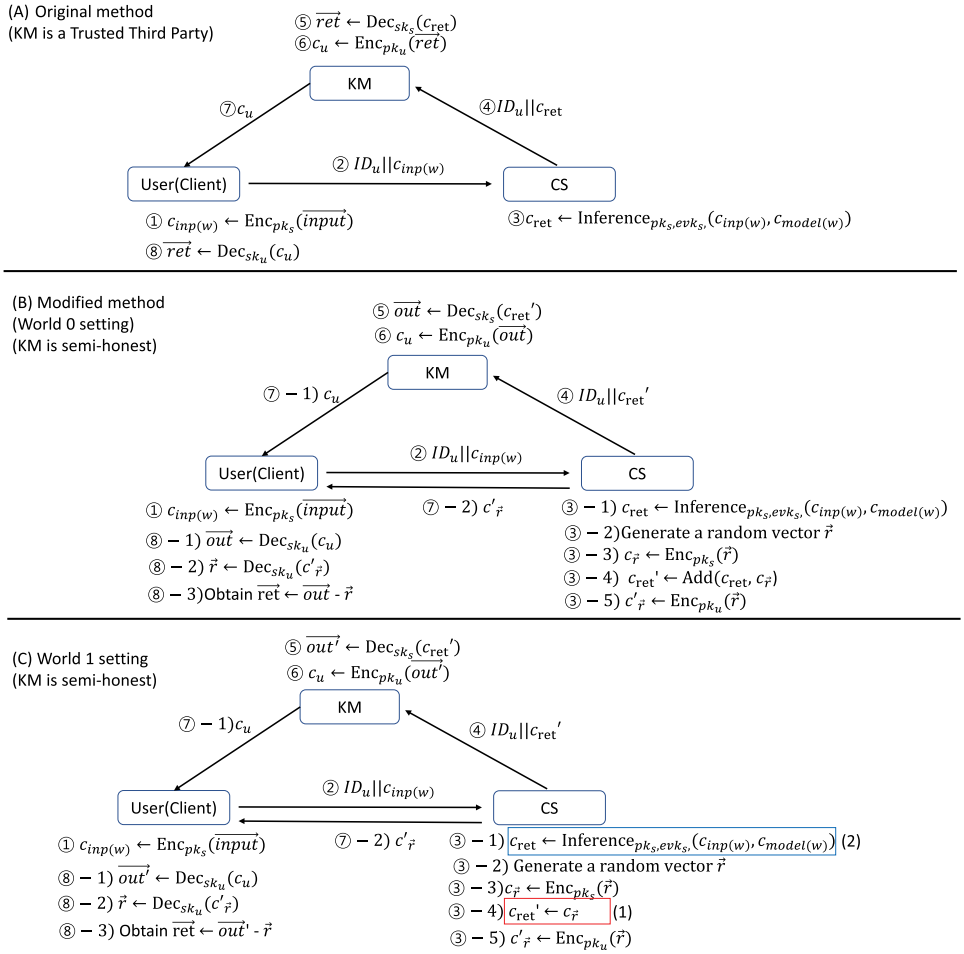

FIGURE 10. The original protocol and the World-0 and World-1 settings for the security proof of the inference protocol in the semi-honest setting ($\mathrm{Inference}_{pk_s,evk_s}(c_{inp(w)}, c_{model(w)})$ refers to the inference algorithm performed by CS. $c_{inp(w)}$ contains the values of the independent variables as an input for inference and $c_{model(w)}$ is the set of ciphertexts that contain the model as a result of training. ID_u is the ID of the user. $\overrightarrow{ret}$ is the inference result.)

result being a particular value is the same in both worlds. Therefore, it is statistically indistinguishable to tell whether KM is in World-0 or World-1 based on the $\overrightarrow{out}$ value alone.

However, if the ciphertext $c'_{\vec{r}}$ in Step ⑦-2) is decrypted, the attacker can distinguish his world, so the adversary is computationally indistinguishable. ■

More detailed description on the inference protocol with semi-honest KM is provided in Fig. 11.

IX. RELATED WORK

A. PRIVACY PRESERVING NAIVE BAYES ALGORITHMS

Initial research into Privacy-Preserving Naive Bayes (PPNB) algorithms, commencing in the early 2000s, was geared towards developing models that utilize user data for training while ensuring the privacy of each participant's data remains intact. This research, referenced in various studies [34], [35], [36], [45], [46], [47], [48], has continued up to the recent past.

The primary outcome of this research is a trained model that either gets shared with all participants or is stored on a central server in an unencrypted (clear-text) format. However, a notable limitation of this early-stage research is its lack of consideration for access control measures for the resulting model. This oversight means that the model cannot be exclusively distributed to entities outside the circle of participants who contributed to its creation. Furthermore, it does not allow for classification tasks to be conducted without revealing the model to those participants involved in its development. Such limitations highlight the need for further advancements in PPNB research, particularly in enhancing the privacy and control aspects of the generated models.

The second category PPNB methods involves introducing a certain level of perturbation to both data and models. This approach, designed to obscure the original data and model details used in training and classification, is documented in various studies [49], [50], [51], [52]. The primary goal of these methods is to achieve differential privacy.

However, as highlighted in [35], a significant drawback of this approach is its reduced classification performance compared to scenarios where data and model perturbation

Algorithm 9 $\mathsf{InferenceOracle}_{sk,b}(\overrightarrow{inp} = (x_0, \cdots, x_{d-1}), \mathsf{state}, \{c_{model(i)}\}_{i\in[0,\lfloor N_{tot}/M\rfloor]})$

procedure: $\mathsf{InferenceOracle}_{sk}(\overrightarrow{inp}, \mathsf{state}, \{c_i^{(model)}\}_{i\in[0,\lfloor N_{tot}/M\rfloor]}, pk, evk)$
Setup the players User(Client), KM, and CS with state, $\{c_i^{(model)}\}_{i\in[0,\lfloor N_{tot}/M\rfloor]}$, pk, and evk.
$\overrightarrow{inp}$ is given to User.
Run the inference protocol with World-b setting. (See Fig. 2)
Extract what KM obtains while running the inference protocol, which is c_{u_ℓ} and $\overrightarrow{out}$.
return c_{u_ℓ} and $\overrightarrow{out}$

TABLE 11. Summary of related work.

NA: Not Applicable

Requirements	[34]	[35]	[36]	[13]	[15]	[17]	[37]	[27]	[38]	[39]	[14]	[12]	[40]	[41]	[42]	[43]	[44]	[16]	Ours
1)	O	O	O	NA	X	X	O	NA	NA	NA	NA	X	NA	NA	NA	NA	NA	X	O
2)	NA	NA	O	X	O	O	O	O	O	O	O	O	O	O	O	X	O	X	O
3)	X	X	X	NA	X	X	O	NA	NA	NA	NA	X	NA	NA	NA	NA	NA	O	O
4)	X	O	X	O	X	X	X	O	O	O	O	O	O	X	X	X	O	X	O

are not applied. Consequently, in situations where maintaining high classification accuracy is critical, this method proves to be less suitable. The trade-off between privacy protection and classification efficiency poses a notable challenge in the application of these perturbation-based PPNB methods.

To address previously identified issues, we propose a centralized training and classification method using HE to protect model privacy and ensure classification accuracy. The process involves users encrypting data with the system's public key and sending it to the server, which lacks the private key but has evaluation keys. The server computes the model using these encrypted inputs. For classification, it receives encrypted data, processes it, and sends the encrypted result back to the user, who decrypts it with their private key.

This method maintains privacy for both the user's input and the models, with all operations centralized on the server, minimizing communication needs and data exposure risks. The server's processing of encrypted data reduces the likelihood of data breaches, even if compromised or malicious. Furthermore, classification is executed without decrypting input data, ensuring only the requesting user can access the result, thus preserving privacy.

However, this approach has not been widely adopted due to technical challenges. Key among these are the complexities in performing certain arithmetic operations homomorphically, such as division for probability calculations and logarithmic operations in Naive Bayes algorithms. Additionally, the argmax operation, essential for identifying the highest value position in encrypted data during classification, presents another significant challenge. These factors have previously impeded the broader implementation of this method.

To circumvent these challenges, many studies have introduced a separate trusted entity responsible for managing the decryption key. Research like [12] and [13] involves decrypting encrypted values for argmax comparison. In [17], probability calculations involving division are performed post-decryption. Other studies, for instance, [53], [54], operate on the assumption that probability distributions follow a normal distribution. Only a handful of methods, such as [27] and [38], execute the argmax operation on encrypted inputs, but these are either computationally intensive [27] or compromise security for efficiency [38].

Several methods employ additive Homomorphic Encryption [55] as opposed to Fully Homomorphic Encryption [13], [14]. These methods require data decryption for operations above, constrained by the addition-only support of the underlying HE. Additionally, the security of these HE methods [55] is known to be vulnerable to quantum computer attacks.

Another approach to maintain efficiency uses an HE that only secures against ciphertext-only attacks [44]. However, this level of security is generally considered inadequate for practical applications.

In a different vein, some studies use multiple cloud servers for enhanced security [36], [37], [45], [56]. This assumes each server is in a distinct security domain with no possibility of malicious collaboration between servers. Each server handles a fraction of the computation during training and inference, keeping the complete model and results undisclosed, thus preserving privacy. Nevertheless, this approach falls short of meeting the requirement 4) outlined in Section III-B.

Lastly, certain studies focus exclusively on inference, such as [27], [57], and [58], but are not suitable for training purposes.

Table 11 provides a summary of these previous works in relation to the requirements in Section III-B. Unfortunately, no existing method satisfies all these requirements.

B. HOMOMORPHIC LOGARITHM

To date, there appears to be no algorithm capable of computing even an approximate logarithm over encrypted inputs in the range (0,1] using HE. For instance, in [13], logarithmic values for PPNB inference are calculated in an unencrypted form before encryption, thereby not computing logarithms

Input:
• User u_ℓ has an input vector $(x_0, x_1, \cdots, x_{d-1})$ where x_i is a categorical value of X_i ($i \in S_X$).
• **CS** prepares for the model $\{\mathrm{c}_i^{(model)}\}_{i\in[0,\lfloor N_{tot}/M \rfloor]}$ which is the result of training.
Output:
• User u_ℓ obtains a ciphertext $c_{\mathbf{ret}}$ that can be decrypted using sk_{u_ℓ}, where $c_{\mathbf{ret}}$ contains the inference result.
User u_ℓ:
$1 \sim 4$. The same as the inference in the paper. (Step ①,② in Fig. 10-(B))
CS: ($N_{t1} = |\vec{h}_{(x_0,\cdots,x_{d-1})}|$, for every $w \in S_w$, perform step 1 and 2.)
$1 \sim 4$. The same as the inference in the paper. (Step ③-1) in Fig. 10-(B))
5. Generate a random vector $\vec{r} \xleftarrow{\$} [-2^{p-1}, 2^{p-1}]^M$ where p is the precision parameter of CKKS, and $\xleftarrow{\$}$ refers to uniform sampling. (Step ③-2) in Fig. 10-(B))
6. Set $c_r \leftarrow \mathsf{Enc}_{pk_s}(\vec{r})$. (Step ③-3) in Fig. 10-(B))
7. $c_{\mathbf{ret}} \leftarrow \mathsf{Add}(c_{\mathbf{ret}}, c_r)$ (Step ③-4) in Fig. 10-(B))
8. Deliver $c_{\mathbf{ret}}$ to KM with the ID of u_ℓ. (Step ④ in Fig. 10-(B))
9. $c_r' \leftarrow \mathsf{Enc}_{pk_u}(\vec{r})$. (Step ③-5) in Fig. 10-(B))
10. Deliver c_r' to u_ℓ (Step ⑦-2) in Fig. 10-(B))
KM:
1. $\overrightarrow{out} \leftarrow \mathsf{Dec}_{\mathsf{sk}_s}(c_{\mathbf{ret}})$. (Step ⑤ in Fig. 10-(B))
2. $c_{u_\ell} \leftarrow \mathsf{Enc}_{\mathsf{pk}_{u_\ell}}(\overrightarrow{out})$. (Step ⑥ in Fig. 10-(B))
3. Deliver c_{u_ℓ} that can be decrypted with sk_ℓ to u_ℓ. (Step ⑦-1) in Fig. 10-(B))
u_ℓ:
1. $\overrightarrow{out} \leftarrow \mathsf{Dec}_{\mathsf{sk}_{u_\ell}}(c_{u_\ell})$.(Step ⑧-1) in Fig. 10-(B))
2. $\vec{r} \leftarrow \mathsf{Dec}_{\mathsf{sk}_{u_\ell}}(c_r')$ (Step ⑧-2) in Fig. 10-(B))
3. Return $\overrightarrow{out} - \vec{r}$ (Step ⑧-3) in Fig. 10-(B))

FIGURE 11. Inference protocol with semi-honest KM.

on encrypted data. Similarly, [59] proposes a method for calculating logarithmic values in a secret sharing-based Multi-Party Computation (MPC) setting, which, as noted in [5], is not applicable to systems utilizing HE.

Among PPNB methodologies that employ HE, the approach by Kjamilji et al. [16] is the closest to an ideal scenario. This method utilizes the BFV HE scheme [60], [61] and facilitates both training and classification on encrypted data. However, due to the computational intensity of logarithmic and division operations with BFV ciphertexts, this method also resorts to using a decryption key during training. Furthermore, to our knowledge, there is no established method for performing homomorphic logarithm operations directly on BFV ciphertexts, highlighting a significant gap in the current capabilities of HE in PPNB.

X. CONCLUSION

This paper presents a novel method for training and classifying data using the Privacy-Preserving Naive Bayes (PPNB) algorithm, integrated with the Cheon-Kim-Kim-Song (CKKS) homomorphic encryption (HE) algorithm. Our approach facilitates non-interactive training and classification by introducing an efficient way to compute a homomorphic logarithm function. This innovation allows for significantly faster NB training and classification processes, achieving speeds arounds 28 times faster than those reported in [17]. Through computational complexity analysis, the efficiency and effectiveness of our proposed method are confirmed. It proves especially beneficial in situations where intermediate decryption is impractical, such as environments with multiple data owners reluctant to share data, or in cases where the trusted entity with the decryption key has limited computational resources.

For future work, enhancing the approximation of the homomorphic logarithm function to require fewer slots is a consideration. This improvement would enable the parallel processing of a greater number of encrypted values, further optimizing the system's performance.

XI. CONSTANTS FOR HERMES_LOG()

In order to implement Hermes_Log() function, the constants $\mathsf{L}_1 \sim \mathsf{L}_7$ are needed. They are given using python expression as

- L_1 = float.fromhex('0 × 1.5555555555593p-1')
- L_2 = float.fromhex('0 × 1.999999997fa04p-2')
- L_3 = float.fromhex('0 × 1.2492494229359p-2')
- L_4 = float.fromhex('0 × 1.c71c51d8e78afp-3')
- L_5 = float.fromhex('0 × 1.7466496cb03dep-3')
- L_6 = float.fromhex('0 × 1.39a09d078c69fp-3')
- L_7 = float.fromhex('0 × 1.2f112df3e5244p-3')

XII. SUMMARY OF CHANGES

This paper has been expanded from its conference proceedings version [62] with the following additions. First, we have included an inference protocol that operates under the assumption that the Key Management server (KM) is semi-honest and have provided a proof of security for this. The previous conference version only proposed a protocol for when KM is a trusted entity. This content is detailed in Section VIII-C.

Second, a thorough analysis of the errors arising from the proposed approximate homomorphic logarithm calculation method is described in Section VIII-A. This analysis considers errors from homomorphic operations, the approximation of logarithm operations, and errors introduced by bootstrapping operations, ultimately proving that the relative error is less than 0.01% for the homomorphic encryption parameters used in this research.

Third, we have significantly enhanced the related work section and conducted a detailed analysis based on the requirements set forth in this paper. The results of this analysis are outlined in Table 10. Additionally, we have included a survey and analysis of previous research related to homomorphic logarithm operations in the related work section IX-B.

REFERENCES

[1] P. Voigt and A. Von dem Bussche, "The EU general data protection regulation (GDPR)," in *A Practical Guide* (GDPR), vol. 10, 1st ed., Cham, Switzerland: Springer, 2017.
[2] E. Goldman, "An introduction to the California consumer privacy act (CCPA)," Santa Clara Univ. Legal Stud. Res. Paper, Santa Clara, CA, USA, 2020.

[3] J. L. Hellerstein, T. S. Jayram, and I. Rish, "Recognizing end-user transactions in performance management," in *Proc. 17th Nat. Conf. Artif. Intell. 12th Conf. Innov. Appl. Artif. Intell.*, Jul. 2000, pp. 596–602.

[4] T. Mitchell, B. Buchanan, G. DeJong, T. Dietterich, P. Rosenbloom, and A. Waibel, "Machine learning," *Annu. Rev. Comput. Sci.*, vol. 4, no. 1, pp. 417–433, 1990.

[5] M. Abbas, K. A. Memon, A. A. Jamali, S. Memon, and A. Ahmed, "Multinomial Naïve Bayes classification model for sentiment analysis," *Int. J. Comput. Sci. Netw. Secur.*, vol. 19, no. 3, p. 62, 2019.

[6] Y. Nurdiansyah, S. Bukhori, and R. Hidayat, "Sentiment analysis system for movie review in bahasa Indonesia using naive Bayes classifier method," *J. Phys., Conf. Ser.*, vol. 1008, Apr. 2018, Art. no. 012011.

[7] V. A. Fitri, R. Andreswari, and M. A. Hasibuan, "Sentiment analysis of social media Twitter with case of anti-LGBT campaign in Indonesia using Naïve Bayes, decision tree, and random forest algorithm," *Proc. Comput. Sci.*, vol. 161, pp. 765–772, Jan. 2019.

[8] C. Slamet, R. Andrian, D. S. Maylawati, W. Darmalaksana, and M. Ramdhani, "Web scraping and Naïve Bayes classification for job search engine," *IOP Conf. Ser., Mater. Sci. Eng.*, vol. 288, no. 1, 2018, Art. no. 012038.

[9] E. S. Berner, *Clinical Decision Support Systems*, vol. 233. Berlin, Germany: Springer, 2007.

[10] M. A. Musen, B. Middleton, and R. A. Greenes, "Clinical decision-support systems," in *Biomedical Informatics*. Berlin, Germany: Springer, 2021, pp. 795–840.

[11] C. Schurink, P. Lucas, I. Hoepelman, and M. Bonten, "Computer-assisted decision support for the diagnosis and treatment of infectious diseases in intensive care units," *Lancet Infectious Diseases*, vol. 5, no. 5, pp. 305–312, May 2005.

[12] C.-Z. Gao, Q. Cheng, P. He, W. Susilo, and J. Li, "Privacy-preserving Naïve Bayes classifiers secure against the substitution-then-comparison attack," *Inf. Sci.*, vol. 444, pp. 72–88, May 2018.

[13] R. Bost, R. A. Popa, S. Tu, and S. Goldwasser, "Machine learning classification over encrypted data," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, vol. 4324, 2015, p. 4325.

[14] T. Li, Z. Huang, P. Li, Z. Liu, and C. Jia, "Outsourced privacy-preserving classification service over encrypted data," *J. Netw. Comput. Appl.*, vol. 106, pp. 100–110, Mar. 2018.

[15] X. Liu, R. Lu, J. Ma, L. Chen, and B. Qin, "Privacy-preserving patient-centric clinical decision support system on Naïve Bayesian classification," *IEEE J. Biomed. Health Informat.*, vol. 20, no. 2, pp. 655–668, Mar. 2016.

[16] A. Kjamilji, E. Savas, and A. Levi, "Efficient secure building blocks with application to privacy preserving machine learning algorithms," *IEEE Access*, vol. 9, pp. 8324–8353, 2021.

[17] X. Liu, R. H. Deng, K. R. Choo, and Y. Yang, "Privacy-preserving outsourced clinical decision support system in the cloud," *IEEE Trans. Services Comput.*, vol. 14, no. 1, pp. 222–234, Jan. 2021.

[18] W. Wolberg, "Breast cancer Wisconsin (original)," UCI Mach. Learn. Repository, Univ. California, Irvine, CA, USA, 1992.

[19] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur.* Berlin, Germany: Springer, 2017, pp. 409–437.

[20] W. Jung, S. Kim, J. H. Ahn, J. H. Cheon, and Y. Lee, "Over 100× faster bootstrapping in fully homomorphic encryption through memory-centric optimization with GPUs," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2021, no. 4, pp. 114–148, 2021, doi: 10.46586/tches.v2021.i4.114-148.

[21] K. Han and D. Ki, "Better bootstrapping for approximate homomorphic encryption," in *Proc. Cryptogr. Track RSA Conf.* Berlin, Germany: Springer, 2020, pp. 364–390.

[22] J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song, "A full RNS variant of approximate homomorphic encryption," in *Proc. Int. Conf. Sel. Areas Cryptogr.* Berlin, Germany: Springer, 2018, pp. 347–368.

[23] J. W. Lee, E. Lee, Y. Lee, Y.-S. Kim, and J.-S. No, "High-precision bootstrapping of RNS-CKKS homomorphic encryption using optimal minimax polynomial approximation and inverse sine function," in *Proc. 40th Annu. Int. Conf. Theory Appl. Cryptograph. Techn.* Berlin, Germany: Springer, 2021, pp. 618–647.

[24] J. H. Cheon, D. Kim, and D. Kim, "Efficient homomorphic comparison methods with optimal complexity," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Security (ASIACRYPT)*. Berlin, Germany: Springer, 2020, pp. 221–256.

[25] CryptoLab. (2022). *HEAAN Library*. [Online]. Available: https://heaan.it/

[26] Y. Lee, J. Seo, Y. Nam, J. Chae, and J. H. Cheon, "HEaaN-STAT: A privacy-preserving statistical analysis toolkit for large-scale numerical, ordinal, and categorical data," *IEEE Trans. Depend. Secure Comput.*, vol. 21, no. 3, pp. 1224–1241, May/Jun. 2024, doi: 10.1109/TDSC.2023.3275649.

[27] H. Park, P. Kim, H. Kim, K.-W. Park, and Y. Lee, "Efficient machine learning over encrypted data with non-interactive communication," *Comput. Standards Interfaces*, vol. 58, pp. 87–108, May 2018.

[28] E. Y. Remez, "Sur la détermination des polynômes d'approximation de degré donnée," *Commun. Soc. Math. Kharkov*, vol. 10, no. 196, pp. 41–63, 1934.

[29] K. E. Atkinson, *An Introduction to Numerical Analysis*. New York, NY, USA: Wiley, 1989.

[30] D. Hermes. (2017). *Remez Algorithm for Log(x)*. [Online]. Available: https://gist.github.com/dhermes/105da2a3c9861c90ea39#file-remez-pdf

[31] J. Czerniak, "Acute inflammations," UCI Mach. Learn. Repository, Tech. Rep., 2009.

[32] *Car Evaluation*, UCI Mach. Learn. Repository, Univ. California, Irvine, CA, USA, 1997.

[33] B. Li, D. Micciancio, M. Schultz, and J. Sorrell, "Securing approximate homomorphic encryption using differential privacy," in *Proc. Annu. Int. Cryptol. Conf.* Springer, 2022, pp. 560–589.

[34] J. Vaidya, M. Kantarcıoğlu, and C. Clifton, "Privacy-preserving Naïve Bayes classification," *VLDB J.*, vol. 17, no. 4, pp. 879–898, 2008.

[35] Z. Yang, S. Zhong, and R. N. Wright, "Privacy-preserving classification of customer data without loss of accuracy," in *Proc. SIAM Int. Conf. Data Mining*. Philadelphia, PA, USA: SIAM, Apr. 2005, pp. 92–102.

[36] X. Yi and Y. Zhang, "Privacy-preserving Naïve Bayes classification on distributed data via semi-trusted mixers," *Inf. Syst.*, vol. 34, no. 3, pp. 371–380, May 2009.

[37] P. Li, J. Li, Z. Huang, C.-Z. Gao, W.-B. Chen, and K. Chen, "Privacy-preserving outsourced classification in cloud computing," *Cluster Comput.*, vol. 21, no. 1, pp. 277–286, Mar. 2018.

[38] X. Sun, P. Zhang, J. K. Liu, J. Yu, and W. Xie, "Private machine learning classification based on fully homomorphic encryption," *IEEE Trans. Emerg. Topics Comput.*, vol. 8, no. 2, pp. 352–364, Apr. 2020.

[39] A. Khedr, G. Gulak, and V. Vaikuntanathan, "SHIELD: Scalable homomorphic implementation of encrypted data-classifiers," *IEEE Trans. Comput.*, vol. 65, no. 9, pp. 2848–2858, Nov. 2016.

[40] A. Kjamilji, A. Idrizi, S. Luma-Osmani, and F. Zenuni-Kjamilji, "Secure Naïve Bayes classification without loss of accuracy with application to breast cancer prediction," in *Proc. Int. Conf. Sci. Eng.*, vol. 3, 2020, pp. 397–403.

[41] H. V. L. Pereira, "Efficient AGCD-based homomorphic encryption for matrix and vector arithmetic," in *Proc. Int. Conf. Appl. Cryptogr. Netw. Secur.* Berlin, Germany: Springer, 2020, pp. 110–129.

[42] Y. Yasumura, Y. Ishimaki, and H. Yamana, "Secure Naïve Bayes classification protocol over encrypted data using fully homomorphic encryption," in *Proc. 21st Int. Conf. Inf. Integr. Web-Based Appl. Services*, 2019, pp. 45–54.

[43] A. Wood, V. Shpilrain, K. Najarian, A. Mostashari, and D. Kahrobaei, "Private-key fully homomorphic encryption for private classification," in *Proc. 6th Int. Conf. Math. Softw. (ICMS)*, South Bend, IN, USA. Berlin, Germany: Springer, Jul. 2018, pp. 475–481.

[44] A. Wood, V. Shpilrain, K. Najarian, and D. Kahrobaei, "Private Naïve Bayes classification of personal biomedical data: Application in cancer data analysis," *Comput. Biol. Med.*, vol. 105, pp. 144–150, Feb. 2019.

[45] D.-H. Vu, "Privacy-preserving Naïve Bayes classification in semi-fully distributed data model," *Comput. Secur.*, vol. 115, Apr. 2022, Art. no. 102630.

[46] H. Kaur, N. Kumar, and S. Batra, "ClaMPP: A cloud-based multi-party privacy preserving classification scheme for distributed applications," *J. Supercomput.*, vol. 75, no. 6, pp. 3046–3075, Jun. 2019.

[47] J. Vaidya and C. Clifton, "Privacy preserving Naïve Bayes classifier for vertically partitioned data," in *Proc. SIAM Int. Conf. Data Mining*. Philadelphia, PA, USA: SIAM, Apr. 2004, pp. 522–526.

[48] M. Kantarcıoglu, J. Vaidya, and C. Clifton, "Privacy preserving Naïve Bayes classifier for horizontally partitioned data," in *Proc. IEEE ICDM Workshop Privacy Preserving Data Mining*, Nov. 2003, pp. 3–9.

[49] W. Tang, Y. Zhou, Z. Wu, L. Lu, and M. Li, "Naïve Bayes classification based on differential privacy," in *Proc. Int. Conf. Artif. Intell. Adv. Manuf.*, Oct. 2019, pp. 1–6.

[50] J. Vaidya, B. Shafiq, A. Basu, and Y. Hong, "Differentially private Naïve Bayes classification," in *Proc. IEEE/WIC/ACM Int. Joint Conf. Web Intell. (WI) Intell. Agent Technol. (IAT)*, vol. 1, Nov. 2013, pp. 571–576.
[51] A. D. Sarwate and K. Chaudhuri, "Signal processing and machine learning with differential privacy: Algorithms and challenges for continuous data," *IEEE Signal Process. Mag.*, vol. 30, no. 5, pp. 86–94, Sep. 2013.
[52] B. C. M. Fung, K. Wang, and P. S. Yu, "Top-down specialization for information and privacy preservation," in *Proc. 21st Int. Conf. Data Eng. (ICDE)*, 2005, pp. 205–216.
[53] X. Wang, J. Ma, Y. Miao, R. Yang, and Y. Chang, "EPSMD: An efficient privacy-preserving sensor data monitoring and online diagnosis system," in *Proc. IEEE Conf. Comput. Commun. (IEEE INFOCOM)*, Apr. 2018, pp. 819–827.
[54] X. Wang, J. Ma, Y. Miao, X. Liu, and R. Yang, "Privacy-preserving diverse keyword search and online pre-diagnosis in cloud computing," *IEEE Trans. Services Comput.*, vol. 15, no. 2, pp. 710–723, Mar. 2022.
[55] P. Pallier, "Public-key cryptosystems based on composite degree residuosity classes," in *Proc. EUROCRYPT*, in Lecture Notes in Computer Science, vol. 1592, 1999, pp. 223–238.
[56] X. Zhao and Z. Xia, "Secure outsourced NB: Accurate and efficient privacy-preserving Naïve Bayes classification," *Comput. Secur.*, vol. 124, Jan. 2023, Art. no. 103011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404822004035
[57] J. Chen, Y. Feng, Y. Liu, W. Wu, and G. Yang, "Non-interactive privacy-preserving Naïve Bayes classifier using homomorphic encryption," in *Proc. Int. Conf. Secur. Privacy New Comput. Environ.* Berlin, Germany: Springer, 2021, pp. 192–203.
[58] A. Kjamilji, A. Levi, E. Savaş, and O. B. Güney, "Secure matrix operations for machine learning classifications over encrypted data in post quantum industrial IoT," in *Proc. Int. Symp. Netw., Comput. Commun. (ISNCC)*, Oct. 2021, pp. 1–8.
[59] A. Aly and N. P. Smart, "Benchmarking privacy preserving scientific operations," in *Proc. Int. Conf. Appl. Cryptogr. Netw. Secur.* Berlin, Germany: Springer, 2019, pp. 509–529.
[60] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," Cryptol. ePrint Arch., Berlin, Germany, Tech. Rep. 2012/144, 2012.
[61] Microsoft Research, Redmond, WA, USA. (Feb. 2019). *Microsoft SEAL (release 3.2)*. [Online]. Available: https://github.com/Microsoft/SEAL
[62] B. Han, Y. Kim, J. Choi, H. Shin, and Y. Lee, "Fully homomorphic privacy-preserving Naïve Bayes machine learning and classification," in *Proc. 11th Workshop Encrypted Comput. Appl. Homomorphic Cryptogr.* New York, NY, USA: Association for Computing Machinery, Nov. 2023, pp. 91–102, doi: 10.1145/3605759.3625262.

BOYOUNG HAN received the B.S. degree in IT management and the M.S. degree in data science from Seoul National University of Science and Technology, South Korea, in 2020 and 2023, respectively. Her current research interests include private AI and applied AI.

HOJUNE SHIN received the B.S. degree in industrial and information systems engineering from Seoul National University of Science and Technology, South Korea, in 2022, where he is currently pursuing the M.S. degree in data science. His research interests include private AI and homomorphic encryption.

YEONGHYEON KIM received the B.S. degree in IT management from Seoul National University of Science and Technology, South Korea, in 2022. He is currently with CryptoLab, South Korea. His research interests include homomorphic encryption and private AI.

JINA CHOI received the B.S. degree in IT management from Seoul National University of Science and Technology, South Korea, in 2024, where she is currently pursuing the M.S. degree in data science. Her research interests include private AI and homomorphic encryption.

YOUNHO LEE (Member, IEEE) received the B.S., M.S., and Ph.D. degrees in computer science from KAIST, South Korea, in 2000, 2002, and 2006, respectively. He was a Visiting Postdoctoral Researcher and a Research Staff with GeorgiaTech Information Security Center, from 2007 to 2009. He is currently a Professor with the Department of Data Science, Seoul National University of Science and Technology, South Korea. His research interests include applied cryptography and data security.