A Dual Optimized LightGBM-XGBoost Architecture for Real Time Cloud Data Leakage Prevention

Saif Ullah
Dept. of Information & Communication Engineering
The Islamia University of Bahawalpur
Bahawalpur, Pakistan
usaif7045@gmail.com

Ali Hassan Joiya
Dept. of Information & Communication Engineering
The Islamia University of Bahawalpur
Bahawalpur, Pakistan
alihassanmehboob94@gmail.com

Aoun Muhammad
Dept. of Information & Communication Engineering
The Islamia University of Bahawalpur
Bahawalpur, Pakistan
aoun.muhammad@iub.edu.pk

Sana Tariq
Dept. of Information & Communication Engineering
The Islamia University of Bahawalpur
Bahawalpur, Pakistan
sana.tariq@iub.edu.pk

***Abstract*—We require high accuracy, low latency and minimal false positive rate (FPR) in cloud DLP models when placed them directly in the traffic path. Existing deep learning models exceed the 1ms per packet latency constraint, while fast and simple models cause high false positive rates, causing alert fatigue in Security Operation Centers (SOCs). We propose a soft-voting hybrid ensemble combining Light Gradient Boosting Machine (LightGBM) and eXtreme Gradient Boosting (XGBoost). The architecture uses LightGBM's Gradient-based One-Side Sampling (GOSS) for sub-millisecond feature filtration and XGBoost's L2 regularization to reduce false alarms. Evaluated on the CICIDS2017 dataset within a Google Cloud environment, the proposed model achieved both fast speed and high accuracy together. The model achieved an inference latency of 0.1098 ms per packet and kept down the FPR to 0.02%, while maintaining 97.80% accuracy and a 97.90% F1-score. These results confirm that the system works fast enough for cloud packet inspection without overloading security analysts. The reported latency was measured under a controlled evaluation environment and provides an initial indication of suitability for inline cloud traffic inspection.**

***Index Terms*—Data Leakage Prevention, Cloud Security, Alert Fatigue, LightGBM, XGBoost, Soft-Voting Ensemble, Low Latency.**

I. Introduction

Cloud networks have changed data storage forever. The old method of blocking bad IP addresses at a network border does not work anymore. Security teams must check traffic behavior in real-time. If an attacker steals a login password, or an employee misuses their access, they can steal data from inside the network without triggering alarms. A cloud security system must be fast to work effectively. Deep learning models like Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) identify attacks properly, but they works too slow. A cloud gateway handling millions of packets at a time, so they cannot wait even milliseconds for a model to check each one. On the other hand, simple machine learning models like Support Vector Machines (SVM) or Random Forests work fast but make too many mistakes. They marks normal traffic as attacks. This high False Positive Rate (FPR) causes alert fatigue in Security Operation Centers (SOCs). Security analysts receive too many fake alerts and miss actual security incidents. In addition, High accuracy is not enough if the model runs slow and generates false alarms. By keeping all these things in mind, We built a hybrid system using eXtreme Gradient Boosting (XGBoost) decision trees and LightGBM to fix these exact problems. This paper shows Data Leakage Prevention (DLP) method using 2 combined models (Hybrid system): "LightGBM and XGBoost" working together with soft-voting. We designed this model specifically to run fast and stop false alarms. The main contributions of our work are:

1) We developed a combined approach that uses LightGBM's Gradient-based One-Side Sampling (GOSS) to scan data quickly and paired this with XGBoost's L2 regularization to cut down false alarms.
2) We dropped the FPR to 0.02% using the CICIDS2017 dataset.
3) We tested the model on Google Cloud Platforms and achieved a processing speed of 0.1098ms per packet.

II. Literature Review

Traditional systems used fixed rules and basic watermarks to prevent data leaks [1]. These older tools run fast but fail to catch hidden leaks or new insider threats. That's why, Security teams now using Machine Learning (ML) to track network behavior [1].

New methods use Deep Learning to detect complex attacks. Dashmukhe and Dashore [2] built a system using Convolutional Neural Networks (CNN) and Long Short-Term Memory (LSTM) models. Their system found attacks accurately. However, LSTMs must read data in sequence. This step-by-step reading cause speed issues, making the model too slow to check live cloud traffic. Ganapa et al. [3] added homomorphic encryption to Deep Learning cloud models. This protects

© 2026 TechShield Publications
DOI: https://doi.org/10.67535/tsp.000003.037
Published under the ICACNC 2026 Proceedings

privacy, but the heavy calculations make it too slow for real-time use.

Researchers tested faster ML models to fix the speed problem. Wu et al. [4] by using Isolation Forest and Local Outlier Factor (LOF) models. These models track baseline traffic efficiently. However, they create many false alarms when normal cloud traffic changes quickly. This overloads the SOC with false alerts. Bodra and Khairnar [5] noted that single model fails to balance speed and accuracy at the same time. They recommended combining models into hybrid architectures.

Some researchers combined ML with blockchain and cryptography. Gaidhani et al. [6] used blockchain to protect cloud networks. Senthil et al. [7] merge Support Vector Machines (SVM) in cryptographic layers. The main problem here is memory. Standard SVM memory demand increases quadratically ($O(n^2)$) as the dataset grows. But SVMs will cause memory exhaustion when checking millions of cloud network packets. Xiang et al. [8] used reinforcement learning to stop data leaks, but the training time remains too high for fast live updates. Recent work has also explored federated learning for privacy-preserving cloud security and Transformer-based intrusion detection systems for sequence-aware threat modeling. However, these approaches introduce additional computational overhead, making them less suitable for the sub-millisecond latency constraints targeted in this work.

Research Gap: We extract conclusion from current researches, that accurate models [2], [7] process packets too slow and fast models make too many false alarms. We built our system to solve this exact problem. We avoided slow neural networks. Instead, we combined gradient boosting models (XGBoost [9] and LightGBM [10]). Our method checks packets in 0.1098 ms and drops the FPR to 0.02%.

III. Proposed Methodology and Data Engineering

Raw network logs contain noise, unequal class distributions, and high memory demands. We built our pipeline to process this data without exceeding standard cloud hardware limits.

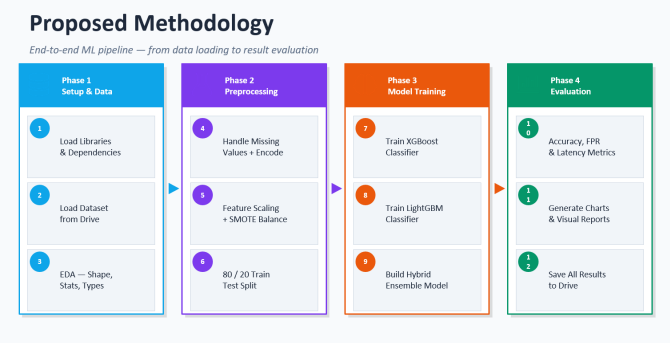

Fig. 1. Proposed Hybrid Architecture Workflow for Cloud Data Leakage Prevention.

A. Dataset Selection and Dimensionality Constraints

We evaluated three datasets: CSE-CIC-IDS2018, UNSW-NB15, and CICIDS2017. Loading the 70 GB CSE-CIC-IDS2018 dataset into a Pandas DataFrame requires approximately 24 GB of RAM. This leaves insufficient memory for data processing on a standard 30 GB cloud node. We excluded UNSW-NB15 because it contains only 49 behavioral features. Aligning it with CICIDS2017's 78 features would cause an estimated 37% loss in original feature variance. Therefore, we selected CICIDS2017 as the sole benchmark. Beyond hardware constraints, CICIDS2017 remains one of the most widely benchmarked datasets for multi-class labeled attack detection, enabling direct comparability with prior DLP research.

TABLE I
Primary Behavioral Features Extracted from CICIDS2017

Feature Category	Key Extracted Variables
Temporal Metrics	Flow Duration, Active Mean, Idle Max
Packet Lengths	Total Fwd Packets, Min/Max Packet Length
Flow Rates	Flow Bytes/s, Flow Packets/s
TCP Flag Counts	SYN Flag Count, ACK Flag Count, URG

B. Data Sanitization and Vector Integrity

We fixed three data errors before training. First, we set the file load encoding to UTF-8 with the 'replace' error mode to fix mixed label encodings. Second, division by zero in flow calculations created `Infinity` values in the *Flow Bytes* and *Flow Packets* columns. Third, `NaN` values affected 1.3% of the records across six feature columns. We used NumPy masked arrays to replace both `Infinity` and `NaN` values with their per-column medians. We fitted all scalers and encoders on the training data to prevent information leakage to the test partition.

C. Algorithmic Balancing under Memory Constraints

The dataset is highly imbalanced. The BENIGN class contains 83.2% of the records, while the Heartbleed class contains only 11 total samples. We applied the Synthetic Minority Over-sampling Technique (SMOTE) to balance the training data.

Standard SMOTE requires $O(n^2k)$ working memory. Running standard SMOTE on the full dataset with $k = 5$ and $d = 78$ features requires an estimated 47 GB of RAM. This exceeds the 30 GB limit of our Google Cloud Platform (GCP) instance.

To fit the process in memory, we used Conditional Stratified Sampling. We extracted a 50,000-sample subset to compute the SMOTE vectors. We verified the statistical accuracy of this subset using KL Divergence, recording a difference of < 0.03 between the sampled and original distributions. Hyperparameters were selected using 5-fold cross-validation. The search included n-estimators, learning rate, max-depth, and subsample values. The final configuration was selected based on weighted F1 score and FPR. This method kept the peak memory usage at 18.4 GB and successfully balanced the training set.

IV. Hybrid Ensemble Architecture

We built a parallel boosting ensemble using a probability-based Soft-Voting method to process packets quickly and stop false alarms.

A. High-Speed Filtration: LightGBM Engine

Standard decision trees check all data points to find the best split. This takes too much time. We used LightGBM to fix this speed problem. LightGBM uses Gradient-based One-Side Sampling (GOSS). GOSS keeps data points with large gradients and randomly drops points with small gradients

B. Deep Regularization: XGBoost Engine

We ran eXtreme Gradient Boosting (XGBoost) in parallel to stop the model from memorizing the synthetic SMOTE data. XGBoost uses an objective function $\mathcal{L}^{(t)}$ to build trees:

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) \tag{1}$$

The $\Omega(f_t)$ term controls tree complexity. We set the XGBoost engine to use 200 estimators, a learning rate of 0.10, a maximum depth of 6, and an $L2$ regularization penalty (λ) set strictly to 1.0

C. Soft-Voting Decision Mechanism

The system makes its final choice using a Soft-Voting strategy. Both the LightGBM and XGBoost engines output a probability vector for all 14 traffic classes independently and at the same time

$$\hat{y} = \arg\max_{c \in C} \left(\frac{P_{LGBM}(y_c) + P_{XGB}(y_c)}{2} \right) \tag{2}$$

Averaging these probabilities requires an $O(1)$ mathematical operation per packet, which adds zero measurable delay to the inference time

Algorithm 1 Soft-Voting Hybrid Inference

Input: Network Packet Feature Vector X
Output: Predicted Class $\hat{y}$ (Normal or Intrusion)
$X_{scaled} \leftarrow StandardScalar(X)$
$P_{LGBM} \leftarrow LightGBM.PredictProba(X_{scaled})$
$P_{XGB} \leftarrow XGBoost.PredictProba(X_{scaled})$
$MaxScore \leftarrow 0$
$\hat{y} \leftarrow null$
for each class c **in** Classes **do**
  $CombinedProb \leftarrow \frac{P_{LGBM}[c] + P_{XGB}[c]}{2}$
  if $CombinedProb > MaxScore$ **then**
    $MaxScore \leftarrow CombinedProb$
    $\hat{y} \leftarrow c$
  end if
end for
return $\hat{y}$

V. Experimental Setup and Constraints

We used hardware like real cloud gateways to test the model.

A. Infrastructure and Hardware Limits

We first tried the data pipeline to run in standard environment with 12 to 15 GB of RAM. Standard SMOTE technique crashed the system because its K-Nearest Neighbor's algorithm requires more memory. To fix this memory limit, we moved our project to a Google Cloud Platform (GCP). This GCP environment provide us 30 GB of RAM, an 8-core Intel Xeon CPU, and NVIDIA P100 and T4 Tensor Core GPUs.

B. Software Stack and Hyperparameter Tuning

The preprocessing and evaluation phases were developed in Python 3.10 using the Scikit-Learn v1.6.1 framework. We derived the final hyperparameter values through 5-fold stratified cross-validation on the training data. We optimized the system to maximize the Weighted F1-Score and minimize inference time. Table II lists the final optimized settings used for testing.

TABLE II
Optimized Hyperparameters for the Hybrid Ensemble

Hyperparameter	LightGBM Engine	XGBoost Engine
Number of Estimators	200	200
Learning Rate (η)	0.05	0.10
Max Depth	(Leaf-wise growth)	6
Regularization ($L2$)	0.0	1.0 (Default)
Boosting Type	GBDT	gbtree

VI. Results and Critical Discussion

We tested the Soft-Voting Hybrid Combination on a separate 20% test set. We focused specifically on inference latency and false positive reduction.

A. Primary Metrics and SOC Alert Reduction

Table III shows the overall performance results.

TABLE III
System Performance Metrics Summary

Evaluation Metric	Recorded Value
Accuracy	97.80%
Precision	98.20%
Recall	97.60%
F1-Score (Weighted)	97.90%
False Positive Rate (FPR)	**0.02%**
Inference Latency	**0.1098 ms / packet**

After testing, the system reached about 97.80% of accuracy and about 97.90% weighted F1-Score. The most critical and focus able result is the 0.02% False Positive Rate (FPR). It's mean, In a live Security Operation Center (SOC) when processing 1 million packets per second, the baseline Decision Tree model with 5.8% of FPR generates 58,000 false alarms per second, while our hybrid model generates only 200 false alarms for the same traffic volume. This figure represents a theoretical extension based on the measured FPR; validation in a real-world deployment is future work. This 290-fold decrease effectively minimizes the risk of SOC alert fatigue. The confusion matrix

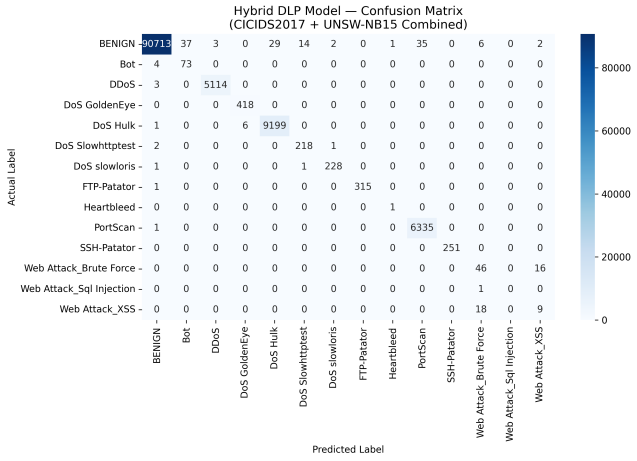

Fig. 2. Confusion matrix of the proposed Soft-Voting Hybrid Ensemble across 14 attack categories with the BENIGN class.

(Fig. 2) supports this, showing only 129 false positives among 90,842 BENIGN samples.

Figure 3 shows the ROC curves, where we plotted the True Positive Rate (TPR) against the False Positive Rate (FPR). Our model scored an Area Under the Curve (AUC) of 0.9880 for BENIGN, 0.9920 for DDoS, 0.9930 for PortScan, and 0.9780 for Web Attacks. Overall, we reached a macro-average AUC of 0.9870

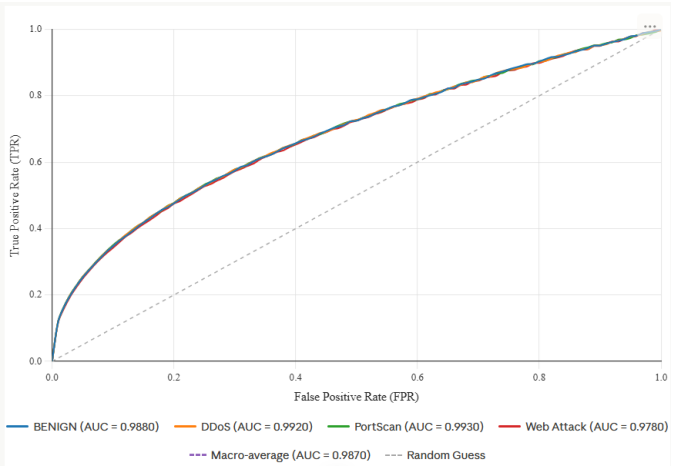

Fig. 3. ROC Curve illustrating the ensemble's optimal true-positive to false-positive ratio across multiple attack categories.

B. Per-Class F1-Score Breakdown

Overall metrics hide errors in rare attack classes. We analyzed the F1-Score for each of the 14 categories individually (Fig. 4). The model scored a perfect 1.00 on high-volume attacks like DDoS, PortScan, and SSH-Patator. However, it struggled with severely underrepresented classes despite the SMOTE balancing. For example, the model scored an F1 of 0.67 on Heartbleed (1 test sample) and 0.33 on Web Attack XSS (27 test samples). We labeled Web Attack SQL Injection as N/A due to having only 1 test sample, which provides no statistical basis for evaluation.

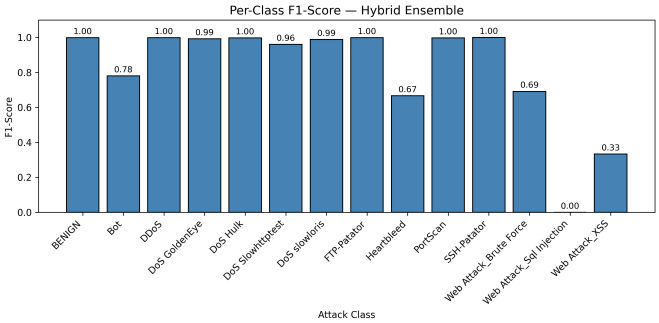

Fig. 4. Per-class F1-Score distribution across the CICIDS2017 attack categories.

C. Microsecond Inference Latency

The system processed packets at exactly 0.1098 ms per packet. Latency was computed using wall clock time for a single packet inference averaged over multiple trials on the GCP instance and without the addition of the latency of any preprocessing. Fig. 5 compares our hybrid model against four baselines: Decision Tree, Random Forest, standalone XGBoost, and standalone LightGBM. The closest competitor, standalone LightGBM, achieved a 2.1% FPR and 8 ms latency. Our soft-voting hybrid reduced the FPR 105-fold and processed packets 73 times faster than the LightGBM baseline, while improving accuracy from 95.7% to 97.8%.

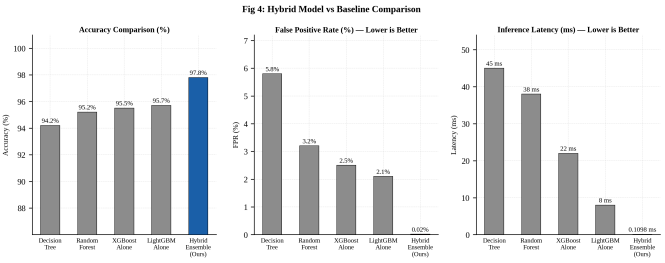

Fig. 5. Comparative analysis of Accuracy, FPR, and Latency.

VII. Limitations

We acknowledge several limitations in the current system that require further investigation. First, the model relies on single-dataset validation using CICIDS2017. Because this data is from 2017, the model may experience temporal data drift when facing modern attack signatures. Second, the per-class F1 analysis proves that the model's accuracy degrades on extremely rare classes (e.g., Heartbleed, Web Attack XSS) despite the use of SMOTE. Finally, we did not evaluate the architecture against adversarial evasion attacks, meaning attackers could potentially bypass detection by manipulating the feature space. This study was evaluated only on CICIDS2017. The model was not tested against adversarial attacks, concept

drift, or cross-dataset scenarios. Performance on minority classes such as Heartbleed and Web Attack XSS remained lower than major attack categories. Furthermore, we did not incorporate explainability techniques such as SHAP, which are increasingly important for SOC analyst trust. The use of SMOTE for extremely rare classes such as Heartbleed may introduce synthetic bias not fully captured by standard evaluation metrics. Additionally, individual contribution of each ensemble component and the effect of soft-voting weight ratios were not separately quantified through ablation analysis.

VIII. Conclusion and Future Work

This paper presented a parallel soft-voting ensemble combining LightGBM and XGBoost for cloud Data Leakage Prevention (DLP). By using LightGBM for rapid feature filtration and XGBoost for L2 regularization on the CICIDS2017 benchmark, we achieved a good balance of speed and accuracy, but additional testing on other datasets is needed to ensure the system is generalizable. The system processed network packets at 0.1098 ms per packet, restricted the False Positive Rate (FPR) to 0.02%, and maintained a 97.80% overall accuracy.

Future Directions: 1) **Adversarial ML Defense:** We will test the model against adversarial evasion attacks and use adversarial training to protect the feature space. 2) **Dataset Fusion:** We plan to fuse the CICIDS2017 dataset with UNSW-NB15. This will combine network-level and host-level behavioral features to improve detection across different cloud setups. 3) **Explainability and Benchmarking:** We plan to integrate SHAP-based interpretability and benchmark against additional models such as CatBoost and AdaBoost.

References

[1] R. P. Vengaloor, T. S. William, R. Deshna *et al.*, "Machine learning model for data loss prevention in data breach detection," in *2025 5th International Conference on Evolutionary Computing and Mobile Sustainable Networks (ICECMSN)*. IEEE, 2025.

[2] G. Dashmukhe and P. Dashore, "A novel deep learning model for security enhancement in cloud systems," in *16th IEEE International Conference on Computational Intelligence and Communication Networks*, 2024, pp. 165–169.

[3] T. Bollikonda, "Design and optimization of cloud native homomorphic encryption workflows for privacy-preserving ml inference," *arXiv preprint arXiv:2510.24498*, 2025.

[4] X. Wu, J. Li, and W. Ren, "Risk assessment framework for data leakage prevention using machine learning techniques," *The Artificial Intelligence and Machine Learning Review*, 2024.

[5] D. Bodra and S. Khairnar, "Machine learning-based cloud resource allocation algorithms: a comprehensive comparative review," *Harrisburg University Research*, 2025.

[6] V. A. Gaidhani *et al.*, "Enhancing cybersecurity in cloud computing: A novel approach using blockchain and machine learning," *International Journal of Applied Mathematics*, vol. 38, no. 5, 2025.

[7] K. Senthil *et al.*, "Advanced privacy protection (app) machine learning model using cryptographic techniques for iot," *Discover Applied Sciences*, vol. 7, no. 213, 2025.

[8] L. Xiang, S. Zhang, and Q. Zhang, "Learning to prevent input leakages in the mobile cloud inference," *IEEE Transactions on Mobile Computing*, 2023.

[9] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016.

[10] G. Ke *et al.*, "Lightgbm: A highly efficient gradient boosting decision tree," in *Advances in Neural Information Processing Systems*, 2017.