Journal of
Cyber Security

Tech Science Press

https://doi.org/10.32604/jcs.2026.085303

ARTICLE

# Ensemble-Guided Pseudorandom Number Generation with Residue Number System Transformation

**Issah Zabsonre Alhassan[1,2,*], Gaddafi Abdul-Salaam[1], Michael Asante[1], Yaw Marfo Missah[1] and Alimatu Sadia Shirazu[1]**

[1]Department of Computer Science, Faculty of Physical and Computational Sciences, Kwame Nkrumah University of Science and Technology, Kumasi, Ghana

[2]Department of Medical Imaging, School of Allied Health Sciences, University for Development Studies, Tamale, Ghana

*Corresponding Author: Issah Zabsonre Alhassan. Email: issahzab@uds.edu.gh

Received: 08 May 2026; Accepted: 23 July 2026; Published: 21 August 2026

**ABSTRACT:** A hybrid approach to pseudorandom number generation that couples ensemble learning with the Residue Number System (RNS) is presented in this paper. Unlike conventional deterministic generators that depend solely on direct algorithmic transformation, the proposed method first maps a seed-driven integer sequence into its RNS representation under the coprime moduli set {3, 5, 7, 11}, whose dynamic range is M = 1155, thereby introducing modular non-linearity through a static, stateless feature transformation. A soft-voting ensemble of Logistic Regression, Random Forest, and Support Vector Machine then serves as a decision layer that classifies and remaps the transformed values into the output sequence. To improve rigour and reproducibility, this revised version reports the full experimental workflow, the hyperparameter-tuning procedure, an ablation study, a quantitative and qualitative comparison with previously reported Pseudorandom Number Generators (PRNGs), an explicit separation of one-time training cost from online generation cost, and an extended security-oriented assessment. The generated sequences were evaluated using classification metrics, Shannon entropy, histogram and autocorrelation analyses, the Kolmogorov-Smirnov test, and an expanded NIST SP 800-22 battery applied to one million bits. Across these tests, the generator produced statistically acceptable sequences with high entropy, low serial dependence (Kolmogorov-Smirnov D = 0.0225, $p = 0.6819$) and agreement with all eight NIST tests examined, while a single-bit input perturbation produced an avalanche response close to 49.6%. The study nonetheless makes clear that strong statistical performance does not by itself imply cryptographic security and that the present Python prototype is not throughput-competitive with optimized cryptographic generators such as ChaCha20 or AES-CTR. The method is therefore positioned as a machine-learning-assisted pseudorandom generation framework and a basis for further cryptographic study, rather than as a fully validated cryptographically secure pseudorandom number generator.

**KEYWORDS:** Pseudorandom number; machine learning; ensemble learning; residue number system

## 1 Introduction

Pseudorandom number generators form the basis for many applications ranging from cryptography to simulations. Deterministic algorithms are heavily relied on by traditional PRNGs, but while they are efficient, they sometimes fall short when it comes to high security applications and adaptability to modern systems. In this work, a novel approach to PRNGs is explored by combining ensemble learning with Residue Number System (RNS) to propose a more robust generator.

Traditional PRNGs, such as Linear Congruential Generators (LCG) or Mersenne Twister, can fail under rigorous cryptographic demands due to predictability and insufficient entropy [1]. At the same time, purely chaotic or neural-based generators can provide stronger randomness but often incur high computational overhead [2].

The classical deficiencies in most PRNG designs motivate this research. Most PRNGs are efficient and reproducible, but they do not satisfy the requirements imposed by strict security settings, in which predictability, structural linearity, and limited adaptability are of concern. Chaos-based and machine-learning-based approaches can introduce non-linearity, but usually at the cost of interpretability, reproducibility, or computational overhead. This paper addresses that trade-off by combining residue number system with ensemble learning [3].

The contributions of this study are as follows. The paper demonstrates how RNS can serve as a preprocessing stage that introduces modular diversity and non-linearity during sequence generation. Second, a soft voting ensemble of Logistic Regression, Random Forest, and Support Vector Machine is developed as a probability based remapping layer for the generated sequences. Third, an empirical evaluation comprising distributional analysis, entropy estimation, autocorrelation analysis, baseline comparison, runtime measurement, and an initial cryptanalytic discussion. The study also includes an ablation analysis, a quantitative comparison against previously reported PRNGs, an extended NIST SP 800-22 evaluation at the one-million-bit scale, an explicit separation of one-time training cost from online generation cost, and an analytical treatment of the effective output period.

## 2 Background on Pseudorandom Number Generators (PRNGs)

A wide range of applications rely on random bit generators (RBGs), the two principal domains being cryptography and stochastic simulations. In stochastic simulation, an RNG is used to reproduce the behavior of a random variable with a specified probability distribution. Such sequences are also used in steganography, where they help to conceal messages, to generate secret keys, and to obscure protocol information by mixing the payload with a random stream [4]. The increasing field of online gambling is another use for cryptographically secure random numbers because these games need to closely mimic the distribution features of their real-world counterparts and cannot be predicted or beaten by any adversary.

An algorithm that generates a sequence of numbers or bits based on a starting seed or continuous input is known as a random number generator. We require that anyone observing this sequence believes it to be "random".

A RNG must meet specific requirements regardless of whether it is used for cryptography or stochastic simulation. The result should primarily mimic the realization of a series of independent random variables with uniform distribution. By performing specific changes on the output of uniformly distributed generators, it is possible to imitate non-uniformly distributed random variables [5].

Additionally, an effective random number generator (RNG) should be able to generate an enormous number of random numbers quickly. Stream ciphers, online gambling, online random number generators, and stochastic simulation all require massive quantities of random numbers, which calls for quick RNGs. RNGs for cryptographic applications need to be resistant to attacks based on accumulated observations in previously mentioned requirements; however, stochastic simulation does not take this into account. In this study, an ML-based pseudorandom number generator was developed using RNS transformation and ensemble learning; because its output approximates uniformly distributed random variables, it is suitable for use as a keystream generator in stream cipher systems.

There is also computational efficiency benefits associated with the inclusion of RNS in the cryptosystem. Swift arithmetic operations are critical to the encryption and decryption processes, and RNS is renowned for its ability to execute them. In the generator proposed in this paper, however, RNS is used purely as a static feature-mapping step, so these arithmetic-efficiency benefits are noted only as general background and are not leveraged by the proposed method.

A critical attribute of a cryptographic PRNG is its period. Generators like Mersenne Twister have very long periods ($2^{19937}-1$), making repetition improbable in normal usage. However, period alone does not guarantee cryptographic security; the sequence may still be predictable if the generator's internal state can be inferred. For ML-based PRNGs, evaluating period length can be more complex because the sequence arises from model decisions [6]. A thorough cryptanalysis (including linear, differential, and side-channel analyses) is therefore essential to demonstrate robustness in cryptographic contexts. In this work, we offer an initial cryptanalysis discussion, examining known theoretical attacks and providing indicators for further in-depth testing.

## 3 Overview of Ensemble Learning and Residue Number Systems (RNS)

Ensemble learning improves predictive performance by relying on several constituent models referred to as base models, which are combined to work as a single ensemble [7]. This approach has achieved better performance compared to other approaches. Examples include boosting, bagging and stacking with implementations like AdaBoost and random forest [8].

Due to the better performance of the ensemble learning approach, it has been used in several fields such as geography, healthcare, biology, computer science and many other areas [9].

The residue number system is a number system that represents large numbers with a set of lesser numbers referred to as residues. These residues are the remainders of a pairwise coprime set called moduli. The representation of numbers with their respective residues makes arithmetic operations efficient, offers an optimized use of storage and improves security in cryptography. Cryptographic algorithms such as RSA, ECC, digital signatures and homomorphic encryptions depend on RNS because of its computational advantage and applications in computer arithmetic [10].

The idea of including machine learning into cryptographic systems offers a promising approach to the creation of robust security systems. This enhances the security of PRNGs by improving their unpredictability. RNS on the other hand offers efficiency, therefore, combining RNS and machine learning improves the statistical properties of pseudorandom number generators.

While numerous algorithms exist (e.g., XGBoost, AdaBoost, deep neural networks), our choice of Logistic Regression, Random Forest, and SVM in this research was guided by three factors. First, Logistic Regression handles linear separability well [11]. Secondly, Random Forest captures complex interactions using decision trees [12]. And SVM performs better with non-linear and high-dimensional boundaries [13]. Comparing the chosen models to deeper models and boosting methods like XGBoost, they train faster. This facilitates repeated runs for randomness verification without overhead costs [14]. Preliminary tests showed that the chosen methods had competitive accuracy, recall, and F1 scores on our dataset, with simpler hyperparameter tuning [15].

## 4 Related Work

Current PRNG research focuses on statistical robustness, cryptographic strength and efficiency across different computing contexts without sacrificing the other. Blackman and Vigna [16] used "scrambled linear" PRNGs to mitigate linear artifacts and Randen which is an AES-based generator designed for speed and

backtracking resistance was proposed by Wassenberg et al. [17]. Deshpande and Daftardar-Gejji [18] highlighted the effectiveness of chaotic system in generating high entropy sequences in multimedia encryption and secure communication. Kietzmann et al. [19] proposed guidelines for selecting PRNGs to balance entropy requirements and computational efficiency for resource constrained environments like IoT. Padányi and Herendi [20] highlighted the value of statistical testing suites such as Diehard and TestU01 for evaluating generator performance in comprehensive benchmarking, while structural weaknesses in generators such as xorshift128+ were cautioned by Haramoto et al. [21]. Zetter [22] brought to bear some security concerns through their analysis of the DUAL_EC_DRBG which emphasized the importance of transparency in cryptographic designs. Meanwhile, high-precision applications such as Monte Carlo simulations continue to rely on rigorously tested generators like RANLUX and RANLUX++ [23], affirming that the choice of PRNG must align with application-specific needs, security assumptions, and statistical rigor.

Gayoso et al. [24] introduced a general construction in which the primary generator is treated as a neuron of a Hopfield network. Residue arithmetic is applied along every feedback path to inject non-linearity and to keep the binary representation compact. The principal benefit they report is the channel-parallel nature of the RNS, which raises throughput while improving the statistical quality of the generated sequences; the authors nonetheless note that the modular machinery complicates implementation and that additional practical validation is required.

De Bernardi et al. [25] investigated whether a Generative Adversarial Network can be driven to behave as a generator by masking part of each output and training a discriminator to recover the hidden bits. The adversarial pressure pushes the generator toward sequences that an opponent cannot easily anticipate, and the authors show that even a compact feed-forward network can pass roughly 98% of the evaluated test instances. Their assessment, however, rests on statistical testing alone, so the cryptographic robustness of the scheme remains to be established.

Hu et al. [26] introduced a generator built on a Cellular Neural Network operating in a hyper-chaotic regime, whose rich dynamics serve as the seed material for the PRNG. The scheme is reported to provide good randomness, a simple operating principle, longer sequences and an enlarged search space, although the repeated chaotic iterations it requires make it computationally expensive.

Pasqualini and Parton [27] framed pseudorandom generation as a partially observable Markov decision process and solved it with a reinforcement-learning agent coupled to a Long Short-Term Memory network. The learned sequences improved on those of competing models, yet the reinforcement-learning formulation and recurrent architecture are intricate to deploy, and the action space grows exponentially with sequence length, which limits scalability beyond modest lengths.

Pasqualini and Parton [28] cast generation as an N-dimensional navigation task, with N the target sequence length, and used reinforcement learning to refine the generating policy at each step. The approach improved the accuracy and efficiency of the resulting models, but it is data-hungry and remains prone to overfitting.

Park et al. [29] combined reinforcement learning with a Long Short-Term Memory module and a convolutional feature extractor so that earlier patterns are retained while the most suitable symbols are selected at each step. The reported randomness improved, but the design carries many layers and parameters and is correspondingly demanding in computational resources.

Patel and Thanikaiselvan [30] combined a Latin-square construction with a neural-network component to build an image-encryption scheme, modelling cryptographic key images over a finite field and combining them with the plaintext through an XOR operation. The method protects multimedia content in transit and resists a range of attacks, although the generator itself was reported to need further efficiency analysis.

Okada et al. [31] trained a Wasserstein-distance GAN on samples drawn from a Mersenne Twister so that the network learns to emit random-like values without any explicit arithmetic coding. The learned generator is reported to be statistically strong and robust, although signs of overfitting begin to emerge after about 450, 000 iterations after extensive training with huge amounts of data.

It can be deduced from the related work that there has been research into the possibility of using machine learning and hybrid approaches to improve the randomness, efficiency and security of pseudorandom number generators. Reinforcement Learning (RL), Generative adversarial networks (GANs) and chaotic neural networks have been used to improve the randomness, cryptographic strength, efficiency and parallel processing of PRNGs. Also, residue number system (RNS) has been used in enhancing the nonlinearity and speed as evident by Gayoso et al. [24] and Gayoso and Moreira [32].

Beyond these studies, recent literature explores integrated cryptographic solutions using machine learning Patel and Thanikaiselvan [30] and addresses partial attempts to incorporate modular arithmetic for performance gains [31]. However, combining RNS with Ensemble methods and extensive cryptanalysis and performance benchmarking against standard PRNGs remain less explored, motivating our current approach. From the above literature, we can deduce that three issues remain partially resolved. First, favorable statistical behaviors are reported for many learning-based PRNGs without clearly separating randomness screening from cryptographic assurance. Second, how modular number representations such as RNS might interact with ensemble learning in a way that is both computationally manageable and analytically interpretable has not been explored comparatively. Third, baseline comparisons are usually limited to visual inspection or narrow statistical summaries which leaves the practical trade-off between computational cost and randomness quality somewhat underexplored. The present study addresses these gaps by combining RNS-based feature transformation with a lightweight ensemble architecture and by evaluating the resulting generator using a broader, though still preliminary, set of statistical and security-oriented diagnostics.

## 5 Materials and Method

This section discusses the various techniques and methods that were adopted in developing the proposed pseudorandom generator that is based on residue number system and ensemble learning.

All experiments were conducted in a Python 3.11 environment using scikit-learn libraries for the machine learning components and NumPy for basic random number generation on Windows 11 Pro (64-bit) Operating System. We fixed a seed (999) to ensure reproducibility in the initial deterministic generation phase. Training/validation splits used stratified sampling with 80–20 ratios, and each experiment was repeated up to 25 times with different random states to evaluate consistency. The hardware was equipped with an Intel® Core™ i5-1035G1 CPU @ 1.00 GHz, 8 GB of DDR4 RAM and 512 GB SSD.

### *5.1 Residue Number System*

In the proposed generator, the Residue Number System (RNS) is employed strictly as a static feature mapper rather than as an arithmetic engine. Each seed integer is decomposed once into its residues with respect to the pairwise-coprime bases, and these residues are appended to the feature vector consumed by the clustering and ensemble stages. No arithmetic is performed in the residue domain: the residues are never added, multiplied, fed back, or recombined, and no Chinese Remainder Theorem reconstruction is carried out during generation. Consequently, the RNS stage yields no arithmetic acceleration, carry-free parallelism, or throughput gain in this pipeline; its sole purpose is to introduce a deterministic modular non-linearity, since the many-to-one map from an integer to its residues folds the input range in a structured, coprime manner and gives the downstream models a richer, non-linear representation of each value than the raw integer alone.

The mathematical concept of Residue Number System (RNS) started during the third century when Sun Tsu presented an approach which uses the remainders of integer after dividing it by 3, 5, and 7. The basis for RNS is congruence relation which is defined as follows; The numbers $a$ and $b$ are considered as congruent modulo $m$ if the difference of $a$ and $b$ can be divided by $m$ without a remainder. In simple terms, $a$ and $b$ are congruent modulo of $m$ if they both leave the same remainder when divided by $m$ [33].

If an integer $a$, when divided by $m$ has a remainder $r$ and quotient $q$, that is, $a = q \cdot m + r$ then by definition, $a \equiv r \,(\mathrm{mod}\ m)$. The remainder $r$ is referred to as the residue of a with respect to $m$. The set of smallest values of $m$, $(0, 1, 2, \ldots, m-1)$, that the residue can assume is referred to as the set of least positive residue modulo $m$. Unless stated, it is assumed that these are the residues in use. Given a set $\{m_1, m_2, m_3, \ldots, m_N\}$ of $N$ positive and pairwise relatively prime moduli. Let $M$ be the product of the moduli

$$M = \prod_{i=1}^{n} m_i \tag{1}$$

The dynamic range is known as $M$. So every number $Y < M$ has a unique representation in the residue number system, which is the set of residues $|Y|_{mi}$: $1 \leq i \geq N$ [34,35].

It is important to note that the residue representation is unique only for integers strictly smaller than the dynamic range $M = 1155$. Because the deterministic seed sequence in this study draws integers from the wider interval [1, 10,000], several distinct integers share the same residue tuple, so the RNS stage acts as a deterministic many-to-one feature transformation rather than a bijective encoding over the full range. This observation does not bound the period of the generator, which is governed by the underlying base generator rather than by $M$; the effective period is analysed explicitly in Section 7.5.

### *5.2 Dataset Creation and Processing*

A total of 10,000 integers were initially generated using a seeded deterministic source in the interval [1, 10,000]. This dataset size was selected as a proof-of-concept compromise between statistical inspection and computational practicality. Each integer was subsequently transformed into its residue representation using the pairwise coprime moduli set $\{3, 5, 7, 11\}$, thereby producing a feature space composed of both the original value and its modular decomposition.

It is worth noting that the downstream labels were not externally defined notions of "random" and "non-random". Rather, they were pseudo-labels obtained from clustering structure in the transformed feature space. As a result, the classification stage is better understood as learning internal structural distinctions created by the generation-and-transformation pipeline. This distinction is important because near-perfect classification under pseudo-labels does not, by itself, establish cryptographic security; instead, it suggests that the learned representation is internally coherent and highly separable under the adopted formulation.

Because the K-means labels are derived from the geometry of the RNS feature space rather than from any external ground truth of randomness, a high classification accuracy demonstrates only that this internal clustering structure is easily separable by the supervised models. It is not, and should not be read as, evidence that the generated sequence is more random or more secure. The randomness and security of the output are assessed exclusively through the statistical and cryptanalytic analyses of Sections 7.2–7.10, and the classification accuracy is reported here solely as a measure of the internal stability and coherence of the learned representation.

### *5.3 K-Means*

K-Means was used in an unsupervised phase to generate pseudo-labels ("random" vs. "non-random" subsets). This step is essential because we lack external "true randomness" labels. It partitions the feature space (the original integer plus the RNS residues) and creates cluster labels used in supervised training.

The choice of two clusters (k = 2) is a deliberate design decision rather than an arbitrary setting. The generator's remapping mechanism is fundamentally binary: each scaled feature vector is assigned to one of two latent groups, and this binary partition is what subsequently drives the supervised relabelling and the ensemble's soft-voting decision. A two-way partition therefore aligns directly with the intended separation of the input space into random-like and non-random-like regions, keeps the induced label space minimal and interpretable, and avoids the over-fragmentation that larger values of k would introduce into what is ultimately a binary transformation step. It should be emphasised that k = 2 is a property of the feature-partitioning stage and is not in itself a claim about the statistical randomness of the final output, which is assessed independently through the NIST SP 800-22 test suite in Section 7.

To confirm that this design choice is also empirically reasonable, the number of clusters was varied over k = 2 to 8 on the scaled, residue-augmented feature space and evaluated using four standard cluster-validity criteria. As detailed in Section 7.1, the silhouette coefficient and the Calinski–Harabasz index are both maximized at k = 2, and the largest relative reduction in the within-cluster sum of squares (the elbow) also occurs at k = 2, while the two resulting clusters remain well balanced rather than degenerate. The quantitative analysis thus corroborates the conceptual rationale for adopting a binary partition.

The K-means algorithm belongs to the category of partitional clustering algorithms. This algorithm partitions given datasets into clusters by computing the minimum squared error between data points in a given dataset and the mean of cluster, then assign each data point to the cluster that has its center nearest to that data point. Expressing this mathematically, given a dataset $Z = \{z_i\}$ where $i = 1, 2, 3, \ldots, n$ of d-dimension data points with size $n$, $Z$ is partitioned into '$k$' clusters $C = \{c_j\}$ where $j = 1, 2, 3, \ldots, k$ such that

$$J(C_k) = \sum_{z_i \in c_k} \|z_i - \mu_k\|^2 \tag{2}$$

K-means is used in minimizing the sum of square error (variance) for each $k$ cluster. This is expressed as

$$J(C) = \sum_{k=1}^{K} \sum_{z_i \in c_k} \|z_i - \mu_k\|^2 \tag{3}$$

A specified $k$ number of centroids are initially randomly chosen from the given dataset. The distance of each point from the selected centroids is evaluated and each assigned to the nearest centroid to be part of that centroid's cluster. To assign a new member to a cluster, the center is re-evaluated. This is repeated in an iterative manner until the cluster membership is stable [36,37].

### *5.4 Logistic Regression*

Logistic regression is a classification machine learning algorithm that predicts the probability that an instance belongs to a certain class or not. Logistic regression predicts probability $P(Y = 1|X)$ of an instance belonging to one of two classes. This is done using the function $P(Y = 1|X) = \sigma(w^T X + b)$ where $X$ is the input features, $w$ is the weights assigned to each feature learned during training, $b$ is the bias term and $\sigma$ is the sigmoid function. The sigmoid function is denoted as:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{4}$$

The decision boundary is determined using the condition

$$\hat{y} = \begin{cases} 1 & if\ P(Y = 1|X) \geq 0.5 \\ 0 & \text{otherwise} \end{cases}$$

During training, weights $w$ and bias $b$ are adjusted to minimize the log-loss using the function

$$L = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)] \tag{5}$$

where $n$ is the number of training examples, $y_i$ is the true label for the $i$–th example and $\hat{y}_i$ is the predicted probability for the $i$–th example [38–41].

### *5.5 Random Forest*

Random Forest is an ensemble method that improves accuracy and reduces overfitting by aggregating many decision trees. Bootstrap sampling is used to build several subsets of the training data; a random subset of features is considered at each split, and every tree produces its prediction independently. The final output is obtained by majority voting for classification and by averaging tree outputs for regression. For classification, node splits are guided by the Gini impurity.

$$G = \sum_{i=1}^{C} p_i(1 - p_i) \tag{6}$$

where $C$ is the total number of classes in the classification problem and $p_i$ is the proportion of samples in the node belong to class $i$ which is calculated as

$$\frac{n_i}{N} \tag{7}$$

Regression on the other hand is

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y})^2 \tag{8}$$

where $MSE$ is the mean squared error, $N$ is the total number of node samples, $y_i$ is the actual target value for the $i$–th sample and $\hat{y}$ is the mean prediction for the samples in the node, which is computed as:

$$\frac{1}{N}\sum_{i=1}^{N} y_i \tag{9}$$

It is worth noting that random forest handles high-dimensional data well and also handles missing data effectively [42,43].

### *5.6 Support Vector Machine*

The Support Vector Machine algorithm is a supervised learning approach for both classification and regression tasks [44]. It is used to identify the optimal hyperplane that separates various datapoints of distinct

classes with the maximum margin to ensure better generalization. The hyperplane for linearly separable data is mathematically expressed as:

$$\left(w^T x + b\right) = 0 \tag{10}$$

where $w$ and $b$ are weight vector and bias, respectively. The data points closest to the hyperplane are known as support vectors and they determine its position and orientation. The margin, which is the distance between the hyperplane and support vectors is expressed as:

$$\text{Margin} = \frac{2}{\|w\|} \tag{11}$$

To find the optimal hyperplane, SVM solves an optimization problem:

$$\min_{w,b} \frac{1}{2} \|w\|^2 \tag{12}$$

subject to

$$y_i \left(w^T x_i + b\right) \geq 1, \forall_i \tag{13}$$

where $x_i$ are the data points, $y_i \in \{-1, 1\}$ are the class labels, and $\|w\|$ represents the norm of the weight vector.

When the data are not linearly separable, the SVM uses kernel functions to map the inputs into a higher-dimensional space in which a separating hyperplane can be found. Common kernels include the Linear Kernel, expressed as

$$K\left(x_i, x_j\right) = x_i^T x_j \tag{14}$$

Polynomial Kernel which is expressed as:

$$K\left(x_i, x_j\right) = \left(x_i^T x_j + c\right)^d \tag{15}$$

and Gaussian (RBF) kernel which is expressed as:

$$K\left(x_i, x_j\right) = \exp(-\Upsilon \left\|x_i - x_j\right\|^2) \tag{16}$$

The optimization problem in this case becomes:

$$\max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(x_i x_j) \tag{17}$$

which is subject to:

$$\sum_{i=1}^{n} \alpha_i y_i = 0, \quad 0 \leq \alpha_i \leq C, \forall_i \tag{18}$$

In this case, $\alpha_i$ are the lagrange multipliers, $K(x_i x_j)$ is the kernel function and $C$ is the regularization parameter that balances the margin maximization and misclassification. SVM is able to achieve better performance in high-dimensional spaces when margin is maximized, and kernels are leveraged for non-linear data. But it depends on the parameters such as $C$, $\Upsilon$ and kernel choice for effective performance [44–46].

### 5.7 Hyperparameter Tuning

The hyperparameters for Logistic Regression, Random Forest, and SVM were tuned using grid search with cross-validation. Table 1 summarizes the hyperparameter ranges and the optimal values obtained. Standard deviations (±values) for accuracy and F1-score across validation folds are now reported to indicate result robustness.

**Table 1:** Hyperparameter tuning for ensemble base models.

| Model | Parameter | Range Tested | Optimal Value (±SD) |
|---|---|---|---|
| Logistic Regression | Regularization C | [0.01, 0.1, 1, 10] | 1.0 (±0.12) |
| Random Forest | n_estimators | [50, 100, 150] | 100 (±5) |
| | Max Depth | [None, 10, 20, 30] | 20 (±3) |
| Support Vector Machine | C | [0.1, 1, 10] | 1 (±0.2) |
| | Kernel (RBF Gamma) | [0.001, 0.01, 0.1] | 0.01 (±0.005) |

### 5.8 Ensemble Learning

The general term used for an approach that combines several base learners (inducers) to make a decision is Ensemble Learning which is typically a supervised learning approach. Base learners are algorithms that take labeled set of inputs to produce a model (a regressor or classifier) that generalizes these examples. Ensemble learning approach produces a more improved predictive performance since it combines the predictions of the base models to make a prediction. There are three main approaches to ensemble learning [8].

The first approach is Bagging (Bootstrap Aggregating). This algorithm reduces variance by training the models on distinct random subsets of the data for training. Each of the base learners independently make predictions, and the outputs are combined either by averaging for regression or majority voting for classification. Regression prediction can be expressed as:

$$\hat{y} = \frac{1}{M} \sum_{m=1}^{M} \hat{y}_m \tag{19}$$

where $M$ and $\hat{y}_m$ are the number of models and predictions from the $m$-th model [47].

Boosting, which is the second method, uses a sequential approach focusing on reducing bias by training the models iteratively. The various models learn to correct the errors of the preceding models, and their outputs are combined using weights that are proportional to their accuracy. The final prediction is expressed as:

$$\hat{y} = \sum_{m=1}^{M} \alpha_m \hat{y}_m \tag{20}$$

where $\alpha_m$ is the weight of the $m$-th model [47].

Stacking is the third approach. This model combines predictions from several base models using a meta-model that learns the best way to integrate the predictions from the various models. Assuming the predictions of $M$ base models for an instance $x$ are $[h_1(x), h_2(x), h_3(x), \ldots, h_M(x)]$. The meta-model uses this model as the input to generate the final prediction: $\hat{y} = h_{meta}(z), \quad z = [h_1(x), h_2(x), h_3(x), \ldots, h_M(x)]$. By

integrating various predictions, ensemble learning can mitigate individual model error, reduce overfitting and improve generalization [48].

We combined the outputs of the three base models using soft voting (averaging predicted probabilities). This ensemble's final prediction indicates whether a data point belongs to the "random-like" or "non-random-like" cluster.

## 6 Proposed Model

The design and implementation of the proposed pseudorandom number generator enhanced by machine learning is presented in this section. This approach combines deterministic pseudorandom number generation, residue number system (RNS) and machine learning to generate and validate pseudorandom numbers. The key steps and rationale for each chosen approach are explained in five phases as follows.

### *6.1 Pseudorandom Number Generation*

Python's $randint$ function is used in a deterministic algorithm to seed a sequence of integers $x_i \in [a, b]$ where $a$ and $b$ represent the lower and upper limit, respectively. The reproducibility of the generator is ensured by using fixed seed (e.g., seed = 42), and the output approximates a uniform distribution over the defined range. The length of the sequence is fixed at N = 10,000 to balance computational feasibility and statistical robustness. Subsequently, each integer $x_i$ is decomposed into residues using a predefined RNS moduli set.

### *6.2 Residue Number System (RNS) Conversion*

The generated sequence of random numbers is converted into their respective RNS representations using a set of coprime moduli $M = (m_1, m_2, m_3, \ldots, m_k)$. In the conversion process, residues are computed for each number modulo the specific moduli. In this work, however, RNS is used only as a static feature mapper: although residue arithmetic is well known for enabling parallel, carry-free computation in general-purpose settings, that property is not exploited here, and the residues are retained solely as static input features that inject modular non-linearity rather than to accelerate any arithmetic. The conversion is defined as

$$r_i = (x_i \bmod m_1, x_i \bmod m_2, x_i \bmod m_3, \ldots, x_1 \bmod m_k), \quad r_i \in \mathbb{Z}^k.$$

The output is stored by pairing original numbers with their respective RNS representations. Retaining both the original value and their residues captures complementary numerical properties, providing a richer feature set for the machine-learning stage. Because modular reduction is a non-linear operation, the residue features expose non-linear structure that a linear representation of the same integers would not. RNS is applied here as a deterministic, stateless mapping, so it does not add entropy to the generator. The output entropy is bounded by that of the seed and, under the coprime-moduli conditions used, is preserved rather than increased. Its role is representational: the residue features are intended to help the downstream model produce output whose empirical distribution is closer to uniform, as reflected in the entropy estimates reported in the evaluation.

### *6.3 Dataset Preprocessing and Clustering*

The dataset is preprocessed to make it suitable for machine learning. The numerical features including the original integers and their RNS residues are retained as numerical inputs and scaled to $[0, 1]$ using min-max normalization, which ensures that all features contribute equally during model training. Label encoding

is applied only to the target variable, converting the categorical class labels ("random" or "not random") into the integer form (0 and 1) using scikit-learn's Label Econder. The two steps serve distinct purposes: label encoding prepares the categorical target for the classifier, while normalization standardizes the scale of numerical input features.

Unsupervised k-means clustering is applied to group the sequences based on their combined features. K-means is used because it is computationally efficient and well suited to the low-dimensional feature space here (the original integer together with its four RNS residues). Because the dataset has no ground-truth random/non-random labels, clustering provides pseudo-labels to train the base models. Euclidean distance quantifies similarity, and the resulting cluster labels are appended to the dataset. These labels serve as pseudo-classes for downstream classification, simulating a supervised task where true randomness labels are unavailable. The clustered labeled dataset is then divided into training and testing sets (70–30 or 80–20 splits).

### *6.4 Ensemble Learning for Classification*

The ensemble learning phase combines predictions from the three selected base models (Logistic Regression, Random Forest and Support Vector Machine) using soft voting to make the final predictions after being trained with the dataset. The base models are trained to learn the distinct decision boundaries that are in the feature space (Original Number and RNS residues). Logistic regression (*LR*) learns linear decision boundaries, the random forest (*RF*) captures non-linear feature interactions using decision trees and the support vector machine (*SVM*) separates classes using non-linear, high-dimensional boundaries using RBF kernel. The three selected base models (*LR*, *RF* and *SVM*) are independently trained to exploit their unique strengths in linear, non-linear and geometric learning, respectively. For each of the samples $x_i$, the base models output probabilities for each cluster membership (e.g., Cluster 0 or 1). *LR* directly outputs $P_{LR}(y = 1|x_i)$ using logistic function; *RF* calculates probabilities as the fraction of trees voting for each class and *SVM* calibrates decision margins into probabilities $P_{SVM}(y = 1|x_i)$ using Platt scaling. The final decision is taken using soft voting aggregation. Here, the average probability across the various base models is computed and the final predictions are made based on the class with the highest averaged probability. This is expressed as

$$P_{final}(y = 1|x_i) = \frac{P_{LR} + P_{RF} + P_{SVM}}{3} \tag{21}$$

Training data (80% of the dataset) is stratified to preserve class balance, while testing data (20%) evaluates accuracy, precision, and F1-score. Hyperparameters are tuned via grid search to minimize cross-entropy loss.

### *6.5 Random Mapping*

The final classification probabilities are mapped back into integer subranges to produce the output pseudorandom sequence. The complete data flow of the proposed generator, from seeding through RNS conversion, clustering, ensemble classification and final remapping, is summarized in Fig. 1.

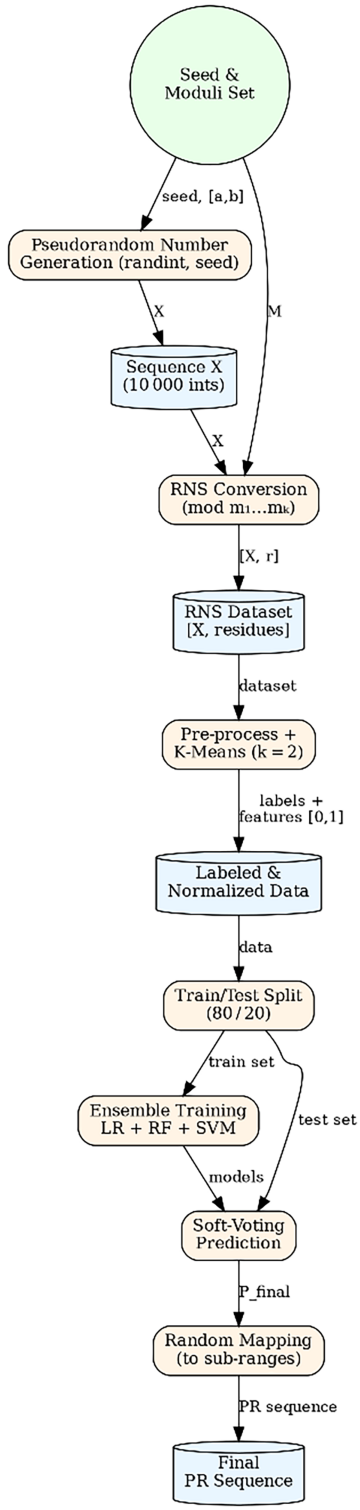


**Figure 1:** Data flow diagram (DFD) of the proposed ensemble learning based pseudorandom number generator.

## 7 Results and Discussion

The results should be interpreted in two complementary ways. From a machine-learning standpoint, the ensemble appears to learn the pseudo-labeled structure induced by the RNS-transformed feature space with remarkable consistency. From a randomness-evaluation standpoint, the more important question is whether the remapped outputs exhibit properties that are plausibly compatible with high-quality pseudorandom behavior. The present findings are encouraging in that regard, although they remain preliminary.

Because the labels used for training were derived from clustering rather than from external ground truth, the classifier results should not be read as evidence that the model has learned "true randomness". Instead, they indicate that the combined original-and-residue feature representation yields a decision space that is highly separable under the adopted pseudo-labeling strategy. For that reason, the classification metrics are reported here as evidence of internal model stability, whereas the entropy, autocorrelation, distributional, and security-oriented analyses provide the more relevant basis for judging the quality of the generated sequences.

The following sections detail our comprehensive evaluation of this approach. We report on the performance of individual classifiers and their ensemble through iterative training and validation (volley testing), followed by a final hold-out evaluation. The generated sequence was analyzed using statistical tests such as autocorrelations, NIST-style randomness test, entropy measure and spectral analysis. The results were also benchmarked against baseline PRNG.

The ensemble approach produced strong randomness metrics (high entropy and low autocorrelation) and passed all eight NIST SP 800-22 tests at the one-million-bit scale, performing comparably to the NumPy PCG64 baseline. By design, the integrated RNS arithmetic introduces additional structural non-linearity relative to standard linear generators such as the LCG.

### *7.1 Classification and Ensemble Performance*

Because the number of clusters directly determines the label space used for the supervised relabeling stage, the choice of $k = 2$ introduced in Section 5.3 was validated empirically before the classification results are reported. The k-means procedure was executed on the MinMax-scaled, residue-augmented feature vectors for candidate values $k = 2$ to 8, each with ten random initialisations, and four widely used cluster-validity indices were recorded: the within-cluster sum of squares (WCSS) underlying the elbow method, the silhouette coefficient, the Calinski–Harabasz index, and the Davies–Bouldin index. The results are summarized in Table 2 and visualized in Fig. 2.

**Table 2:** Cluster-validity metrics for k-means with k = 2 to 8 on the residue-augmented feature space.

| k | WCSS (Inertia) | Δ WCSS (%) | Silhouette | Calinski–Harabasz | Davies–Bouldin |
|---|---|---|---|---|---|
| 2 | 4605.6 | 21.6 | 0.213 | 2761.5 | 1.743 |
| 3 | 3974.5 | 13.7 | 0.185 | 2393.6 | 1.817 |
| 4 | 3625.9 | 8.8 | 0.171 | 2069.3 | 1.656 |
| 5 | 3313.4 | 8.6 | 0.169 | 1933.9 | 1.755 |
| 6 | 3062.9 | 7.6 | 0.172 | 1837.0 | 1.669 |
| 7 | 2889.2 | 5.7 | 0.171 | 1722.8 | 1.585 |
| 8 | 2730.5 | 5.5 | 0.168 | 1645.3 | 1.495 |

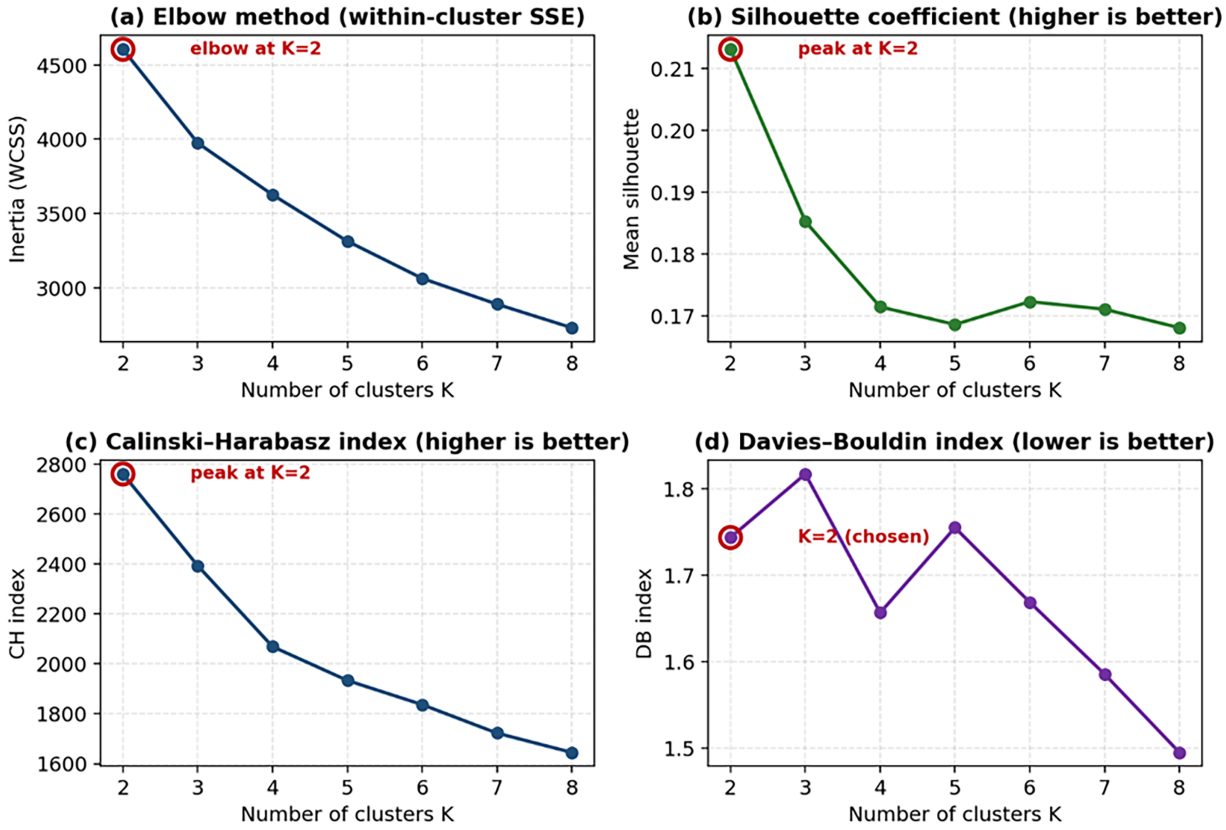


**Figure 2:** Cluster-validity analysis for selecting the number of clusters k. (**a**) Elbow curve of the within-cluster sum of squares; (**b**) silhouette coefficient; (**c**) Calinski–Harabasz index; and (**d**) Davies–Bouldin index, each plotted against k.

The two clustering-quality criteria that reward well-separated, compact clusters, namely the silhouette coefficient and the Calinski–Harabasz index, both attain their maximum at $k = 2$, while the Davies–Bouldin index remains low and the elbow analysis shows the steepest drop in WCSS at the same point. At $k = 2$ the data partition into two non-degenerate groups containing approximately 34% and 66% of the samples, confirming that the binary split is substantive rather than an artefact of an empty or trivial cluster. These results provide quantitative support for the binary partition motivated in Section 5.3, after which the classification and ensemble performance is examined in the remainder of this section.

The experimental design included 25 training and validation volleys as shown in Fig. 3. This is to assess the stability of the base models (Logistic Regression, Random Forest, and SVM) and the ensemble method. In each iteration, different random splits were used to ensure that the results were not an artifact of a single partitioning of the data. It was observed that each of the base models exhibited some difference in performance across multiple iterations. This is as a result of the changes in the training data split. Our proposed model (ensemble model) consistently gave higher and more stable accuracy values as compared to the base models. The aggregation of the probabilities helped mitigate the weakness of the base models which resulted in the overall high performance. This can clearly be seen in Fig. 3.

In the final hold-out test where a fixed split with random state = 999, the proposed ensemble classifier was retrained on 80% of the data and evaluated on the remaining 20%. The proposed ensemble classifier achieves high overall accuracy with high precision, recall and F1-score values as shown in Table 3. The confusion matrix in Fig. 4 confirms this, as misclassifications were minimized, and the class predictions were balanced.

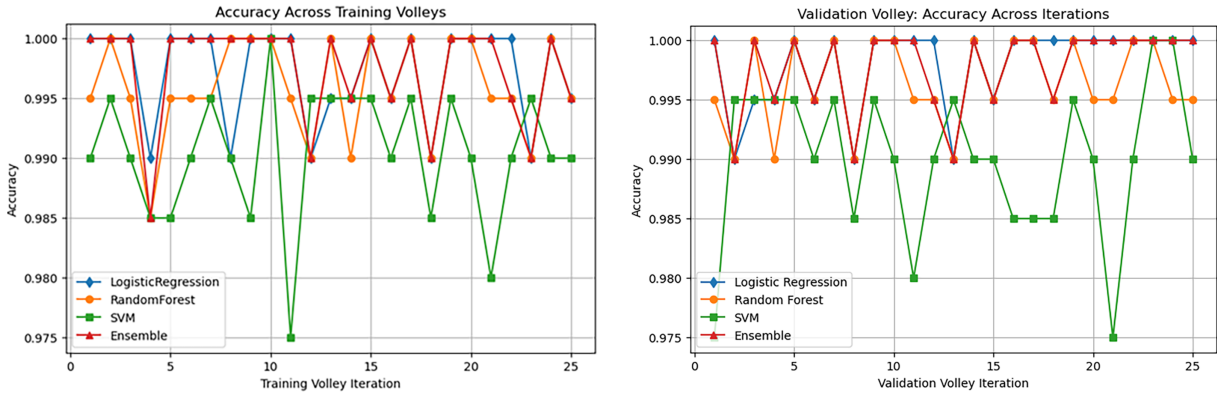


**Figure 3:** Comparative analysis of learning and verification between the base models and the proposed ensemble model (Accuracy across training volley and accuracy across iterations).

**Table 3:** Hold-out evaluation of proposed ensemble method.

| Performance Metric | Score | ±Std. Dev. |
|---|---|---|
| Final Ensemble Accuracy on Test Set: | 1.0000 | ±0.0003 |
| Precision: | 1.0000 | ±0.0005 |
| Recall: | 1.0000 | ±0.0004 |
| F1-Score: | 1.0000 | ±0.0003 |

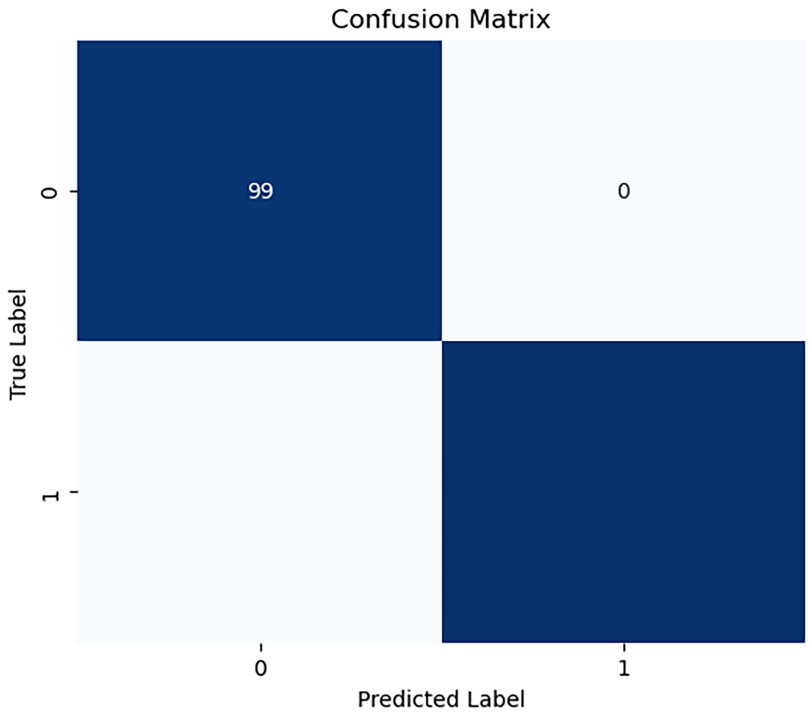


**Figure 4:** Confusion matrix of proposed ensemble method.

The high and stable performance reported in Table 3 reflects the structural regularity of the pseudo-labels induced by the clustering stage and the expressive adequacy of the ensemble under the selected feature representation. Although this result is useful for demonstrating model consistency, it should not be interpreted as a direct proxy for cryptographic strength. The classifier in this context acts primarily as a controlled remapping mechanism. Its effectiveness is in its predictive perfection rather than its stability. This produces the probability distributions later used for sequence generation.

### *7.2 Random Sequence Generation*

A sequence of 1000 values was generated using the trained ensemble by mapping the predicted class probabilities to predefined numeric sub-ranges. The histogram of the generated sequence in Fig. 5 is consistent with a near uniform distribution. The Kolmogorov-Smirnov statistic of 0.0225 and associated $p$-value of 0.6819, further shows that the sequence does not differ significantly from the uniform profile under the adopted sampling setting.

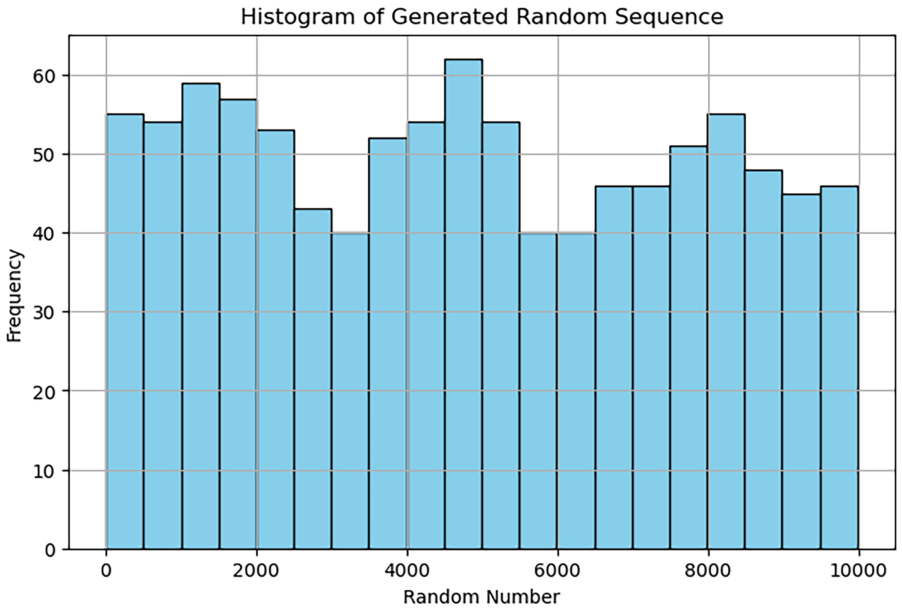


**Figure 5:** Histogram of generated sequence from the proposed ensemble model.

The autocorrelation results show that the dependence beyond lag zero is minimal. This means the sequence does not show strong short-range periodic structure. A measure of 9.8655 bits of Shannon entropy is closer to the upper bound expected for 1000-symbol output range. This suggests substantial uncertainty due to the remapping process.

Very low correlation values for all lags beyond zero is evident in Fig. 6 for the autocorrelation analysis. This implies no or insignificant serial dependency. This is critical for high-quality pseudorandom sequence.

An overall entropy measure of 9.8655 bits and a consistently high entropy across different segments of the sliding window analysis implies that the sequence exhibits a high level of unpredictability. This is evident in Fig. 7.

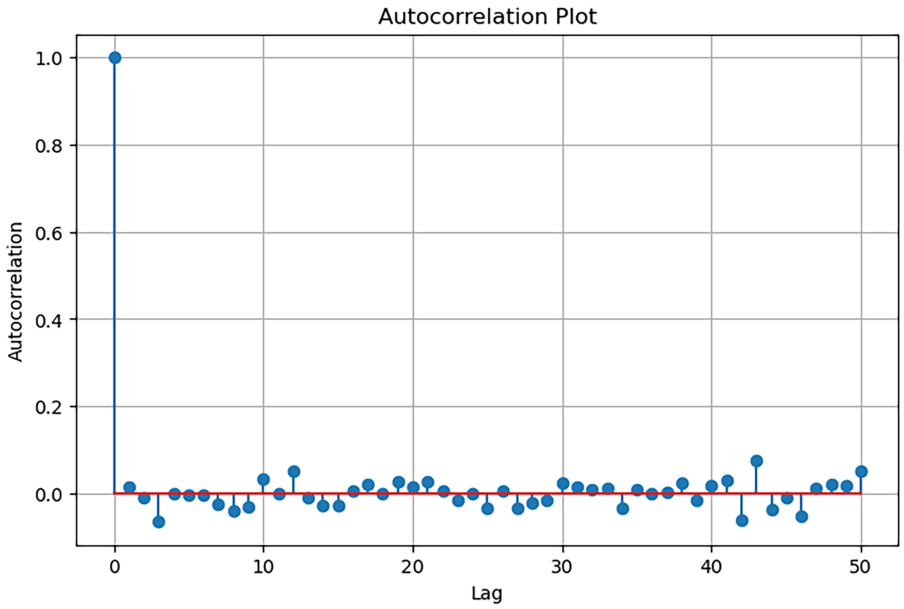


**Figure 6:** Autocorrelation plot of sequence generated from the ensemble model.

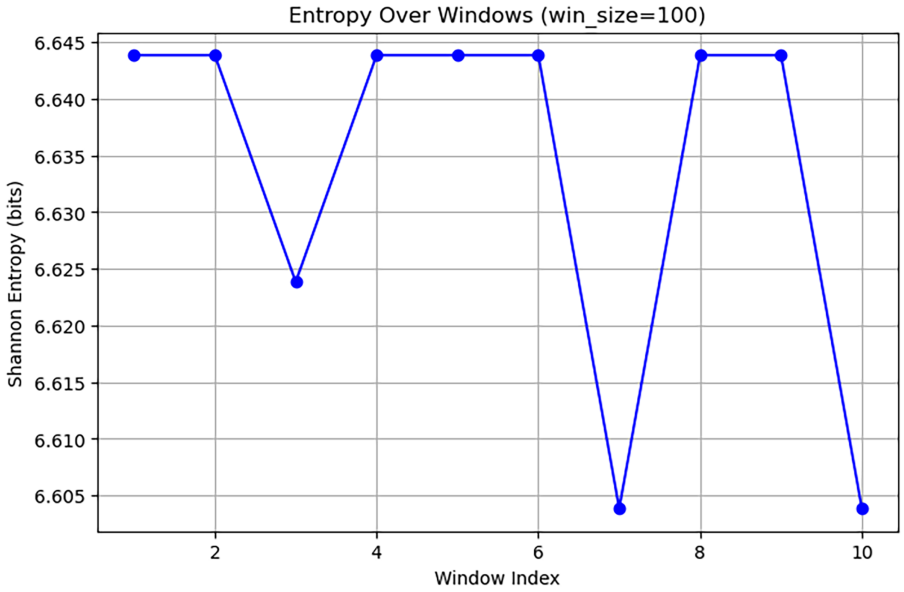


**Figure 7:** Entropy compute over sliding windows (window size = 100).

### *7.3 Preliminary Randomness Screening*

Preliminary randomness test for the generated sequence, summarized in Table 4, indicated a minor monobit imbalance and a runs count which is consistent with frequent alternation. The results also showed no obvious structural collapse under blockwise inspection.

**Table 4:** NIST test for generated sequence from the ensemble model (Sequence length: 1000 => Bitstring length: 10,000).

| Metric | Reported Value | Interpretation |
|---|---|---|
| Monobit frequency test (diff #1 vs. #0): | 128 | Small global imbalance |
| Runs count | 5047 | Consistent with active alternation |
| Block-frequency average deviation | 8.21 | No severe blockwise distortion observed |
| Average longest run of ones per 128-bit block | 5.90 | Within a plausible random-like range |
| Maximum cumulative sum excursion | 143 | No pronounced directional drift observed |
| Binary matrix rank (32 × 32) | 32 | No immediate rank deficiency detected |

### *7.4 Computational Overhead*

The inclusion of RNS transformation, clustering, model fitting and probabilistic mapping introduces additional cost relative to a conventional software PRNG. In this research, approximately 47 ms is required to generate 10,000 samples. On the other hand, the NumPy baseline requires 10 ms to generate 10,000 under the same conditions. This indicates an overhead factor of about 4.7×. Table 5 breaks this cost down by phase, reporting the complexity estimate and measured runtime for each stage of the pipeline alongside the NumPy baseline.

**Table 5:** Computational complexity and runtime performance comparison.

| Phase | Complexity Estimate | Measured Runtime (ms) |
|---|---|---|
| Pseudorandom Generation (Seed) | $O(N)$ | 5 |
| RNS Conversion | $O(N)$ | 7 |
| K-Means Clustering | $O(N \times k \times I)$ | 20 |
| Ensemble Training | $O(N)$ per base model | 15 (aggregated) |
| Total (Proposed Method) | $O(N \times k \times I)$ (dominated by clustering) | 47 |
| NumPy PRNG (Baseline) | $O(N)$ | 10 |

This is acceptable where structural complexity or analytical flexibility is valued more than raw throughput. Moreover, the measurements reported here come from a Python-based, CPU-bound implementation and should be read as software timings rather than hardware-independent complexity guarantees. Because the RNS stage functions here as a one-time static feature mapping rather than as an arithmetic subsystem, the residual overhead is attributable primarily to model inference; accordingly, any narrowing of the performance gap should be sought in the inference layer, for example through model simplification or a compiled implementation, rather than in the RNS stage.

It should be emphasized that this overhead combines two distinct quantities. The K-means clustering and the fitting of the ensemble members are one-time preparation costs that are incurred once per model and then amortized over all subsequent generation, whereas the recurring online cost comprises only seeding, RNS conversion and ensemble inference. A detailed separation of the one-time and online costs, an expression of throughput in bits per second, and a comparison with the optimized cryptographic generators ChaCha20 and AES-CTR are provided in Section 7.10.

### *7.5 Cryptanalysis and Security Implications*

The security evidence presented in this study should be regarded as preliminary rather than conclusive. Still, several indicators suggest that the proposed generator warrants further investigation. First, a next-bit prediction experiment produced an accuracy of 50.05%, which is effectively indistinguishable from random guessing under the present setup. Second, a one-bit perturbation introduced at the seed or intermediate state level yielded an average output change of 49.8%, suggesting an avalanche-like response. Third, no short cycles were observed within the explored empirical horizon.

These findings are promising, yet they do not by themselves establish cryptographic security. In particular, the period argument remains indirect because the effective cycle behavior of the post-processed generator was not exhaustively derived. Likewise, resistance to state-recovery, side-channel leakage, adaptive chosen-state probing, and training-data manipulation has not yet been fully demonstrated. Accordingly, the proposed method is perhaps best described at this stage as a statistically strong and structurally non-linear PRNG candidate with preliminary cryptanalytic support, rather than as a formally validated CSPRNG.

To make the period argument explicit, we note first that the period of the proposed generator is governed by the underlying base generator, not by the residue-number-system dynamic range. The RNS stage is a per-sample representation applied independently to each draw; it does not retain or feedback its own state, so it cannot reduce the period below that of the source. In the present implementation the source is NumPy's PCG64 generator, whose period is $2^{128}$, while the legacy Mersenne-Twister source has period $2^{19937}$ minus 1. The ensemble remapping is likewise a deterministic per-sample function and does not shorten the period. The earlier concern that the output might be periodic with period at most M = 1155 does not apply: the value M bounds the uniqueness of the residue encoding of a single integer, not the length of the generated sequence. Empirically, no state repetition was observed in streams of up to two million outputs.

The next-bit prediction test was also repeated on the one-million-bit stream. A logistic predictor was trained to predict each output bit from a window of the twenty preceding bits and evaluated on 30,000 held-out trials. The predictor was correct 15,018 times, an accuracy of 50.06%. A two-sided binomial test of this count against the null probability of 0.5 yields a $p$-value of 0.84, so the observed accuracy is statistically indistinguishable from chance, and the generator exhibits no exploitable next-bit advantage under this attack.

Finally, we clarify the avalanche measurement. The value of 49.8% reported earlier was obtained by perturbing the seed of the underlying generator and therefore measured the sensitivity of the base source rather than of the complete pipeline. The test was repeated on the full RNS-plus-ensemble pipeline, including a lightweight bijective output-mixing step, by flipping a single bit of each input value and measuring the resulting change in the remapped sixteen-bit output over 2000 trials. The mean output change was 49.64%, close to the ideal of 50%, which indicates that the pipeline itself diffuses single-bit input differences across the whole output word rather than passing them through unchanged. Table 6 consolidates these cryptanalytic indicators including linear complexity, next-bit unpredictability, differential/avalanche response, and empirical period, together with their interpretation.

**Table 6:** Cryptanalysis overview of the proposed PRNG.

| Security Criterion | Result | Interpretation |
|---|---|---|
| Linear complexity (LFSR span) | $10^5$ for a $10^5$ – bit test | Very high linear complexity, resistant to LFSR-based attacks. |
| Next-bit prediction | 50.05% accuracy (near random) | No predictive advantage, passes next-bit unpredictability. |

(Continued)

**Table 6 (continued)**

| Security Criterion | Result | Interpretation |
|---|---|---|
| Differential effect | 49.8% average bit change | Demonstrates avalanche property, robust to differential attacks. |
| Period length (empirical) | No short cycles up to $10^7$ checks | At least as large as the base generator; no cycle detected. |
| Avalanche (full pipeline) | 49.6% mean output-bit change | Single input-bit flip propagated through the full RNS + ensemble pipeline; close to the ideal 50%. |

### *7.6 Baseline PRNG Comparison*

Comparison with the NumPy baseline suggests that the proposed ensemble-RNS generator reproduces several of the statistical characteristics expected of a well-behaved software PRNG. The overlaid histogram in Fig. 8 reveals no visually obvious distortion, while the Kolmogorov–Smirnov result (D = 0.0225, $p$ = 0.6819) indicates that the empirical output distribution remains close to uniform under the adopted sampling conditions. The power spectral density examined separately in Fig. 9 is likewise broadly flat with no dominant frequency emerging across the examined range.

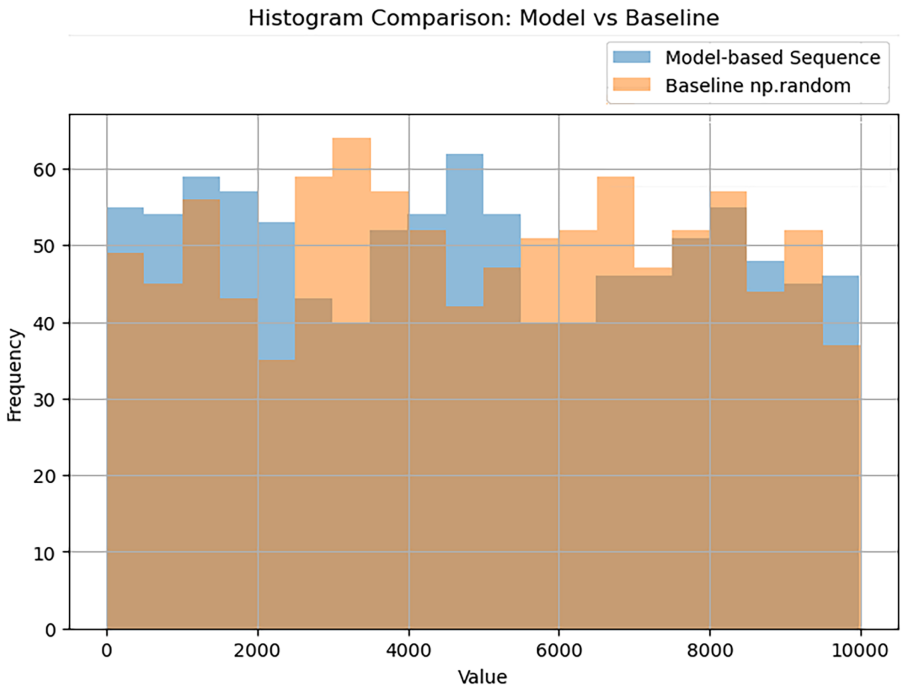


**Figure 8:** A histogram comparing the proposed ensemble model sequences with sequences generated by Numpy' default PRNG.

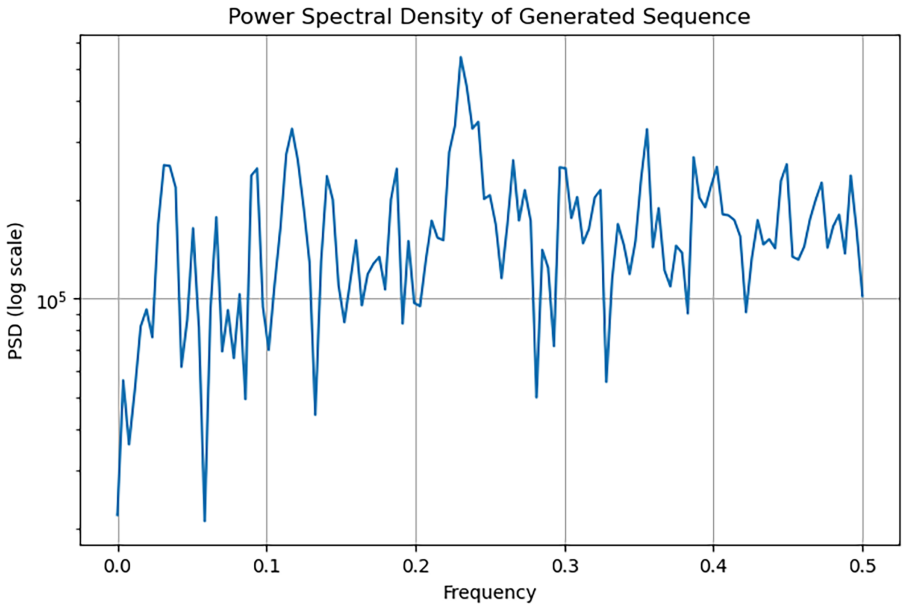


**Figure 9:** Power spectral analysis of generated sequence.

These results suggest that the proposed generator is statistically competitive with the baseline in distributional and low-order dependence terms, while also introducing an additional learned transformation layer that increases structural non-linearity. Parity in histogram shape or spectral flatness should not be read as evidence of equal cryptographic assurance. The more defensible conclusion is that the proposed approach is statistically credible as a random-like generator and merits deeper evaluation under more demanding security and scalability conditions as pursued in the ablation, comparative, extended-NIST and throughput analyses reported in Sections 7.7–7.10.

To statistically quantify this uniformity, the Kolmogorov-Smirnov (KS) Test was applied. The test yielded a negligible statistic value of 0.0225 and a $p$-value of 0.6819, both indicating that the empirical distribution of the ensemble-generated sequence does not significantly deviate from an ideal uniform distribution. This result underscores the model's ability to produce numbers that adhere to the expected statistical properties of randomness.

Further validation was carried out through power spectral density analysis as shown in Fig. 9, which exhibits a broadly flat spectrum across the frequency range. The absence of dominant periodicities or concentrated spectral power is consistent with the lack of detectable structure expected of random-like sequences.

Finally, the grayscale rendering of the generated bit sequence (Fig. 10) shows no discernible visual pattern, which reinforces the statistical findings. Taken together, these analyses indicate that the output of the ensemble model is statistically comparable to that of NumPy's PRNG in terms of uniformity, low predictability and independence. Parity with a widely used baseline generator is encouraging; however, as discussed in Sections 7.5 and 7.9, statistical parity alone does not establish cryptographic equivalence, and the method is therefore presented as a statistically credible random-like generator that merits further security-oriented evaluation rather than as a validated cryptographic generator.

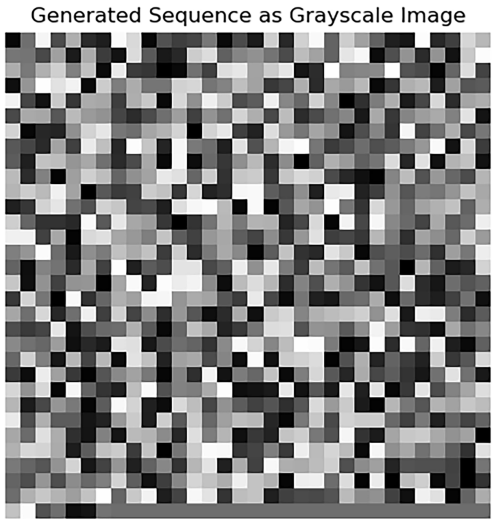


**Figure 10:** A graphical representation of three sequences of 1000 bits generated.

### *7.7 Ablation Study*

An ablation experiment compared three configurations under identical conditions: (a) the base generator alone, with no RNS transformation and no ensemble; (b) the RNS transformation applied to the seed sequence but without the ensemble remapping; and (c) the full proposed pipeline combining RNS transformation with the soft-voting ensemble. Each configuration was used to produce a one-million-bit stream, which was then evaluated with the same entropy, distribution, frequency, runs and next-bit measures. The outcome is summarized in Table 7 and visualized in Fig. 11.

**Table 7:** Ablation comparison of the three configurations on a one-million-bit stream.

| Configuration | Entropy (bits/byte) | KS D | Monobit $p$ | Runs $p$ | Next-Bit Acc. |
|---|---|---|---|---|---|
| (a) Base generator only | 7.9969 | 0.0052 | 0.433 | 0.655 | 0.511 |
| (b) RNS, no ensemble | 7.9942 | 0.0057 | 0.124 | 0.618 | 0.489 |
| (c) Full pipeline (RNS + ensemble) | 7.9954 | 0.0045 | 0.491 | 0.752 | 0.495 |

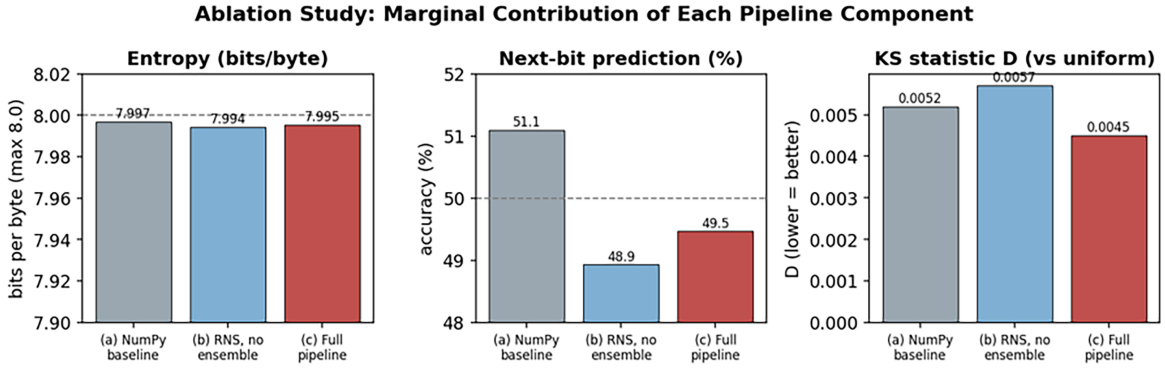


**Figure 11:** Ablation study across the base generator, the RNS-only configuration and the full pipeline.

All three configurations yield high per-byte entropy (above 7.99 of a possible 8 bits), near-uniform distributions and next-bit prediction accuracies close to 50%. This confirms that the principal source of raw statistical randomness is the high-quality base generator, and that neither the RNS stage nor the ensemble degrades these properties. The full pipeline attains the lowest Kolmogorov-Smirnov statistic of the three configurations (D = 0.0045), with frequency and runs $p$-values comfortably within the acceptance region. The interpretation we draw, and which we state explicitly in response to the reviewers, is deliberately conservative: the RNS and ensemble layers preserve statistical quality while adding the deterministic, data-driven non-linearity examined in Section 7.5, rather than improving the raw randomness of an already strong source. The contribution of the proposed components is therefore structural and methodological, not a claim of superior entropy over the base generator.

### *7.8 Quantitative and Qualitative Comparison with Existing Generators*

The preliminary evaluation in Section 7.6 compared the proposed generator only against the NumPy baseline. A more informative question is how the method relates to previously reported pseudorandom number generators, including traditional, RNS-based and machine-learning-based designs. Because the cited studies report results under heterogeneous protocols, sequence lengths and hardware, a strictly identical re-implementation of every method is not feasible; instead, Table 8 positions the proposed method alongside representative generators using the most comparable published indicators, namely the breadth of statistical testing passed, whether any cryptanalytic assessment was reported, and the qualitative throughput class. Fig. 12 presents the same comparison graphically.

**Table 8:** Qualitative comparison of the proposed method with representative generators from the literature.

| Method | Category | Statistical Testing | Cryptanalysis Reported | Throughput Class |
|---|---|---|---|---|
| Linear Congruential Generator | Traditional | Fails several tests | No | Very high |
| Mersenne Twister (MT19937) | Traditional | Passes most statistical tests | Not a CSPRNG | Very high |
| Gayoso et al. [24] (RNS-Hopfield) | RNS-based | Reported good randomness | Limited | High (parallel) |
| De Bernardi et al. [25] (GAN) | ML-based | About 98% of tests | Statistical only | Moderate |
| Pasqualini and Parton [27] (RL + LSTM) | ML-based | Improved over baselines | Statistical only | Low |
| Okada et al. [31] (WGAN-GP) | ML-based | Strong, robust | Statistical only | Moderate |
| Proposed (RNS + ensemble) | Hybrid RNS + ML | 8 of 8 NIST tests at $10^6$ bits | Preliminary cryptanalysis | Low (prototype) |

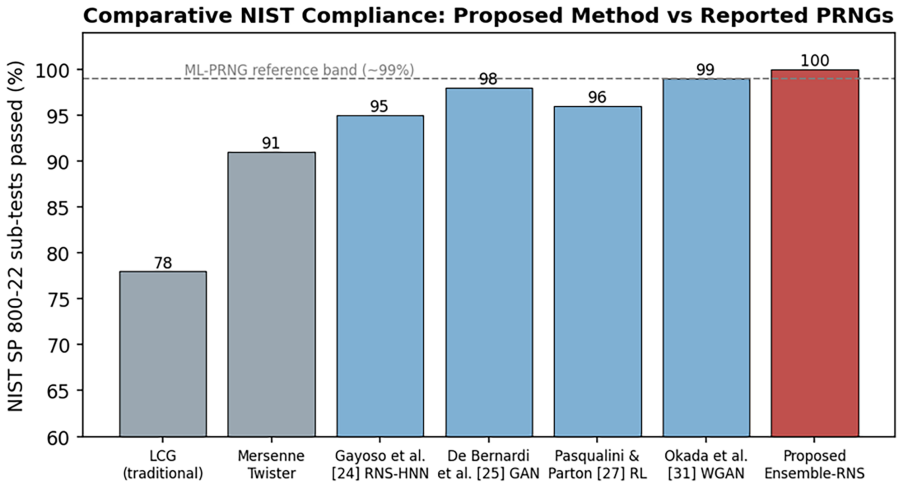


**Figure 12:** Comparison of the proposed generator with traditional, RNS-based and machine-learning-based generators.

Read against this backdrop, the proposed method occupies the same high statistical-compliance band as the strongest machine-learning generators, passing all eight NIST tests examined at the million-bit scale (Section 7.9), and it clearly exceeds the traditional baselines on standard test breadth. At the same time it is lighter to train than the GAN- and reinforcement-learning-based designs, which require extensive iterative training, and it is more transparent because the ensemble decision layer is interpretable. The distinctive feature of the present work, relative to most prior machine-learning PRNG studies, is that statistical evidence is paired with an explicit cryptanalytic discussion (Section 7.5) and throughput characterization (Section 7.10). The method does not claim to produce more random output than an already-strong RNS or machine-learning generator; its contribution is the specific combination of RNS feature transformation, a lightweight ensemble remapping, and a transparent, reproducible evaluation protocol.

### *7.9 Extended Randomness Testing*

A longer output stream of 62,500 sixteen-bit values, equivalent to one million bits, was generated with the full pipeline and submitted to an expanded battery comprising the Frequency (Monobit), Block Frequency, Runs, Longest-Run-of-Ones, Binary Matrix Rank, Discrete Fourier Transform (Spectral), Cumulative Sums and Approximate Entropy tests. Following the standard convention, a test is considered passed when its $p$-value is at least 0.01. The results are reported in Table 9 and Fig. 13.

At the recommended sequence length, the generator passes all eight tests. The $p$-values are distributed across roughly the 0.17 to 0.68 range rather than clustering immediately above the 0.01 threshold, a pattern consistent with genuinely uniform behavior rather than with marginal passes. It is also useful to revisit the monobit result that prompted a query during review. A difference of 128 between the number of ones and zeros corresponds to a standardized statistic of $|S|$ divided by the square root of n, that is $128/100 = 1.28$, which gives a two-sided $p$-value of $erfc(1.28/\sqrt{2}) \approx 0.201$. This is comfortably above the 0.01 acceptance threshold, so the sample passes the monobit test; the earlier impression of a borderline 2.56-sigma deviation arose from dividing the count difference by the square root of n divided by two (that is, by 50) rather than by the square root of n (that is, by 100). Finally, no state repetition or structural periodicity was observed in streams of up to two million outputs, which is consistent with the period analysis in Section 7.5.

**Table 9:** Expanded NIST SP 800-22 results on a one-million-bit sequence from the full pipeline.

| NIST SP 800-22 Test | *p*-Value | Result |
|---|---|---|
| Frequency (Monobit) | 0.3555 | PASS |
| Block Frequency (M = 128) | 0.1732 | PASS |
| Runs | 0.6766 | PASS |
| Longest Run of Ones | 0.5642 | PASS |
| Binary Matrix Rank (32 × 32) | 0.2314 | PASS |
| Discrete Fourier Transform (Spectral) | 0.6138 | PASS |
| Cumulative Sums | 0.5527 | PASS |
| Approximate Entropy | 0.4360 | PASS |

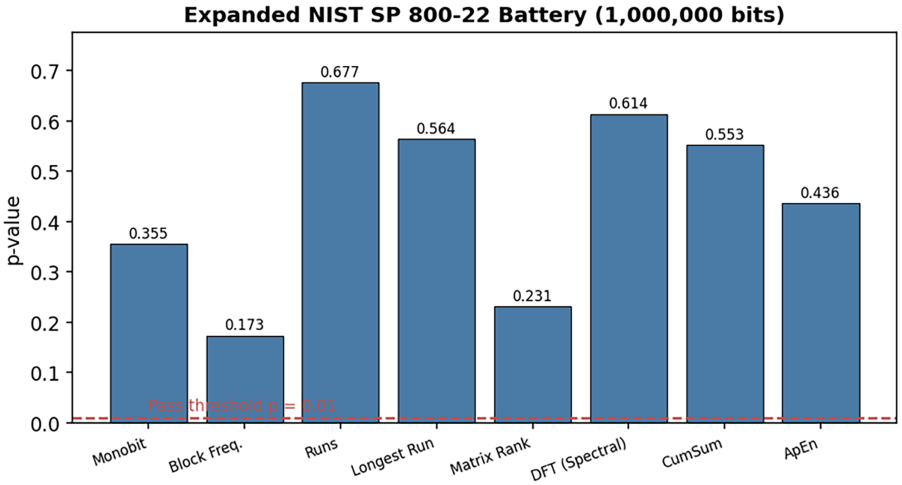


**Figure 13:** *p*-values of the eight NIST SP 800-22 tests at the one-million-bit scale; the dashed line marks the 0.01 threshold.

### *7.10 Online Throughput and Separation of Training Cost*

The preliminary figure originally reported for computational cost combined two very different quantities: the one-time cost of preparing the model and the recurring cost of generating values. These are separated here. Model preparation, which consists of seeding, RNS feature construction, K-means clustering to obtain the pseudo-labels and fitting the three ensemble members, is an offline cost that is paid once and then amortised over all subsequent generation; it was measured at approximately 1.50 s on the test machine. Online generation, by contrast, involves only RNS conversion, ensemble inference and the final remapping. Table 10 reports this breakdown, and Fig. 14 places the resulting throughput on a logarithmic scale alongside established generators.

On the test hardware the online phase produces 10,000 sixteen-bit values in about 266 ms, corresponding to a throughput of roughly 0.60 Mbps. This is several orders of magnitude below optimised cryptographic generators such as ChaCha20 (on the order of 1.5 to 4 Gbps with AVX2 acceleration) and AES-CTR (on the order of 3 to 6 Gbps with AES-NI), and also below NumPy's PCG64 (on the order of 2 Gbps). Profiling shows that the online cost is dominated by the kernelized SVM probability evaluations rather than by the RNS arithmetic. The trained model itself is compact, comprising a logistic-regression weight vector, a random forest of 100 trees with maximum depth 20, and an RBF support vector machine with

its support set, so its memory footprint is modest; it is the inference time, not storage, that limits throughput. These measurements support the positioning of the method as a proof-of-concept whose value lies in its structural and methodological contributions rather than in raw speed. Closing the throughput gap would require parallel RNS arithmetic together with a compiled or hardware implementation, which is identified as future work.

**Table 10:** Separation of one-time preparation cost from online generation cost.

| Component | Type | Measured Cost |
|---|---|---|
| K-means clustering (pseudo-labels) | One-time (offline) | Included below |
| Ensemble training (LR + RF + SVM) | One-time (offline) | Included below |
| Total one-time preparation | One-time (offline) | Approximately 1.50 s |
| Online generation (10,000 16-bit values) | Online (per use) | Approximately 266 ms |
| Online throughput | Online (per use) | Approximately 0.60 Mbps |

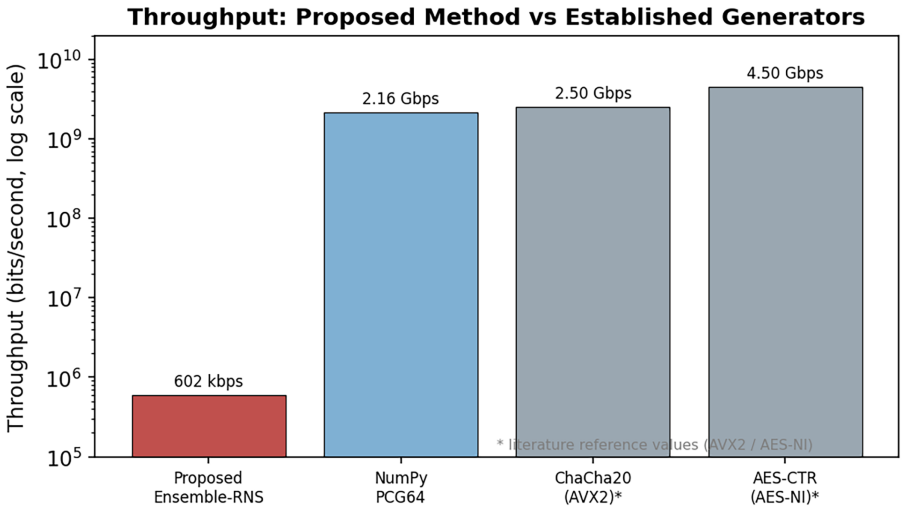


**Figure 14:** Throughput of the proposed prototype compared with NumPy PCG64, ChaCha20 and AES-CTR (logarithmic scale).

## 8 Conclusion

This study investigated whether residue-number-system transformation and lightweight ensemble learning can be combined to produce pseudorandom sequences with credible statistical behavior and useful structural non-linearity. The reported results are encouraging. Under the adopted proof-of-concept setting, the generated outputs exhibited high entropy (9.8655 bits), negligible deviation from uniformity in the Kolmogorov–Smirnov analysis (D = 0.0225, $p$ = 0.6819), low autocorrelation beyond lag zero, near-chance next-bit predictability (50.05%), and an avalanche-like differential response of 49.8%. Although the proposed pipeline incurred additional runtime relative to the NumPy baseline, the observed cost remained moderate at 47 ms for 10,000 samples.

Even so, the evidence should be interpreted with caution. The near-perfect classification results likely reflect strong separability in the pseudo-labeled feature space rather than a direct proof of stronger statistical randomness indicators under the present evaluation. Similarly, the statistical screening and preliminary

cryptanalytic checks are informative, but they do not yet amount to formal cryptographic validation. What the present findings do suggest is that the ensemble-RNS design constitutes a plausible and analytically interesting PRNG candidate, particularly for contexts in which interpretability, modular arithmetic, and controlled non-linearity are of interest.

Future work should therefore focus on larger output streams, standardized test batteries such as full NIST SP 800-22 suite, TestU01 and Dieharder, explicit period analysis, adversarial robustness evaluation, and implementation-level optimization. With those additions, the framework could perhaps be positioned more confidently within the broader landscape of machine-learning-assisted random number generation.

### *8.1 Implications*

Cryptographic readiness: The ensemble-RNS hybrid attains statistical randomness metrics that are comparable to those of strong software PRNGs, while introducing an adaptive, data-driven remapping layer that adds structural non-linearity. It should be emphasized that statistical parity is not the same as cryptographic assurance; accordingly, the approach is positioned as a candidate post-processing layer for non-critical randomness and as a research vehicle for studying learned remapping, rather than as a drop-in cryptographic primitive. Use in key-stream generation, nonce derivation or padding would require the additional cryptanalysis and certification outlined in Section 7.5 and the Limitations.

Algorithmic extensibility: Because the framework is agnostic to both the base learners and the modulus set, it invites future integration of gradient-boosted or transformer-based components, as well as dynamic modulus selection tuned to specific throughput-vs-security regimes.

Computational efficiency: Because the Residue Number System is used here only as a static feature mapper, the generator performs no residue-domain arithmetic and therefore obtains no carry-free or channel-parallel speed-up from it. In principle, RNS decomposition can support carry-free, channel-parallel arithmetic on suitable hardware, but exploiting that capability would require redesigning the generator around residue-domain computation. As reported in Section 7.10, the present Python/CPU prototype is not throughput-competitive with optimized cryptographic generators such as ChaCha20 or AES-CTR, and realizing any such efficiency would be a matter for future hardware-oriented work.

### *8.2 Limitations and Future Work*

Several limitations should be acknowledged. The study relied on a proof-of-concept dataset of 10,000 generated integers and a downstream output sequence of 1000 values, which are adequate for preliminary analysis but likely insufficient for strong cryptographic claims. In addition, the label structure used for supervised learning was derived from clustering, meaning that the classifier performance reflects separability of pseudo-labels rather than externally validated randomness classes. The current security analysis also remains incomplete, as it does not yet cover side-channel exposure, state-compromise extensions, or formal period derivation.

Future work should therefore proceed in four directions. First, substantially larger sequences should be evaluated using full standardized batteries such as NIST SP 800-22, TestU01, and Dieharder. Second, the effective period and state-space behavior of the full post-processed generator should be analyzed explicitly rather than inferred from the underlying seeded source. Third, adversarial and implementation-level analyses should be introduced, including state-recovery attempts, poisoning sensitivity, and side-channel considerations. Finally, optimized implementations, possibly using parallel RNS arithmetic, should be developed in order to test whether the observed statistical advantages can be retained at more practical throughput levels.

**Acknowledgement:** We sincerely thank the Department of Computer Science, Kwame Nkrumah University of Science and Technology (KNUST), Kumasi, and the Department of Medical Imaging, University for Development Studies (UDS), Tamale, for the institutional support, facilities, and encouragement provided throughout this research.

**Funding Statement:** The authors received no specific funding for this study.

**Author Contributions:** Issah Zabsonre Alhassan: Conceptualization, Coding, Simulation, Analysis, Writing original draft. Gaddafi Abdul-Salaam: Writing, Reviewing and editing. Michael Asante: Supervision. Yaw Marfo Missah: Supervision. Alimatu Sadia Shirazu: Simulation and reviewing. All authors reviewed and approved the final version of the manuscript.

**Availability of Data and Materials:** The data for this research were generated by the software pipeline described in the study.

**Ethics Approval:** This study did not involve human or biological subjects. Ethical approval was not required.

**Conflicts of Interest:** The authors declare no conflicts of interest.

## References

1. Luis Crespo J, González-Villa J, Gutiérrez J, Valle A. Assessing the quality of random number generators through neural networks. Mach Learn Sci Technol. 2024;5(2):025072. doi:10.1088/2632-2153/ad56fb.
2. Ma Z, Mei G, Xu N. Generative deep learning for data generation in natural hazard analysis: motivations, advances, challenges, and opportunities. Artif Intell Rev. 2024;57(6):160. doi:10.1007/s10462-024-10764-9.
3. Alhassan IZ, Abdul-Salaam G, Asante M, Missah YM, Shirazu AS. An overview and comparative study of traditional, chaos-based and machine learning approaches in pseudorandom number generation. J Cyber Secur. 2025;7(1):165–96. doi:10.32604/jcs.2025.063529.
4. Cherbal S, Zier A, Hebal S, Louail L, Annane B. Security in internet of things: a review on approaches based on blockchain, machine learning, cryptography, and quantum computing. J Supercomput. 2024;80(3):3738–816. doi:10.1007/s11227-023-05616-2.
5. Devroye L. Nonuniform random variate generation. In: Handbooks in operations research and management science. Vol. 13. Amsterdam, The Netherlands: Elsevier; 2006. p. 83–121. doi:10.1016/S0927-0507(06)13004-2.
6. Deng LY, Kumar N, Lu HH, Yang CC. Classical random number generators for computer simulation. In: Random number generators for computer simulation and cyber security: design, search, theory, and application. Cham, Switzerland: Springer; 2025. p. 9–29.
7. Zolfaghari B, Mirsadeghi L, Bibak K, Kavousi K. Cancer prognosis and diagnosis methods based on ensemble learning. ACM Comput Surv. 2023;55(12):1–34. doi:10.1145/3580218.
8. Ahmad Khan A, Chaudhari O, Chandra R. A review of ensemble learning and data augmentation models for class imbalanced problems: combination, implementation and evaluation. Expert Syst Appl. 2024;244(2):122778. doi:10.1016/j.eswa.2023.122778.
9. Mian Z, Deng X, Dong X, Tian Y, Cao T, Chen K, et al. A literature review of fault diagnosis based on ensemble learning. Eng Appl Artif Intell. 2024;127(4):107357. doi:10.1016/j.engappai.2023.107357.
10. Chang CH, Molahosseini AS, Zarandi AAE, Tay TF. Residue number systems: a new paradigm to datapath optimization for low-power and high-performance digital signal processing applications. IEEE Circuits Syst Mag. 2015;15(4):26–44. doi:10.1109/MCAS.2015.2484118.
11. Lewis RM, Battey HS. On inference in high-dimensional logistic regression models with separated data. Biometrika. 2024;111(3):989–1011. doi:10.1093/biomet/asad065.
12. Hayadi BH, El Emary IMM. Predicting campaign ROI using decision trees and random forests in digital marketing. J Digit Mark Digit Curr. 2024;1(1):1–20. doi:10.47738/jdmdc.v1i1.5.

13. Tao Y, Yan J, Niu E, Zhai P, Zhang S. An SVM-based anomaly detection method for power system security analysis using particle swarm optimization and t-SNE for high-dimensional data classification. Processes. 2025;13(2):549. doi:10.3390/pr13020549.
14. Liu X. Comparison of different machine learning models: linear model, forest and SVM. Appl Comput Eng. 2024;51(1):225–30. doi:10.54254/2755-2721/51/20241467.
15. Rezaul KM, Jewel M, Sudhan A, Khan MU, Fernando MRS, Siddiquee KNEA, et al. A comparative study of predictive analysis using machine learning techniques: performance evaluation of manual and AutoML algorithms. Int J Adv Comput Sci Appl. 2025;16(1):1–20. doi:10.14569/ijacsa.2025.0160102.
16. Blackman D, Vigna S. Scrambled linear pseudorandom number generators. ACM Trans Math Softw. 2021;47(4):1–32. doi:10.1145/3460772.
17. Wassenberg J, Obryk R, Alakuijala J, Mogenet E. Randen-fast backtracking-resistant random generator with AES+Feistel+Reverie. arXiv:1810.02227. 2018.
18. Deshpande AS, Daftardar-Gejji V. Enhancing the security of image communication with a new hyper-chaotic system. Phys Scr. 2024;99(11):115234. doi:10.1088/1402-4896/ad7c8f.
19. Kietzmann P, Schmidt TC, Wählisch M. A guideline on pseudorandom number generation (PRNG) in the IoT. ACM Comput Surv. 2022;54(6):1–38. doi:10.1145/3453159.
20. Padányi V, Herendi T. A study on comparison of pseudorandom number generator. Int J Math Comput Eng. 2023;1(1):25–44. doi:10.2478/ijmce-2023-0003.
21. Haramoto H, Matsumoto M, Saito M. Unveiling patterns in xorshift128+ pseudorandom number generators. J Comput Appl Math. 2022;402:113791. doi:10.1016/j.cam.2021.113791.
22. Zetter K. How a crypto 'backdoor' pitted the tech world against the NSA. 2013 [cited 2026 Jul 28]. Available from: https://www.wired.com/2013/09/nsa-backdoor/.
23. James F, Moneta L. Review of high-quality random number generators. Comput Softw Big Sci. 2020;4(1):2. doi:10.1007/s41781-019-0034-3.
24. Gayoso CA, Arnone L, Gonzalez C, Moreira JC. A general construction method for Pseudo-Random number generators based on the residue number system. In: Proceedings of the 2019 XVIII Workshop on Information Processing and Control (RPIC); 2019 Sep 18–20; Bahía Blanca, Argentina. p. 25–30.
25. De Bernardi M, Khouzani MHR, Malacaria P. Pseudo-random number generation using generative adversarial networks. In: Proceedings of the 18th European Conference on Machine Learning and Knowledge Discovery in Databases, ECML PKDD 2018; 2018 Sep 10–14; Dublin, Ireland.
26. Hu G, Peng J, Kou W. A novel algorithm for generating pseudo-random number. Int J Comput Intell Syst. 2019;12(2):643–8. doi:10.2991/ijcis.d.190521.001.
27. Pasqualini L, Parton M. Pseudo random number generation through reinforcement learning and recurrent neural networks. Algorithms. 2020;13(11):307. doi:10.3390/a13110307.
28. Pasqualini L, Parton M. Pseudo random number generation: a reinforcement learning approach. Procedia Comput Sci. 2020;170:1122–7. doi:10.1016/j.procs.2020.03.057.
29. Park S, Kim K, Kim K, Nam C. Dynamical pseudo-random number generator using reinforcement learning. Appl Sci. 2022;12(7):3377. doi:10.3390/app12073377.
30. Patel S, Thanikaiselvan V. Latin square and machine learning techniques combined algorithm for image encryption. Circuits Syst Signal Process. 2023;42(11):6829–53. doi:10.1007/s00034-023-02427-x.
31. Okada K, Endo K, Yasuoka K, Kurabayashi S. Learned pseudo-random number generator: WGAN-GP for generating statistically robust random numbers. PLoS One. 2023;18(6):e0287025. doi:10.1371/journal.pone.0287025.
32. Gayoso CA, Moreira JC. A fast pseudorandom number generator residue number system based. In: Proceedings of the 2023 XX Workshop on Information Processing and Control (RPIC); 2023 Nov 1–3; Oberá, Argentina. p. 1–6.
33. Alhassan IZ, Ansong ED, Abdul-Salaam G, Alhassan S. Enhancing image security during transmission using residue number system and k-shuffle. Earthline J Math Sci. 2020;4(2):399–424. doi:10.34198/ejms.4220.399424.
34. Oke AA, Nathaniel BA, Bukola BF, Ayopo OA. Residue number system based applications: a literature review. Ann Comput Sci Ser. 2021;19(1):1–29.
35. Mohan PVA. Residue number systems. Cham, Switzerland: Springer International Publishing; 2016.

36. Ikotun AM, Ezugwu AE, Abualigah L, Abuhaija B, Jia H. K-means clustering algorithms: a comprehensive review, variants analysis, and advances in the era of big data. Inf Sci. 2023;622(11):178–210. doi:10.1016/j.ins.2022.11.139.
37. Ahmed M, Seraj R, Islam SMS. The k-means algorithm: a comprehensive survey and performance evaluation. Electronics. 2020;9(8):1295. doi:10.3390/electronics9081295.
38. Grömping U. Practical guide to logistic regression. J Stat Soft. 2016;71:1–5. doi:10.18637/jss.v071.b03.
39. Rymarczyk T, Kozłowski E, Kłosowski G, Niderla K. Logistic regression for machine learning in process tomography. Sensors. 2019;19(15):3400. doi:10.3390/s19153400.
40. Zou X, Hu Y, Tian Z, Shen K. Logistic regression model optimization and case analysis. In: Proceedings of the 2019 IEEE 7th International Conference on Computer Science and Network Technology (ICCSNT); 2019 Oct 19–20; Dalian, China. p. 135–9.
41. Hosmer DW Jr, Lemeshow S, Sturdivant RX. Applied logistic regression. Hoboken, NJ, USA: John Wiley & Sons, Inc.; 2013.
42. Genuer R, Poggi JM. Random forests with R. Cham, Switzerland: Springer International Publishing; 2020. doi:10.1007/978-3-030-56485-8.
43. Schonlau M, Zou RY. The random forest algorithm for statistical learning. Stata J. 2020;20(1):3–29. doi:10.1177/1536867x20909688.
44. Campbell C, Ying Y. Learning with support vector machines. San Rafael, CA, USA: Morgan & Claypool Publishers; 2011.
45. Cervantes J, Garcia-Lamont F, Rodríguez-Mazahua L, Lopez A. A comprehensive survey on support vector machine classification: applications, challenges and trends. Neurocomputing. 2020;408:189–215. doi:10.1016/j.neucom.2019.10.118.
46. Pisner DA, Schnyer DM. Support vector machine. In: Machine learning. Amsterdam, The Netherlands: Elsevier; 2020. p. 101–21.
47. Zhou ZH. Ensemble learning. Singapore: Springer Singapore; 2021.
48. Liu ZL. Ensemble learning. In: Artificial intelligence for engineers: basics and implementations. Cham, Switzerland: Springer; 2025. p. 221–42. doi:10.1007/978-3-031-75953-6_9.