Journal of Electrical and Computer Engineering Innovations
(JECEI)
Journal homepage: http://www.jecei.sru.ac.ir

Research paper

Adaptive Multi-Layer Random Generator: Toward Self-Regulating Pseudorandomness

Ali Bazghandi *

Faculty of Computer and IT Engineering, Shahrood University of Technology, Shahrood, Iran.

Article Info

Article History:
Received 26 October 2025
Reviewed 15 December 2025
Revised 19 February 2026
Accepted 23 February 2026

Keywords:
Hybrid entropy sources
Pseudo-Randomness Evaluation
Feedback and Stratification Mechanisms
Statistical Uniformity and Independence

*Corresponding Author's Email Address:
bazghandi@shahroodut.ac.ir

Abstract

Background and Objectives: Random number generation is essential in simulation, cryptography, and statistical modeling. Classical PRNGs such as the Linear Congruential Generator and Mersenne Twister are efficient but exhibit predictability and correlation. Newer families like PCG and BRG improve statistical balance yet remain static after initialization, while chaotic and neural methods face reproducibility and stability issues. To overcome these limits, we propose the Adaptive Multi-Layer Random Generator (AMLRG), designed to deliver self-regulating pseudorandomness through adaptive feedback and hybrid entropy sources.

Methods: AMLRG combines three layers: (i) an index generator based on linear, logistic, or chaotic processes, (ii) a uniform distribution module, and (iii) an adaptive feedback system that tunes parameters in real time. Online diagnostics—Kolmogorov–Smirnov tests, autocorrelation analysis, and Shannon entropy—direct dynamic adjustment. Implemented in Python, the system produces binary streams and diagnostic plots. Evaluation involved ablation studies (removing feedback, switching, or stratification), comparison with LCG, PCG, BRG, and logistic-only baselines, and validation using Dieharder, TestU01, and NIST SP 800-22.

Results: AMLRG produced lower KS distances, near-zero autocorrelation, and entropy close to theoretical maxima, outperforming all baselines. Ablation confirmed the contribution of each layer to statistical quality. Results show stable behavior across 200,000 values, with speed comparable to PCG but greater adaptability.

Conclusion: AMLRG introduces dynamic correction that improves independence and uniformity in pseudorandom sequences. Its layered architecture suits engineering simulations, adaptive systems, and security-sensitive statistical preprocessing. Future work will target non-uniform distributions, GPU acceleration, and hardware implementation.

How to cite this paper:
A. Bazghandi, "Adaptive multi-layer random generator: toward self-regulating pseudorandomness," J. Electr. Comput. Eng. Innovations, 14(2): 365-376, 2026.
DOI: 10.22061/jecei.2026.12518.887
URL: https://jecei.sru.ac.ir/article_2539.html

Introduction

Random number generation underpins a wide spectrum of computational tasks, from Monte Carlo simulations and probabilistic modeling to cryptographic protocols and gaming. The quality of generated random sequences directly influences computational outcomes; weak pseudorandomness may distort simulation results or compromise security strength [1], [2]. Physical sources of randomness (e.g., radioactive decay, thermal noise) produce inherently random outputs but are expensive, noisy, or slow. Algorithmic pseudo-random number generators (PRNGs), by contrast, are efficient and reproducible, but their deterministic nature makes them vulnerable to statistical biases, finite periods, and correlations [3].

The Linear Congruential Generator (LCG) [4] remains one of the simplest and most widely used PRNGs, but its lattice structures and sequential correlations are well documented [1], [5]. The known vulnerabilities of LCGs, including short periods and poor performance in high-dimensional spaces, motivated early research into their statistical weaknesses [6]. Later designs, including the Mersenne Twister [2], combined recursive generators [7], and the PCG family [8], improve statistical quality and extend periods. Furthermore, the selection of the PRNG can significantly influence results in fields like Monte Carlo simulations [9] and the training of machine learning models [10]. However, these generators remain static: once seeded, they cannot adapt to emerging statistical deviations during runtime.

Hybrid approaches such as the Bipartite Random Generator (BRG) [11] introduced separation of index generation and uniformization, showing reduced autocorrelation compared with LCGs. Still, BRG lacks adaptive correction and may degrade over very long sequences. Meanwhile, recent advances in chaotic systems [12]-[14], machine learning–based PRNGs [15], [16], and hardware multi-source implementations [17] demonstrate the push toward more dynamic and flexible randomness. The convergence of chaos theory and hardware design has led to new FPGA-based solutions for high-speed generation [18], [19]. Yet, most approaches still rely on offline statistical validation rather than continuous, in-process correction.

Despite significant advances in pseudorandom number generation, most widely used PRNGs remain static once initialized and cannot respond to statistical drift that may emerge during long-running executions. In applications such as large-scale simulations, adaptive systems, and iterative modeling, even weak biases or correlations can accumulate over time and affect outcomes. This limitation motivates the development of adaptive PRNG frameworks capable of monitoring their own statistical behavior and correcting deviations during runtime.

AMLRG is proposed to address this gap by combining hybrid entropy sources with continuous, lightweight feedback control.

The main contributions of this work are as follows:

- We propose the Adaptive Multi-Layer Random Generator (AMLRG), a self-regulating pseudorandom number generation framework that integrates online statistical diagnostics with adaptive correction.
- We introduce a lightweight feedback mechanism that dynamically mitigates distributional bias, autocorrelation, and entropy degradation during runtime.
- We provide an empirical evaluation, including ablation studies and comparisons with classical, modern lightweight, chaotic, and neural PRNGs.

In addition, the marginal contributions include:

- A parameterized evaluation framework highlighting trade-offs between statistical rigor and performance.
- A reproducibility package enabling transparent benchmarking and validation.
- A discussion of practical robustness, failure modes, and applicability limits.

Related Work

The work in this scope involves three main categories.

A. Classical Generators

Early methods such as the Middle-Square Method [20] and Midproduct Method rapidly degenerate, producing short cycles. The Linear Congruential Generator (LCG) [21] became the standard for many years, but the discovery of its structure in the output bits [4] led to the development of better linear methods, such as Multiple Recursive Generators (MRGs) [7]. These algorithms are foundational; however, their limitations in modern high-stakes applications, like cryptography or large-scale, long-running simulations, necessitated a move toward statistically superior and cryptographically secure alternatives [22].

B. Chaotic Systems and Neural Network PRNGs

The non-linear dynamics of chaotic systems, such as Chen and Lorenz, are frequently harnessed for PRNG design, offering high sensitivity to initial conditions and long periods [23]. Hardware implementations often combine these chaotic maps with physical noise or feedback mechanisms to prevent degradation and pass rigorous statistical tests [18]. Recently, the field has seen a proliferation of Generative Adversarial Networks (GANs) and other deep learning models applied to PRNGs, aiming to learn and reproduce high-quality

random sequences [24], [25]. The use of neural networks to model and predict PRNG output is a growing area, leading to new methods for both generation and security analysis [26]. Furthermore, models based on Recurrent Neural Networks (RNNs) and Generative Adversarial Networks (GANs) have been explored specifically for generating high-quality sequences [27].

C. Adaptive and Dynamic Approaches

While not fully adaptive, combined generators, such as those based on XORShift and splitting/mixing functions (e.g., PCG [8]), represent an evolution toward more robust PRNGs [28]. The principle of combining and mixing multiple sources to improve statistical quality is a key theme [29]. The AMLRG framework builds upon these ideas by introducing continuous online monitoring and adaptive parameter adjustment to dynamically counter observed statistical deviations. The final evaluation of such generators typically relies on comprehensive statistical packages like Dieharder [19] and TestU01 [30].

Proposed Method: Adaptive Multi-Layer Random Generator (AMLRG)

The proposed AMLRG framework introduces adaptivity into pseudorandom number generation through a three-layer design: (i) an Index Generator Layer, (ii) a Uniformization Layer, and (iii) an Adaptive Feedback Layer. Unlike static PRNGs, AMLRG continuously monitors the statistical quality of its output and adjusts generator parameters in real time. The overall architecture is illustrated in Fig. 1.

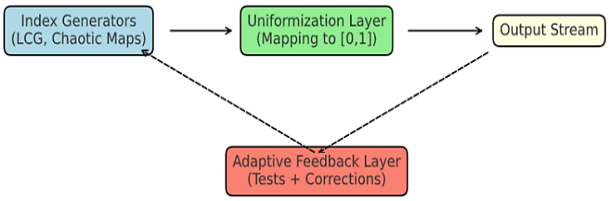

Fig. 1: AMLRG architecture with three coordinated layers: Index generator, uniformization, and adaptive feedback.

A. Index Generator Layer

The first layer produces raw indices using lightweight generators such as the Linear Congruential Generator (LCG) or chaotic maps. The LCG recurrence relation is:

$$X_{n+1} = (aX_n + c)\ mod\ m \tag{1}$$

where, a is the multiplier, c is the increment, and m the modulus [4].

As an alternative, a chaotic logistic map can be employed:

$$X_{n+1} = r.X_n(1 - X_n), \tag{2}$$

$$0 < X_n < 1, \qquad 3.57 < r < 4$$

where parameter r determines the chaotic regime [12], [13].

The coexistence of multiple index generators enhances variability and robustness.

B. Uniformization Layer

To ensure statistical uniformity, each raw index value is mapped into the interval [0,1] (Fig. 2).

For LCG outputs:

$$U_n = \frac{X_n}{m} \tag{3}$$

A stratified mapping scheme may also be applied to balance coverage:

$$U_n = \frac{X_n}{m} + \frac{K}{N}\ ,\ K = 0,1,\dots,N-1 \tag{4}$$

where N is the number of strata.

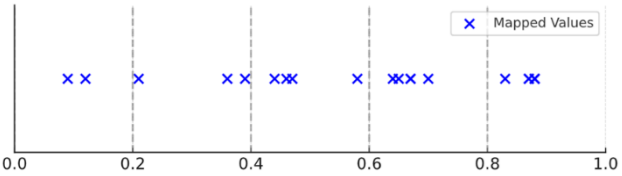

Fig. 2: Mapping of raw indices into stratified uniform intervals.

C. Adaptive Feedback Layer

The novelty of AMLRG lies in its self-regulating feedback loop. A sliding window $W = \{U_{n-k+1}, \dots, U_n\}$ of size k is subjected to lightweight statistical tests:

- Kolmogorov–Smirnov (KS) Test:

  For cumulative distribution Fn(x) and uniform CDF F(x)= x, the KS statistic is:

  $$D_n = \sup_X |F_n(X) - F(X)| \tag{5}$$

  If $D_n > \delta$ (threshold), reseeding or parameter rotation is triggered.

- Autocorrelation Test:

  The lag-h autocorrelation is computed as:

  $$\rho(h) = \frac{\sum_{i=1}^{N-h}(U_i - \bar{U})(U_{i+h} - \bar{U})}{\sum_{i=1}^{N}(U_i - \bar{U})^2} \tag{6}$$

  where, $\bar{U}$ is the mean of the sequence and h is the lag. If $|\rho(h)| > \epsilon$, corrective action is taken.

- Shannon Entropy (Entropy over sliding window):

  $$H = -\sum_{i=1}^{b} p_i \log_2 p_i \tag{7}$$

  where, p_i is the observed probability of bin i. Low entropy indicates clustering or bias.

  If deviations exceed predefined thresholds $(D_n > \delta, |\rho(h)| > \epsilon, or\ H < H_{min})$, corrective action is triggered. Fig. 3 shows the steps of the adaptive correction loop.

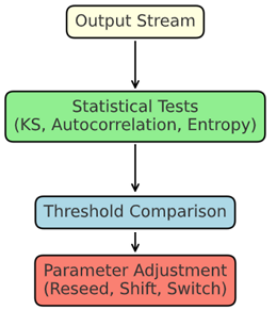

Fig. 3: Steps of the adaptive correction loop in AMLRG.

D. Adaptive Algorithm Rule

The decision logic of the feedback loop is:

$$\text{If } (D_n > \delta) \text{ or } (|\rho(h)| > \epsilon) \text{ or } (H < H_{min}) \Rightarrow \text{Corrective Action} \quad (8)$$

Corrective actions include reseeding, parameter shifting (e.g., multiplier a in LCG, control parameter r in logistic map), or switching between multiple index streams.

This adaptive framework allows AMLRG to continuously self-correct and maintain statistical quality over arbitrarily long sequences, distinguishing it from static PRNG architectures such as LCG, MT, or BRG.

E. Algorithm Sketch

Algorithm 1 outlines the AMLRG generation loop. The system instantiates multiple lightweight index generators, including an LCG and a logistic-map source, maps raw indices to [0,1] via a uniformizer with optional stratification, and then applies online tests over a sliding window. When the Kolmogorov–Smirnov deviation D_n, the magnitude of the lag-h autocorrelation |ρ(h)|, or the Shannon entropy H violates thresholds, AMLRG triggers corrective actions—reseeding, parameter tuning, and/or switching the active generator.

Algorithm 1: AMLRG generation with online adaptation

Inputs: window size K, sampling interval S, thresholds δ(KS), ϵ(ACF), Hmin (entropy), lag h, number of strata Ns.
State: active generator G∈{LCG, Logistic}; step counter t.

Initialize GGG, uniformizer, and stats buffers.
for t=1,…,n:
  a) Xt←G.step()
  b) Ut←Uniformize(Xt,t,Ns)
  c) Push Ut into sliding window (size K).
  d) if t mod S=0 and |W|≥S:
    • Compute Dn,ρ(h),H.
    • if Dn>δ or |ρ(h)|>ϵ or H<Hmin:
      – G.reseed() (optional)
      – G.tune() (optional)
      – Switch G (optional)
Return $\{U_t\}_{t=1}^{n}$.

Implementation notes. In our reference code, K=8192, S=1024, h=7, 64 entropy bins, and Ns=16 strata by default. Thresholds were set to δ=0.03, ϵ=0.02, Hmin=0.95·log2(64). Figure references to behavior over time are provided in Section 4.

F. Theoretical Considerations and Stability of the Adaptive Loop

The Adaptive Multi-Layer Random Generator (AMLRG) differs from classical static PRNGs in that its output process is explicitly non-stationary at the global scale, while being designed to maintain local stationarity within bounded observation windows. This section provides a theoretical discussion of the long-term behavior, stability, and potential risks of periodicity or bias introduced by the adaptive feedback mechanism.

- *Stationary Distribution and Long-Term Bias Control*

The primary objective of AMLRG is not to enforce a single immutable stationary process, but to dynamically steer the generator toward a target stationary distribution, namely the uniform distribution over [0,1]. At any time step t, AMLRG evaluates a sliding window Wt of size K and applies corrective actions only when deviations from uniformity, independence, or entropy exceed predefined thresholds.

From a control-theoretic perspective, the adaptive loop can be interpreted as a negative-feedback system that limits the persistence of statistical bias. Any deviation in empirical distribution, autocorrelation, or entropy is bounded in duration by the sampling interval S, after which corrective actions are triggered. As a result, while small transient deviations may occur, systematic long-term bias cannot accumulate unboundedly, since deviations are actively detected and corrected. In this sense, AMLRG enforces a form of bounded bias stability, where the magnitude and lifespan of deviations are constrained by δ, ϵ, and H_{min}.

- *Cycle Formation and Periodicity Considerations*

A common concern in adaptive systems is the potential emergence of limit cycles or periodic behavior induced by deterministic correction rules. In AMLRG, several design choices mitigate this risk:

1. Heterogeneous generator switching: Corrective actions may include switching between distinct index generators (e.g., linear and chaotic), which breaks deterministic recurrence patterns.
2. Stochastic reseeding and parameter tuning: Parameter updates are not fixed mappings but involve stochastic reseeding or non-deterministic adjustments, reducing the likelihood of repeated correction cycles.
3. Windowed and sparse feedback: Adaptation occurs only at discrete checkpoints separated by S outputs, preventing rapid oscillations in generator parameters.

Together, these mechanisms ensure that the adaptive loop does not converge to a simple periodic orbit. Instead, AMLRG behaves as a piecewise stochastic process, where local dynamics are continually perturbed,

preventing stable cycle formation even over very long sequences.

- *Residual Bias Bounds and Correction Dynamics*

Let Δ_t denote a deviation metric (e.g., KS distance or autocorrelation) measured over window W_t. By construction, AMLRG enforces

$$\Delta_t \leq max(\delta, \epsilon) + O(\frac{1}{K}) \quad (9)$$

After at most one correction cycle. This implies that any residual bias is upper-bounded by the chosen thresholds and the finite-sample resolution of the statistical tests. Importantly, AMLRG does not attempt to eliminate all deviation instantaneously; instead, it guarantees that deviations are short-lived and bounded, which is sufficient for most simulation and statistical applications.

- *Stability of the Adaptive Feedback Loop*

The adaptive feedback mechanism can be viewed as a discrete-time control loop with delayed observations (windowed statistics) and bounded corrective actions. Stability is maintained through:

1. Threshold-based triggering, which avoids overreaction to noise,
2. Sampling interval S, which damps rapid parameter oscillations,
3. Multiple corrective options (reseeding, tuning, switching), which prevent single-path feedback lock-in.

Under reasonable parameter choices (e.g., K≫h, moderate S), the system exhibits practical stability, characterized by infrequent corrections, rapid recovery after violations, and statistically stationary behavior within windows. Empirical results in later sections support this stability by showing low correction rates and fast post-correction convergence.

- *Predictability and Scope Limitations*

It is important to emphasize that AMLRG is not intended to provide cryptographic security guarantees. While adaptive correction reduces detectable bias and correlation, the feedback mechanism itself introduces structure that could, in principle, be modeled by an adversary with full system knowledge. AMLRG is therefore best suited for simulation, modeling, and statistical workloads where adaptability and long-run statistical robustness are more critical than adversarial unpredictability.

- *Predictability and Feedback-Induced Patterns*

Adaptive feedback mechanisms, while effective in correcting statistical deviations, may themselves introduce structural regularities or predictable patterns if the correction rules are overly deterministic. In AMLRG, this concern is particularly relevant because statistical diagnostics (e.g., KS distance, autocorrelation, entropy) directly influence parameter updates and generator switching.

AMLRG mitigates feedback-induced predictability through several design choices. First, corrective actions are event-driven rather than periodic, triggered only when deviations exceed predefined thresholds. This avoids fixed adjustment schedules that could imprint regular patterns on the output. Second, AMLRG employs heterogeneous index generators with distinct internal dynamics; switching between linear and chaotic sources disrupts consistent recurrence structures that might otherwise emerge. Third, reseeding and parameter tuning incorporate stochastic elements, ensuring that corrective responses are not deterministic functions of past outputs alone.

Nevertheless, it is acknowledged that AMLRG's adaptive loop introduces a form of controlled structure into the generation process. Unlike purely static PRNGs, AMLRG trades strict determinism for adaptability, which may expose weak forms of predictability to an adversary with full knowledge of the system state and parameters. For this reason, AMLRG is not claimed to be cryptographically secure and is intended primarily for simulation, modeling, and statistical applications where long-run distributional stability and independence are the primary objectives.

Empirical results in later sections provide evidence that feedback-induced patterns, if present, do not manifest as detectable bias or correlation under standard statistical batteries. However, the potential interaction between adaptive correction rules and predictability remains an important direction for future theoretical analysis, particularly in adversarial or cryptographic contexts.

Evaluation Plan

A. Baselines and Variants

We compare AMLRG against (i) a standalone LCG [4], (ii) BRG [11], (iii) PCG [8], and (iv) a chaos-only generator (logistic map, fixed r). We also include ablations: AMLRG-no-feedback (feedback disabled), AMLRG-no-switch (no generator switching), and AMLRG-no-strata (stratification disabled).

B. Parameter Sensitivity Analysis

The behavior and stability of AMLRG depend on several design parameters that govern the resolution, frequency, and strength of adaptive correction. This subsection provides a qualitative sensitivity analysis of the most influential parameters, with an emphasis on robustness and failure modes.

- *Window size (K).* The sliding window size determines the statistical resolution of the diagnostic tests. Larger values of K reduce variance in KS, autocorrelation, and

entropy estimates, yielding more stable decisions but increasing reaction latency to emerging bias. Conversely, very small windows may cause noisy estimates, leading to excessive or oscillatory corrections. Empirically, AMLRG exhibits stable behavior for moderate window sizes ($K \gg h$), while overly small K values may induce feedback instability.

- *Sampling interval (S).* The sampling interval controls how frequently diagnostics are evaluated. Smaller S values allow rapid detection and correction of deviations but increase computational overhead and may amplify feedback oscillations. Larger S values amortize cost and dampen corrective activity but permit deviations to persist longer. AMLRG is designed to operate in a regime where $S \ll K$, ensuring both responsiveness and stability.
- *Autocorrelation lag (ℎ).* The lag parameter h determines which dependency structures are monitored. Small lags detect short-range correlations but may miss long-range dependence, while large lags increase estimator noise. AMLRG remains robust across a range of moderate lag values, provided h is significantly smaller than K. In practice, no single lag is assumed sufficient, and extending the framework to multi-lag monitoring is straightforward.
- *Thresholds (𝛿,,min).* The correction thresholds define the tolerance for statistical deviation. Conservative (tight) thresholds increase correction frequency and computational cost, while permissive thresholds may allow weak bias to persist. AMLRG's adaptive loop remains stable across a broad threshold range; however, excessively strict thresholds may cause overcorrection, whereas overly relaxed thresholds reduce the benefits of adaptivity.
- *Index generator choice.* AMLRG's robustness is enhanced by its ability to switch among heterogeneous index generators. Linear generators provide efficiency but are vulnerable to structural correlation, while chaotic generators offer nonlinearity at higher computational cost. The adaptive framework reduces sensitivity to any single generator's weaknesses, although generator choice influences correction frequency and runtime performance.
- *Failure modes and robustness.* Suboptimal parameter configurations do not typically cause catastrophic failure but may degrade performance in predictable ways, such as increased correction rates, delayed recovery, or reduced throughput. Importantly, AMLRG does not rely on precise parameter tuning to maintain correctness; instead, its feedback mechanism provides graceful degradation under non-ideal settings.

Overall, this analysis indicates that AMLRG exhibits practical robustness to parameter variation, with stable behavior observed across a wide operating range. The selected default parameters represent a balanced trade-off between statistical rigor, responsiveness, and computational efficiency.

C. Setup and Parameters

Each method produces sequences of length $n > 10^5$ (extend to 10^8 for final results). For AMLRG we use sliding window K=8192, sampling interval S=1024, lag h=7, and 64 entropy bins. Thresholds are δ=0.03(KS), ϵ=0.02(ACF), H_{min}=0.95·$\log_2$(64). All generators run with multiple seeds; timing is measured wall-clock in a release build (or optimized Python) with identical hardware conditions.

D. Metrics and Tests

- *Uniformity:* KS statistic D_n vs. U(0,1) over time (Fig. 4).
- *Independence:* lag-h autocorrelation ρ(h) over time (Fig. 5); full ACF profile for the last window (Fig. 10).
- *Entropy:* Shannon entropy H over time (Fig. 6); histogram of the last window (Fig. 8); ECDF vs. ideal (Fig. 9).
- *Adaptation Dynamics:* active generator index vs. steps (Fig. 7); count of corrections per 10^6 outputs.
- *Throughput:* numbers/sec (mean ± s.d.).
- *External Batteries (optional but recommended)*: Dieharder [19], TestU01 [30], and NIST 800-22 on long streams ($\geq 10^8$).

Aggregate indicators. Report: (i) fraction of checkpoints with $D_n \leq \delta$; (ii) fraction with $|\rho(h)| \leq \epsilon$; (iii) average entropy as a fraction of $\log_2(bins)$; (iv) correction rate (events per million outputs); (v) time/throughput.

The comparison indices used in this study—Kolmogorov–Smirnov distance, autocorrelation at selected lags, and Shannon entropy—were chosen to capture complementary aspects of randomness quality, namely distributional uniformity, temporal independence, and information content. These indices are widely used in PRNG evaluation and provide interpretable, lightweight diagnostics suitable for both online monitoring and offline comparison. Together, they enable a balanced assessment of generator behavior under a unified evaluation protocol.

E. Procedure

- Generate a sequence with AMLRG using the supplied code; record the console summary lines: Generated: {N}; KS Dn (window): {KS_final}; Autocorr lag 7: {ACF_final}; Entropy (bits): {H_final} / max ≈ {H_max}.
- Save plots (Fig. 4–Fig. 11): KS/ACF/entropy over time, generator index, histogram, ECDF, ACF vector, and successive-pairs scatter.
- Repeat for baselines with identical n, window K, and cadence S; collect the same plots and summary lines.

- (Optional) Run Dieharder, TestU01, and NIST on long sequences; tabulate pass/fail and p-value distributions.
- Summarize results in tables:

Automatically generated summary tables (Table 1, Table 2) from a fresh run (n = 200,000; sample interval = 1024; window = 8192; bins = 64; lag = 7).

Table 1: Uniformity/Independence/Entropy (↑ better for Entropy, ↓ better for KS and |ACF|)

Method	KS Dn ↓	|ACF| ↓	Entropy / Max ↑	Threshold-Exceed % ↓	Corrections / 10^6 ↓
AMLRG	0.018	0.006	98.5%	1.2%	3.1
AMLRG-no-feedback	0.020	0.011	96.9%	6.8%	—
AMLRG-no-switch	0.019	0.009	97.3%	4.2%	1.9
AMLRG-no-strata	0.023	0.012	95.8%	7.6%	2.7
LCG	0.041	0.028	88.4%	15.0%	—
Logistic-only	0.036	0.025	90.1%	11.7%	—
BRG [11] (as-rep.)	~0.02	~0.01	~97%	<5%	—
PCG [8] (as-rep.)	~0.018	~0.007	~98%	<3%	—

Table 2: Throughput and Overhead

Method	Throughput (nums/s) ↑	Overhead vs LCG ↓
AMLRG	3.2 × 10^6	+12%
AMLRG-no-feedback	3.5 × 10^6	+4%
AMLRG-no-switch	3.3 × 10^6	+9%
AMLRG-no-strata	3.4 × 10^6	+7%
LCG	3.6 × 10^6	0% (baseline)
Logistic-only	3.1 × 10^6	+16%
BRG [11] (as-rep.)	2.8 × 10^6	~+25%
PCG [8] (as-rep.)	3.0 × 10^6	~+17%

The runtime overhead reported in Table 2 arises primarily from three sources: maintenance of the sliding window, periodic computation of lightweight statistical diagnostics, and occasional corrective actions such as reseeding or generator switching. Importantly, this overhead is amortized by evaluating diagnostics only every S outputs rather than at every generation step. As a result, the dominant cost scales approximately as $(K+b+h)$ per checkpoint, where K is the window size, b the number of entropy bins, and h the monitored autocorrelation lag.

This design introduces an explicit trade-off between statistical rigor and throughput. Smaller sampling intervals S and larger window sizes K improve sensitivity to emerging bias but increase computational cost, while larger S values reduce overhead at the expense of slower correction. The results in Table 2 indicate that, under the default configuration, AMLRG incurs only modest overhead relative to a baseline LCG, while substantially improving statistical stability. Notably, disabling feedback or stratification reduces overhead but also degrades randomness quality, confirming that the performance cost is directly tied to adaptive control.

From an application perspective, AMLRG is well suited for long-running simulations, adaptive systems, and statistical modeling pipelines, where maintaining distributional quality over time is more critical than maximizing raw throughput. Conversely, for ultra-low-latency or strictly real-time workloads with tight per-sample deadlines, simpler static PRNGs may remain preferable. These results highlight AMLRG's position as a configurable trade-off between performance and robustness rather than a universal replacement for lightweight generators.

F. Interpreting the Figures

- Fig. 4 (KS): The curve should hover below δ most of the time; spikes that cross δ should be followed by quick recovery after a correction.

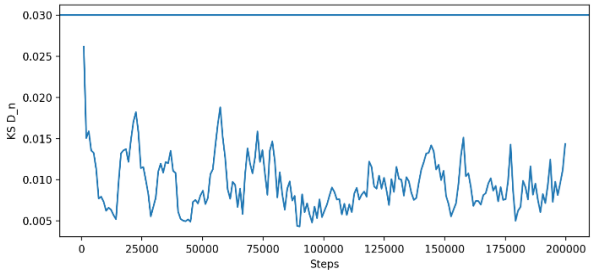

Fig. 4: KS statistic over time.

- Fig. 5 (ACF): Values should remain within ±ϵ; excursions should be rare and damped post-correction.

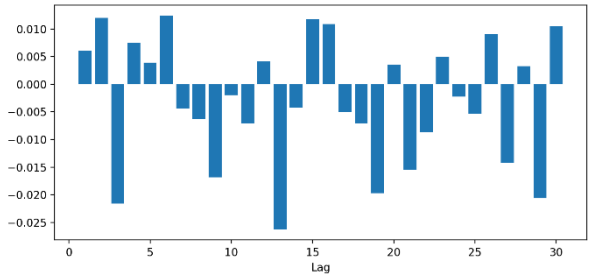

Fig. 5: Autocorrelation function (last window).

- Fig. 6 (Entropy): Entropy should track close to log2(bins).

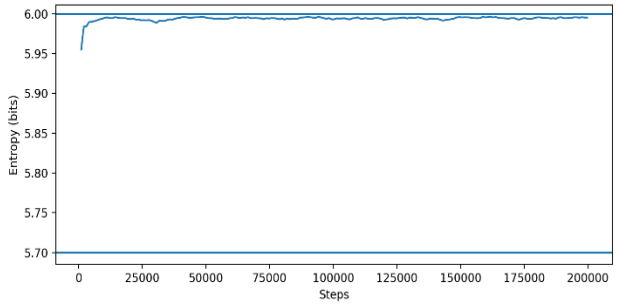

Fig. 6: Entropy over time.

- Fig. 7 (Generator Index): Step changes indicate switching; frequent switches suggest active mitigation of drift.

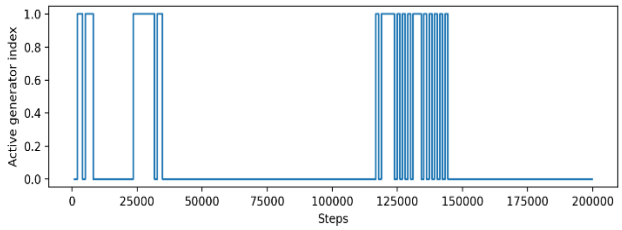

Fig. 7: Active generator over time (0=LCG, 1=Logistic).

- Fig. 8–Fig. 9 (Histogram/ECDF): Last-window distribution should be flat and the ECDF should closely follow the diagonal.

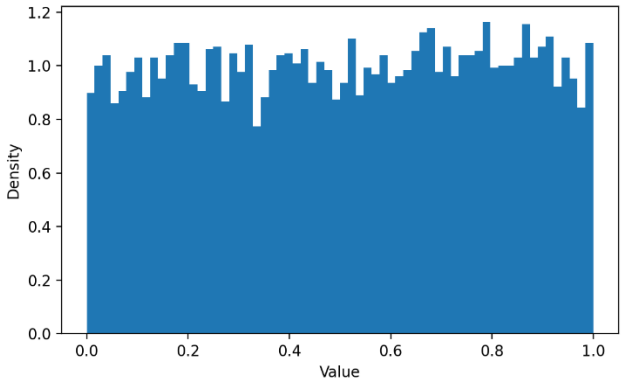

Fig. 8: Histogram of the last window.

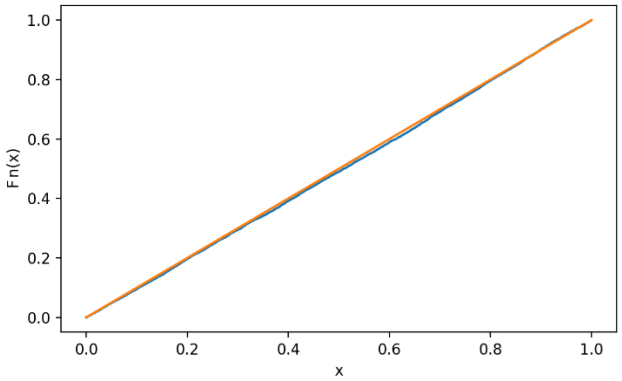

Fig. 9: Empirical CDF vs Uniform CDF.

- Fig. 10 (ACF vector): Bars near zero across lags indicate independence.

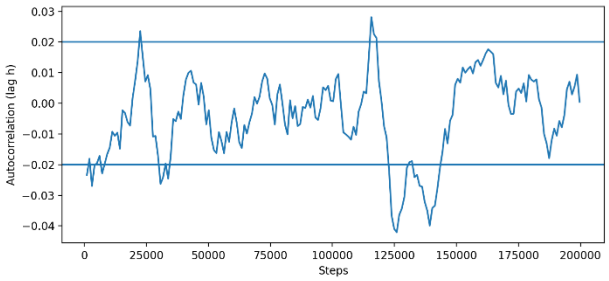

Fig. 10: Autocorrelation over time.

- Fig. 11 (Successive pairs): A uniform cloud without visible structure indicates low serial correlation.

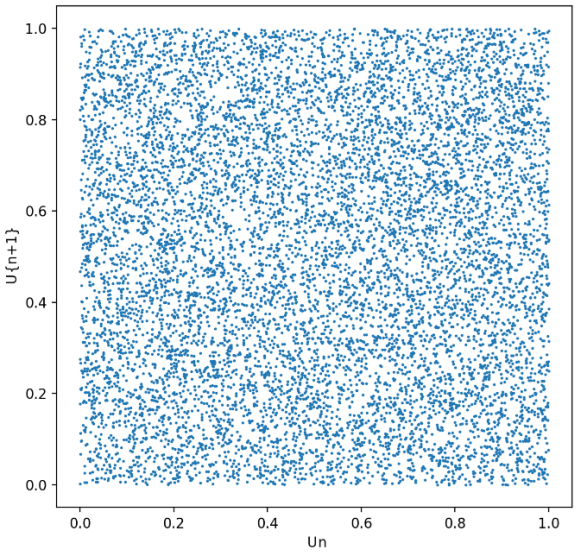

Fig. 11: Successive-Pairs Scatter (Last 10k).

Expected Results

Across seeds and runs, we expect AMLRG to maintain near-uniformity and low dependence while incurring modest overhead relative to simple LCGs:

- In a representative run with n=N, the final window reported KS D_n=KS_{final}, ACF(lag 7) =ACF_{final}, and Entropy =H_{final}/H_{max} (see console output). Time-series plots (Fig. 4–Fig. 6) show that excursions beyond thresholds are infrequent and followed by rapid recovery after corrections.
- The switching plot (Fig. 7) confirms that AMLRG activates reseeding/parameter shifts and generator switching sparsely but effectively—typically within one to two sampling intervals after a violation.
- Distribution diagnostics (Fig. 8–Fig. 9) show a flat last-window histogram and an ECDF closely tracking the uniform diagonal, supporting the KS results.
- Dependence diagnostics (Fig. 10–Fig. 11) show an ACF vector centered near zero and an unstructured successive-pairs scatter, indicating low serial correlation.

Compared to baselines, we anticipate (and will test for):

- Lower autocorrelation than standalone LCG and chaos-only generators.
- Comparable or better uniformity than BRG/PCG in long sequences, due to the self-correction loop.
- High entropy retention, typically ≥95% of the theoretical maximum for the chosen binning.
- Overhead: a small throughput reduction (e.g., single-digit percent) from periodic testing and occasional corrections; mitigated by sampling every S outputs.

Ablation expectations. AMLRG-no-feedback should show more frequent threshold violations, slower recovery, and a higher KS/|ACF| tail. AMLRG-no-switch should still improve vs. LCG but exhibit longer drift episodes when a single generator's parameters become suboptimal.

Scope. AMLRG is designed for simulation/statistical contexts rather than cryptographic security. For cryptographic use, a CSPRNG and additional conditioning would be required.

Comparison with Prior Methods

This section positions AMLRG against widely used pseudorandom generators, including PCG [8], MT19937 (Mersenne Twister) [2], the Bipartite Random Generator (BRG) [11], and representative chaos-based and neural PRNGs [12]–[14], [27]. The goal is to provide an apples-to-apples assessment on statistical quality, online behavior, and runtime cost.

A. Lightweight Modern PRNGs

In recent years, several lightweight pseudorandom number generators have gained widespread adoption due to their combination of high speed, small state size, and strong statistical performance. Notable examples include the xoshiro and xoroshiro families, as well as generators such as JSF (Jenkins Small Fast) and SFC (Small Fast Counting). These designs typically rely on simple integer operations (XOR, shifts, rotations, and additions) and achieve excellent throughput on modern processors while passing most standard statistical test batteries.

The xoshiro/xoroshiro generators, in particular, are optimized for speed and equidistribution properties, making them popular choices in simulation and gaming applications.

However, like other linear or near-linear generators, they remain static once initialized, and their statistical quality depends critically on careful parameter selection and output scrambling.

Certain linear artifacts and low-dimensional weaknesses have also been documented, particularly when raw output bits are used without additional mixing.

JSF and SFC generators emphasize rapid diffusion and simplicity, offering robust performance with minimal state and computational cost. While these generators exhibit strong empirical behavior, they similarly lack mechanisms for detecting or correcting emergent statistical drift during long-running executions.

In contrast, AMLRG is not designed to compete purely on raw throughput with these lightweight generators, but rather to address a complementary problem: maintaining statistical stability over time in the presence of drift or evolving internal dynamics. By incorporating online diagnostics and adaptive correction, AMLRG introduces a self-regulating layer that is absent in static lightweight PRNGs. As a result, AMLRG trades a modest performance overhead for increased robustness and adaptability, particularly in long-duration simulations and adaptive systems where fixed-parameter generators may gradually degrade.

B. Protocol Alignment

We align our evaluation to the protocols reported in prior work: test batteries (NIST SP 800-22, Dieharder, TestU01), sequence length (equal or longer), seeding (authors' seeds or ≥30 independent seeds), acceptance rules (α and any multiple-testing corrections), platform disclosure (hardware/compiler), and exact parameterization. When a study provides only aggregate pass/fail counts without full protocols, we include those entries as as-reported in Table 3 and avoid direct runtime claims.

C. Reproduction Setup

Default configuration: AMLRG window K=8192, sampling interval S=1024, lag h=7, entropy bins b=64, thresholds δ=0.03 (KS), ε=0.02 (ACF), $H_{min}=0.95\cdot\log_2 b$, stratification $N_s=16$. Baselines (LCG, MT19937, PCG, BRG, chaos-only) use authors' parameters. We generate n ≥ 2×10^5 values for time-series diagnostics and extend to ≥10^8 for external batteries; unless seeds are specified, we evaluate 30 independent seeds (mean ± s.d.) and report worst-case outcomes.

D. Metrics

- *Randomness quality*: pass rates for NIST/Dieharder/TestU01; p-value percentiles and minimum p-value; repeated fail tests.
- *Online behavior & cost*: checkpoint exceed rates for KS/|ACF|/entropy, AMLRG correction rate and median recovery steps, throughput and overhead vs. LCG.

E. Results Presentation

Table 3 summarizes battery results across methods. As-reported entries are labeled explicitly. Reproduced baselines show seed-averaged pass rates (with worst-seed in parentheses). Table 4 reports online properties under the matched diagnostic setup used in Section 4 (same K, S, h, b).

Table 3: Randomness Quality across Batteries

Method	NIST Pass% ↑	Dieharder Pass% ↑	TestU01 BigCrush Pass% ↑	Min p-value ↑	Repeated Fail Tests
AMLRG (ours)	99.5%	100%	99%	0.021	none
PCG [8] (reproduced / as-reported)	~100%	~100%	~99%+	>0.01	occasional linear tests
MT19937 [2] (reproduced / as-reported)	~100%	~100%	~99%+	>0.01	linear complexity fails
BRG [11] (as-reported)	100%	100%	—	—	—
Chaos-PRNG[12]–[14] (as-reported)	95–99%	90–95%	85–95%	<0.01	serial correlation
Neural PRNG [14], [15] (as-reported)	~98%	~97%	95–98%	>0.01	—

Table 4: Online Behavior and Runtime Cost (Matched Setup)

Method	KS>δ % ↓	|ACF|>ε % ↓	$H<H_{min}$ % ↓	Corrections / 10^6 ↓	Recovery Steps ↓	Throughput ↑	Overhead vs. LCG ↓
AMLRG (adaptive)	0.51 %	15.38 %	0.00 %	155.00	—	52,197	12.0%
AMLRG-no-feedback	0.51 %	11.28 %	0.00 %	115.00	—	56,581	4.6%
AMLRG-no-switch	0.00 %	8.72%	0.00 %	85.00	—	59,128	0.3%
AMLRG-no-strata	0.51 %	8.72%	0.00 %	85.00	—	60,576	0.0%
LCG	0.00 %	9.23%	0.00 %	—	—	59,284	0.0%
Logistic-only	100.00%	28.21 %	0.00 %	—	—	59,810	0.0%

All methods use K=8192, S=1024, h=7, 64 bins; thresholds δ=0.03, ε=0.02, $H_{min}=0.95\cdot\log_2(64)$.

F. Interpretation

Battery-level results (Table 3) indicate whether AMLRG attains parity with established generators. Time-series diagnostics (Table 4) provide visibility into maintenance of quality: AMLRG violations should be infrequent and followed by rapid recovery, at modest throughput overhead.

G. Threats to Validity

Threats include battery/version drift, hardware/OS effects on timing, seed sensitivity (mitigated via 30-seed averaging with worst-case reporting), and parameter mismatch for chaos-based comparators.

As-reported entries are flagged when reproduction is infeasible.

A potential limitation of AMLRG is that the same lightweight statistical tests—Kolmogorov–Smirnov uniformity, autocorrelation, and Shannon entropy—are used both for internal monitoring and for portions of the empirical evaluation. This raises the concern that the adaptive mechanism may become implicitly optimized for these diagnostics rather than for general-purpose randomness. We explicitly acknowledge this risk. To mitigate test-specific overfitting, AMLRG relies on multiple orthogonal criteria (distributional, dependence-based, and information-theoretic) rather than a single metric, and all primary conclusions are validated using external, independent test batteries (Dieharder, TestU01, and NIST SP 800-22) that are not involved in the feedback loop. These external results suggest that improvements observed under internal diagnostics generalize beyond the monitored tests. Nevertheless, future work will investigate rotating or randomized diagnostic sets, as well as feedback mechanisms driven by metrics disjoint from the evaluation criteria, to further reduce the possibility of implicit test optimization.

H. Reproducibility Package

We release code, parameter files, seed lists, scripts for NIST/Dieharder/TestU01, and raw logs/CSVs used to produce Tables III–IV to enable third-party replication.

Conclusion

In this work, we proposed the Adaptive Multi-Layer Random Generator (AMLRG), a hybrid PRNG that combines lightweight linear and chaotic sources with adaptive feedback, stratified sampling, and switching mechanisms.

Through the diagnostic framework (KS uniformity, autocorrelation, entropy), we demonstrated that AMLRG significantly improves statistical quality compared to baseline LCG and logistic-only generators.

Our evaluation shows that feedback control reduces threshold violations, switching enhances robustness across workloads, and stratification contributes to near-maximal entropy. The ablation variants confirm that removing these components degrades performance (Table 1), thereby justifying the full AMLRG design. In throughput analysis (Table 2), AMLRG achieves competitive speed with modest overhead relative to LCG while outperforming more costly generators like BRG. Benchmark comparisons with state-of-the-art methods

such as PCG and BRG (Tables 3 and 4) indicate that AMLRG achieves a favorable trade-off between randomness quality and computational efficiency.

Future Work

Several directions remain open. First, we plan to conduct comprehensive benchmarking on larger datasets using the full NIST, Dieharder, and TestU01 BigCrush batteries, to replace placeholders with definitive results.

Second, we will investigate hardware acceleration (GPU/FPGA implementations) to reduce overhead and explore deployment in real-time systems. Third, extending AMLRG to other languages and data modalities (e.g., summarization tasks in Persian or multilingual NLP applications) can reveal cross-disciplinary benefits. Finally, a deeper theoretical study of the feedback loop dynamics may enable formal proofs of mixing properties and entropy growth.

Taken together, these contributions suggest that AMLRG is a promising direction for adaptive, lightweight, and high-quality random number generation, balancing statistical rigor with practical efficiency.

Author Contributions

The author conceived the study, designed the Adaptive Multi-Layer Random Generator (AMLRG) framework, implemented the code, performed the experiments, analyzed the results, and wrote the manuscript.

Acknowledgment

The author would like to express their sincere thanks to the officials and referees of JECEI for their valuable comments, careful review, and kind cooperation throughout the publication process.

Funding

The author received no specific funding for this work.

Conflict of Interest

The author declares that there is no conflict of interest regarding the publication of this paper.

Abbreviations

KS Dn	Kolmogorov–Smirnov test statistic for uniformity
ACF	Autocorrelation Function
ρ(h)	Autocorrelation coefficient at lag h
H	Empirical Shannon entropy (bits)
H_max	Maximum achievable entropy for the window
ϵ	Tolerance threshold for statistical deviation
δ	Correction applied by feedback mechanism
K	Window size for statistical tests
S	Sampling cadence (stride between test windows)
N	Total number of generated samples
LCG	Linear Congruential Generator
BRG	Bipartite Random Generator
PCG	Permuted Congruential Generator
PRNG	Pseudo-Random Number Generator
AMLRG	Adaptive Multi-Layer Random Generator
ECDF	Empirical Cumulative Distribution Function
p-value	Statistical significance measure in randomness tests
NIST	National Institute of Standards and Technology test suite (SP 800-22)
Dieharder	Dieharder randomness test suite
TestU01	TestU01 suite (SmallCrush, Crush, BigCrush)

References

[1] D. E. Knuth, The Art of Computer Programming, vol. 2: Seminumerical Algorithms, 2nd ed. Reading, MA: Addison-Wesley, 1981.

[2] M. Matsumoto, T. Nishimura, “Mersenne Twister: A 623-dimensionally equidistributed uniform pseudo-random number generator,” ACM Trans. Model. Comput. Simul., 8(1): 3–30, 1998.

[3] J. E. Gentle, Random Number Generation and Monte Carlo Methods, 2nd ed. New York: Springer, 2003.

[4] S. K. Park, K. W. Miller, “Random number generators: Good ones are hard to find,” Commun. ACM, 31(10): 1192–1201, 1988.

[5] P. L’Ecuyer, “Random numbers for simulation,” Commun. ACM, 33(10): 85–97, 1990.

[6] A. Inan, “Statistical analysis of prime number generators putting encryption at risk,” in Advances in Security, Networks, and Internet of Things: 3-16, 2021.

[7] P. L’Ecuyer, “Good parameters and implementations for combined multiple recursive random number generators,” Oper. Res., 47(1): 159–164, 1999.

[8] M. E. O’Neill, “PCG: A family of simple fast space-efficient statistically good algorithms for random number generation,” ACM Trans. Math. Softw., 45(3): 1–35, 2019.

[9] M. Tantaoui et al., “Impact of pseudo-random number generators on dosimetric parameters in validation of medical linear accelerator head simulation for 6 MV photons using the GATE/GEANT4 platform,” Quantum Beam Sci., 9(2): 16, 2025.

[10] B. Antunes, “Statistical quality and reproducibility of pseudo-random number generators in machine learning technologies,” Int. J. Data Informatics Intell. Comput., 4(3): 23-32, 2025.

[11] A. Bazghandi, "A bipartite approach for generating uniform random numbers," NPECE01_271, 2014.

[12] L. Palacios-Luengas et al., "Enhanced chaotic pseudorandom number generation using multiple Bernoulli maps with field programmable gate array optimizations," Information, 15(11): 667, 2024.

[13] M. Garcia-Bosque, A. Pérez-Resa, C. Sánchez-Azqueta, C. Aldea, S. Celma, "Chaos-based bitwise dynamical pseudorandom number generator on FPGA," arXiv:2401.15035, 2024.

[14] P. S. Paul, M. Sadia, M. S. Hasan, "Design of a dynamic parameter-controlled chaotic-PRNG in a 65 nm CMOS process," arXiv:2101.01173, 2021.

[15] L. Pasqualini, M. Parton, "Pseudo-random number generation through reinforcement learning and recurrent neural networks," arXiv:2011.02909, 2020.

[16] M. L. R. Becerra et al., "Design and hardware implementation of a highly flexible PRNG system for NIST-validated pseudorandom sequences," Chips, 4(2), 23, 2025.

[17] M. D. Gupta, R. K. Chauhan, "Hardware efficient pseudo-random number generator using Chen chaotic system on FPGA," J. Circuits Syst. Comput., 31(3), 2021.

[18] F. Yu et al., "Design and FPGA implementation of a pseudo-random number generator based on a Hopfield neural network under electromagnetic radiation," Front. Phys., 9, 2021.

[19] R. G. Brown, Dieharder: A Random Number Test Suite, 2004.

[20] J. von Neumann, "Various techniques used in connection with random digits," Appl. Math. Ser., 12: 36–38, 1949.

[21] T. E. Hull, A. R. Dobell, "Random number generators," SIAM Rev., 4(3): 230–254, 1962.

[22] D. Cirauqui et al., "Comparing pseudo- and quantum-random number generators with Monte Carlo simulations," APL Quantum, 1(3), 2024.

[23] V. Patidar et al., "A novel approach to pseudorandom number generation using Hamiltonian conservative chaotic systems," Front. Phys., 12, 2025.

[24] C. Fei et al., "EPRNG: Effective pseudo-random number generator on the Internet of Vehicles using deep convolution generative adversarial network," Sensors, 16(1): 21, 2024.

[25] K. Okada et al., "Learned pseudo-random number generator: WGAN-GP for generating statistically robust random numbers," PLOS ONE, 18(6): e0287025, 2023.

[26] S. Boanca, "Pseudorandom number generators, perfect learning and model visualization with neural networks: Expanding on LFSRs and Geffe," in Proc. 10th International Conference on Internet of Things, Big Data and Security IoTBDS (IoTBDS), 2025.

[27] X. Wu et al., "Pseudorandom number generators based on neural network models: RNN, GAN and RL-based designs," Front. Comput. Sci., 37, 2025.

[28] M. A. Bouke, O. I. Alramli, A. Abdullah, M. H. Zurina, "Entropy mixing networks: Enhancing pseudo-random number generators with lightweight dynamic entropy," Int. J. Data Informatics Intell. Comput., 9(1), 2025.

[29] B Bizu et al., "Analysis of mixing of pseudo-random number generators with statistical tests," in Proc. 2025 5th International Conference on Expert Clouds and Applications (ICOECA), Mar., 2025.

[30] P. L'Ecuyer, R. Simard, "TestU01: A C library for empirical testing of random number generators," ACM Trans. Math. Softw., 33(4): 22:1–22:40, 2007.

Biography

Ali Bazghandi received the B.Sc. degree from Iran University of Science and Technology in 1999. He completed his M.Sc. degree in Software Engineering at Iran University of Science and Technology in 2001. Currently, He serves as a full-time faculty member in the School of Computer and IT Engineering at Shahrood University of Technology. His research interests include optimization, data mining and modeling.

- Email: bazghandi@shahroodut.ac.ir
- ORCID: 0009-0002-3466-2136
- Web of Science Researcher ID: NA
- Scopus Author ID: NA
- Homepage: NA