Transformers in Pseudo-Random Number Generation: A Dual Perspective on Theory and Practice

Ran Li, Lingshu Zeng*

School of Information Science and Technology, Northeast Normal University, Changchun, China
lir660@nenu.edu.cn, zengls188@nenu.edu.cn

Abstract

Pseudo-random number generators (PRNGs) are high-nonlinear processes, and they are key blocks in optimization of Large language models. Transformers excel at processing complex nonlinear relationships. Thus it is reasonable to generate high-quality pseudo-random numbers based on transformers. In this paper, we explore this question from both theoretical and practical perspectives, highlighting the potential benefits and implications of Transformer in PRNGs. We theoretically demonstrate that decoder-only Transformer models with Chain-of-Thought can simulate both the Linear Congruential Generator (LCG) and Mersenne Twister (MT) PRNGs. Based on this, we conclude that the log-precision decoder-only Transformer can represent non-uniform AC0. Our simulative theoretical findings are validated through experiments. The random numbers generated by Transformer-based PRNGs successfully pass the majority of NIST tests, whose heat maps exhibit clear statistical randomness. Finally, we assess their capability in prediction attacks.

Code — https://github.com/zeroDtree/transformer_prng

Extended version —
https://www.arxiv.org/abs/2508.01134

1 Introduction

Pseudo-random numbers are essential components in scientific and technological applications, from cryptography to statistical sampling and numerical computing. Modern applications require PRNGs that can generate high-quality random sequences. A PRNG uses a deterministic algorithm to produce reproducible sequences that appear random, based on an initial seed value. (Gentle 2003; Press 2007).

In recent years, Transformer-based large language models (LLMs) have shown exceptional performance across multiple domains. While these models excel in language processing and knowledge tasks, their theoretical foundations remain unclear. Specifically, how Transformers simulate complex nonlinear functions is still not fully understood.

Recent theoretical advances have provided significant insights into the expressive power of Transformers. Notably, it has been demonstrated that Transformers with constant depth and precision, when enhanced with Chain-of-Thought (CoT) prompting (Wei et al. 2022) of polynomial length $O(\text{poly}(n))$, possess sufficient expressive power to capture the complexity class P/poly (Theorem 3.3 in (Li et al. 2024)). This theoretical foundation suggests that Transformers might be particularly well-suited for processing nonlinear functions, with PRNGs serving as a classic and practical example of such functions.

The application of Transformers as PRNGs offers several key characteristics: (1) Transformer's advanced feature learning capabilities enable effective extraction and learning of complex statistical patterns in random sequences, (2) The sophisticated position encoding mechanisms systematically capture temporal dependencies and sequential relationships between numerical elements (Wu et al. 2025b), (3) The multi-head attention architecture facilitates parallel computation across diverse representation subspaces, enabling simultaneous analysis of multiple sequence characteristics. Building on these advantages, our research explores a fundamental question: What are the advantages of Transformers as PRNGs or tools for evaluating PRNGs security? To address this, we conduct a comprehensive study combining theoretical analysis with empirical validation of Transformer models' capabilities in pseudo-random number generation.

Our result This paper makes several significant contributions to understand the expressiveness of Transformer models in the context of PRNGs. Our work encompasses both theoretical analysis and empirical validation. Specifically, we demonstrate that Transformers can be utilized for prediction attacks, where in a Transformer model is trained on a sequence of generated pseudo-random numbers to predict subsequent values, thereby providing a mechanism to assess the security of the PRNGs.

First, we provide a comprehensive theoretical analysis of Transformer models' capability to express two widely-used PRNGs: the Linear Congruential Generator (LCG) and the Mersenne Twister (MT) algorithm. We demonstrate that Transformer architectures equipped with CoT [1] can precisely simulate these PRNGs using only a constant number of attention heads and layers, while maintaining $O(n)$

*Corresponding Author.

[1] In this paper, we define Chain of Thought as a step-by-step reasoning process, aligning with the interpretation in (Feng et al. 2024), rather than as a specialized prompting technique.

parameter complexity in the hidden layer (Theorem 5 and Theorem 6). Our proofs are constructive in nature, providing explicit mechanisms for simulating each computational step of both algorithms. To validate these theoretical findings, we conduct comprehensive experiments across multiple bit-widths (8-bit, 12-bit, and 16-bit) for pseudo-random number generation based on the MT algorithm, achieving near-perfect accuracy in all cases. Significantly, during this analysis, we establish a non-uniform AC^0 lower bound for Decoder-only Transformers with $\log(n)$ precision (Theorem 7),contributing to the theoretical understanding of Transformer limitations.

Second, we perform a comprehensive empirical evaluation of Transformer-based pseudo-random number generators (PRNGs) through three critical experiments:

- We investigate the potential of Transformer architectures as novel implementations for PRNGs. Through empirical analysis utilizing heat map visualizations, we clearly observe that sequences generated by Transformers exhibit significant statistical randomness properties.
- To rigorously analyze the capabilities of Transformers as pseudo-random number generators, we focus on their inherent non-linear transformations, which enhance the unpredictability of the generated sequences and potentially improve security. Consequently, we implement a Transformer model that simulates the MT algorithm for generating 16-bit pseudo-random numbers. The generated sequences successfully pass the majority of NIST statistical test suites.
- While Transformers can effectively learn complex statistical patterns from large-scale data, this capability has significant security implications for PRNGs, as their security fundamentally depends on the unpredictability of output sequences. Leveraging this insight, we propose utilizing Transformer models for security analysis of PRNGs through prediction attacks. Specifically, we investigate whether a Transformer model, after training on a sequence of generated pseudo-random numbers, can accurately predict subsequent values. Our experimental results demonstrate that Transformers has the potential to as a tool for evaluating PRNGs security by identifying potential statistical vulnerabilities in their output sequences.

To the best of our knowledge, this work represents the first comprehensive study of Transformer models in the context of pseudo-random number generation, establishing both theoretical foundations and practical viability. Our findings explore new possibilities of PRNGs based on transformer.

2 Preliminaries

We introduce the key notations used throughout the paper: n denotes the number of tokens in a sample sequence, d represents the dimension of embedding vectors, H is the number of attention heads in the model, $d_k = \frac{d}{H}$ defines the dimension per attention head, and L indicates the number of layers in the Transformer model. boldface letters are used to denote vectors, and normal letters are used to denote scalars. Unless otherwise specified, all vectors are represented as row vectors.

2.1 Decoder-only Transformers

Transformer models, introduced in 2017 (Vaswani 2017), revolutionized natural language processing through their self-attention mechanism, which effectively captures long-range sequence dependencies.

The auto-regressive Transformer, also known as the Decoder-Only Transformer (Radford et al. 2019), is a sequence-to-sequence neural network model as follows defined by the following equations: first, the input tokens s_i are embedded into a d-dimensional vector $\mathbf{x}_i^0 = \text{Embed}(s_i) + \text{pos}(i) \in \mathbb{R}^d$ for $i = 1, \ldots, n$, where $pos(i)$ is the position embedding of the i-th token. Then, the Transformer model processes the sequence of vectors through a series of attention layers. the l-th layer of the Transformer is defined as

$$\mathbf{x}_i^l = \text{Attention}^l(\mathbf{x}_i^{l-1}) + \text{FFN}^l(\text{Attention}^l(\mathbf{x}_i^{l-1})) \quad (1)$$

$$\text{Attention}^l(\mathbf{x}_i^{l-1}) = \text{Attention}^l(\mathbf{x}_i^{l-1}) + \mathbf{x}_i^{l-1} \quad (2)$$

$$\text{FFN}^l(\mathbf{x}) = \text{GeLU}(\mathbf{x}\mathbf{W}_1^l + \mathbf{b}_1^l)\mathbf{W}_2^l \quad (3)$$

where Attention is the residual multi-head attention layer and FFN is the feed-forward network, and $\mathbf{W}_1^l, \mathbf{W}_2^l \in \mathbb{R}^{d\times d}$, $\mathbf{b}_1^l, \mathbf{b}_2^l \in \mathbb{R}^d$ are the weights and biases of the l-th layer. $\text{Attention}^l(\mathbf{x}_i^{l-1})$ is given by

$$\sum_{h=1}^{H} \text{softmax}\left(\frac{\mathbf{q}_i^{(l,h)}(\mathbf{K}^{(l,h)})^T}{\sqrt{d_k}}\right)\mathbf{V}^{(l,h)}$$

where $\mathbf{W}_q^{(l,h)}, \mathbf{W}_k^{(l,h)}, \mathbf{W}_v^{(l,h)} \in \mathbb{R}^{d\times d_k}$ are the weights of the query, key, and value vectors, $\mathbf{q}_i^{(l,h)} = \mathbf{x}_i^{l-1}\mathbf{W}_q^{(l,h)}$, $\mathbf{K}^{(l,h)} = \mathbf{X}^{l-1}\mathbf{W}_k^{(l,h)}$, $\mathbf{V}^{(l,h)} = \mathbf{X}^{l-1}\mathbf{W}_v^{(l,h)} \in \mathbb{R}^{d_k}$ are the query, key, and value vectors, $\mathbf{X}^{l-1} = [(\mathbf{x}_1^{l-1})^T, \ldots, (\mathbf{x}_n^{l-1})^T]^T \in \mathbb{R}^{n\times d}$ is the matrix of the key vectors.

2.2 Pseudo-random Number Generator

Random Number Generation can be categorized into two main types (Smid et al. 2010): True Random Number Generators (TRNGs) generate unpredictable sequences using physical entropy sources, while PRNGs are deterministic algorithms that approximate randomness based on initial seed values. For cryptographic security, PRNG seeds must come from TRNGs.

The linear Congruential Generator The linear congruential generator is a simple and fast PRNG (MacLaren 1970). defined as follows:

$$\mathbf{x}_{n+1} = (\mathbf{a}\mathbf{x}_n + \mathbf{c}) \bmod m \quad (4)$$

where $\mathbf{a}, \mathbf{c} \in \mathbb{Z}^d$, $m \in \mathbb{Z}$, and parameters $\mathbf{a}$ and m determine the sequence period length. $\mathbf{x}_0$ is the initial seed and $\mathbf{x}_i$ is the i-th generated number.

The Mersenne Twister Algorithm The Mersenne twister algorithm is mentioned by Makoto Matsumoto and Takuji Nishimura in 1998 (Matsumoto and Nishimura 1998), improved upon previous PRNGs by offering high-quality random numbers with a long period ($2^{19937} - 1$) and fast generation speed.

The MT algorithm is defined as two parts: first, the initial values $\mathbf{x}_0, \ldots, \mathbf{x}_{n-1}$ are generated using LCG, $\mathbf{x}_i \in \{0,1\}^d$. Then the subsequent values are generated through the following recurrence relation including the rotation and extraction operations:

Rotation The rotation operation rotates the variables to generate the next state of the MT algorithm.

$$\mathbf{t} \leftarrow (\mathbf{x}[i] \wedge \mathbf{upper}) \vee (\mathbf{x}[(i+1) \bmod n] \wedge \mathbf{lower}) \tag{5}$$

$$\mathbf{z} \leftarrow \mathbf{x}[(i+m) \bmod n] \oplus (\mathbf{t} >> 1) \oplus \begin{cases} \mathbf{0} & \text{if } t_0 = 0 \\ \mathbf{a} & \text{otherwise} \end{cases} \tag{6}$$

where $\mathbf{upper} = \mathbf{1} << (d - r)$ and $\mathbf{lower} = \neg\mathbf{upper}$,$\mathbf{a}$ are constants vectors.

Extraction The extraction operation generates the next pseudorandom number through a series of bit-wise operations.

$$\mathbf{x}[i] \leftarrow \mathbf{z} \tag{7}$$
$$\mathbf{y} \leftarrow \mathbf{x}[i] \tag{8}$$
$$\mathbf{y} \leftarrow \mathbf{y} \oplus (\mathbf{y} >> u) \tag{9}$$
$$\mathbf{y} \leftarrow \mathbf{y} \oplus ((\mathbf{y} << s) \wedge \mathbf{b}) \tag{10}$$
$$\mathbf{y} \leftarrow \mathbf{y} \oplus ((\mathbf{y} << t) \wedge \mathbf{c}) \tag{11}$$
$$\mathbf{y} \leftarrow \mathbf{y} \oplus (\mathbf{y} >> l) \tag{12}$$

where u, l, s, t are constants, $\mathbf{b}, \mathbf{c}$ are constant vectors. The above bit-wise operations generate the next pseudorandom number, operating on each bit individually.

2.3 Circuit Complexity

We begin by introducing several fundamental circuit complexity classes, arranged in order of increasing computational power. For a comprehensive treatment, we refer readers to (Arora and Barak 2009).

- TC^{k-1} comprises languages computed by circuits with $O(\log^{k-1} n)$ depth, polynomial size, and unbounded fan-in MAJORITY gates (with NOTs available at no cost). A MAJORITY gate outputs the most frequent input value, defaulting to 1 in case of ties. Of particular interest is the class TC^0.
- $\text{AC}^{k-1}[m]$ consists of languages computed by constant-depth, polynomial-size circuits with unbounded fan-in, using AND, OR, and MOD m gates. A MOD m gate outputs 1 if and only if the sum of its input bits is divisible by m. While NOT operations can be simulated using MOD m gates, it remains an open question whether AND (or OR) operations can be simulated using only MOD m gates in constant depth.
- $\text{ACC} = \bigcup_m \text{AC}^0[m]$, representing the union of all $\text{AC}^0[m]$ classes.
- AC^{k-1} is defined as $\text{AC}^{k-1}[m]$ without MOD m gates.

These circuit complexity classes form a strict hierarchy, characterized by the following containment relations:

$$\text{AC}^0 \subseteq \text{AC}^0[m] \subseteq \text{ACC} \subseteq \text{TC}^0$$

Circuit classes can be non-uniform, using different algorithms for different input lengths, or uniform, using a single algorithm for all inputs. In the uniform model, an efficient algorithm can generate the appropriate n-th circuit for any n-bit input, independent of input length. While uniform versions exist for all circuit complexity classes, this paper focuses on the non-uniform class AC^0.

3 PRNGs via Transformer

We adopt a widely used and realistic setting known as the log-precision Transformer (Merrill and Sabharwal 2023b; Liu et al. 2022), in which all parameters of the Transformer are constrained to $O(\log(n))$ bit precision, where n represents the maximum length of the input sequence

The proofs of both theorems are constructive, providing explicit implementations of PRNGs using Transformer architectures. Our approach centers on utilizing the fundamental components of the Transformer architectures, specifically the attention mechanism and feed-forward network (FFN) layers—to systematically realize essential computational operations. These operations encompass both logical operations (XOR, NOT, AND, OR) and arithmetic operations (multiplication, addition, modulus) performed on pseudo-random numbers and constants. Our construction methodology extends the theoretical foundations established by (Feng et al. 2024) and (Yang et al. 2024). Particularly their seminal work on the function approximation capabilities of FFN.

It is noteworthy that the complete constructive proofs for Lemma 1-4, Theorem 5, and Theorem 6 are provided in the extended version.

3.1 Main Theoretical Results

We first present the implementation of fundamental Boolean operations using Transformer architecture, where $\phi(i)$ denotes the binary representation of the i-th number.

Lemma 1. *For a given variable vector $(\phi(i), \phi(j), i, 1)$, there exists a Transformer layer that approximates function $f(i,j)$ with a constant $\epsilon > 0$ such that $\|f(i,j) - \phi(i) \wedge \phi(j)\|_\infty \leq \epsilon$. The parameters of this layer exhibit an ℓ_∞ norm bounded by $O(poly(M, 1/\epsilon))$, where M denotes the upper bound for all parameter values.*

Lemma 2. *For a given variable vector $(\phi(i), \phi(j), i, 1)$, there exists a Transformer layer that approximates function $f(i,j)$ with a constant $\epsilon > 0$ such that $\|f(i,j) - \phi(i) \vee \phi(j)\|_\infty \leq \epsilon$. The parameters of this layer exhibit an ℓ_∞ norm bounded by $O(poly(M, 1/\epsilon))$, where M denotes the upper bound for all parameter values.*

Lemma 3. *Given a variable vector $(\phi(i), i, 1)$, there exists a attention layer that can approximate the boolean function $\neg\phi(i)$.*

And we use a two-layer MLP to approximate the modulus function, which is a key operation in the LCG and MT algorithms.

Lemma 4. *Given a positive number n, and an integer $i \in [1, n^2]$, there exists a two-layer MLP $f : \mathbb{R}^2 \to \mathbb{R}$ with GeLU activation, and the hidden dimension is $O(n)$. Then there exists a constant $\epsilon > 0$ such that $\|f(i, n) - (i \bmod n)\|_\infty \leq \epsilon$ and parameters with ℓ_∞ norm upper bounded by $O(poly(M, 1/\epsilon))$, where M is an upper bound for all parameter values.*

It is worth noting that the above lemma can be used not only in the proofs of the LCG and MT algorithms, but also as a tool and gadget for proving the expressive power of the Transformer. And building on the aforementioned lemmas, we can utilize the Transformer module to implement the specific steps of both the LCG and MT algorithms, thereby completing our proof.

Theorem 5. *For the linear congruential generator algorithm, we can construct an autoregressive Transformer as defined in Section 2 that is capable of simulating and generating n pseudo-random numbers. The constructed Transformer has a hidden dimension of $d = O(n)$, consists of **one layer** with **one attention head**, and all parameters are polynomially bounded by $O(poly(n))$.*

Proof Sketch. We utilize the attention module of the Transformer to implement the multiplication of the pseudo-random number x_{i-1} by the constant a (as established in the lemma). This result is then added to the constant c, followed by the application of the FFN module to perform the modulus operation $\mod(m)$. □

In Theorem 6, we employed chain of thought to enable the Transformer to simulate the intermediate steps of the MT algorithm sequentially.

Theorem 6. *For the Mersenne Twister algorithm, we can construct an autoregressive Transformer with Chain of Thought as defined in Section 2 that is capable of simulating and generating n pseudo-random numbers. The constructed Transformer has a hidden dimension of $d = O(n)$, consists of **seventeen layers** with at most **four attention heads per layer**, and all parameters are polynomially bounded by $O(poly(n))$.*

Proof Sketch. Initially, we employ an attention layer to compute the total count of generated pseudo-random numbers, denoted as $cnt_\Rightarrow$. Subsequently, utilizing this $cnt_\Rightarrow$ in conjunction with the attention layer, we determine the requisite variable indices for generating new pseudo-random numbers in the MT algorithm. Based on these indices, we retrieve the necessary random numbers and implement the rotation and extraction operations characteristic of the MT algorithm using a multi-layer Transformer architecture. Finally, we implement an MLP to distinguish between symbolic output and pseudo-random number generation. □

We can also simulate basic Boolean operations using Transformers, demonstrating that log-precision autoregressive Transformers can express non-uniform AC^0.

Corollary 7. *Log-precision autoregressive Transformers with polynomial size can simulate any circuit family in non-uniform AC^0. Specifically, for any circuit family $\{C_n\}_{n\in\mathbb{N}}$ in AC^0 with size poly(n), there exists a family of log-precision decoder-only Transformers $\{T_n\}_{n\in\mathbb{N}}$ of size poly(n) that can simulate $\{C_n\}_{n\in\mathbb{N}}$.*

Proof. Leveraging Theorem 1 and Theorem 2, we demonstrate that a polynomial-sized Transformer layer can simulate the fundamental circuit gates AND and OR in AC^0. This result establishes that log-precision autoregressive Transformers possess at least the expressive power of non-uniform AC^0. □

In the aforementioned theorems, we provided detailed constructions using Transformer architecture to implement each step of both the LCG and MT pseudo-random number generation algorithms, thereby establishing a theoretical foundation for Transformers' capability to simulate some pseudo-random generators. Subsequently, in Section 4, we validate our theoretical findings through comprehensive empirical evaluation, designing experiments to assess both the effectiveness of Transformers as pseudo-random number generators and their potential in prediction-based attack scenarios.

4 Experiments

Although we analyze the MT algorithm and the LCG can be simulated by transformer with constant depth in previous section, it is still important to verify the effectiveness of our theoretical analysis by experiments. In this section, we will design several experiments. First, we will verify the effectiveness of our theoretical analysis on the MT algorithm. Then, we find that generator based on transformer can pass the most statistical tests designed by NIST. Finally, we explore how transformers can be used for prediction attacks, such as predicting the output of the MT algorithm. All experiments related to training and sequence generation were conducted on a machine running the RokeyLinux operating system, equipped with an A800 GPU with 80 GB of memory. The NIST statistical tests were performed separately on a personal laptop computer. All data preprocessing and training code is available in our GitHub repository.

4.1 Transformer Simulation of Mersenne Twister

Model and Dataset We use the MT 19937 algorithm to generate 32bit pseudo-random numbers(mt19937-32) as our base dataset. The numbers were subsequently converted into 8-bit, 12-bit, and 16-bit representations using modular arithmetic operations. The dataset was then split into training and test sets at a 9:1 ratio. A fixed random seed of 31 was used for all experiments. Training was conducted for 50 epochs, and the test accuracy was adopted as the evaluation metric. The detailed dataset configurations are presented in Table 1.

To evaluate the simulation capability, we trained a GPT-2 architecture on the aforementioned datasets. The model

Dataset	Sequence Length	Training Set Size
mt19937-8bit	256	256
mt19937-12bit	4096	4096
mt19937-16bit	4096	4096

Table 1: Dataset configuration

was trained to minimize the cross-entropy loss, with training accuracy serving as our primary metric for assessing the quality of PRNG simulation. We conducted extensive experiments with various hyperparameter configurations, and the detailed model architecture specifications are summarized in Table 2.

Parameter	Value	Description
attn_pdrop	0.1	Dropout for attention layers
embd_pdrop	0.1	Dropout for embeddings
model_type	gpt2	Architecture type
n_embd	768	Embedding dimension
n_head	12	Attention heads
n_layer	12	Transformer layers
n_positions	1024	Max sequence length
resid_pdrop	0.1	Dropout for residuals
vocab_size	50257	Vocabulary size

Table 2: Model configuration

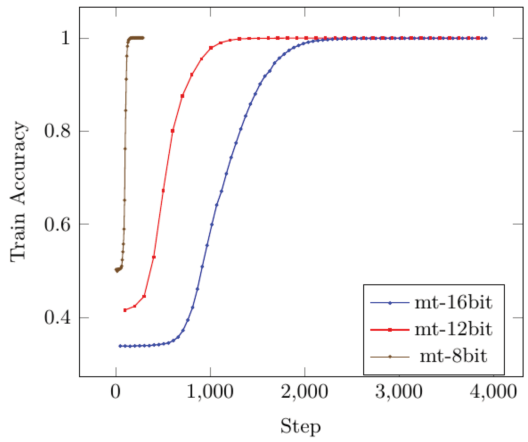

Figure 1: Training Accuracy

Results The experimental results demonstrate consistent convergence patterns across different bit-width configurations (8-bit, 12-bit, and 16-bit) of the MT simulation. As shown in Figure 1, the training accuracy curves exhibit sigmoid-like behavior, with the 8-bit model converging most rapidly (within 500 steps), followed by the 12-bit model (approximately 1,000 steps), and the 16-bit model requiring the longest training period (around 2,000 steps) to achieve convergence. Correspondingly, the training loss curves display

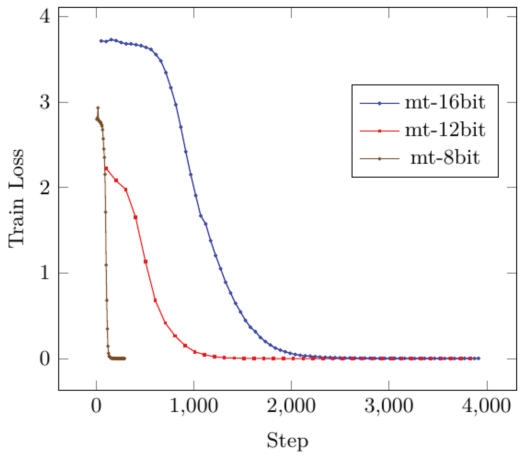

Figure 2: Training Loss

exponential decay, with all three models eventually stabilizing at minimal loss values, indicating successful model optimization. Notably, all configurations ultimately achieve near-perfect accuracy (approximately 1.0), suggesting that the Transformer architecture can effectively simulate the MT algorithm regardless of bit-width, albeit with varying convergence rates inversely proportional to the complexity of the target bit-width.

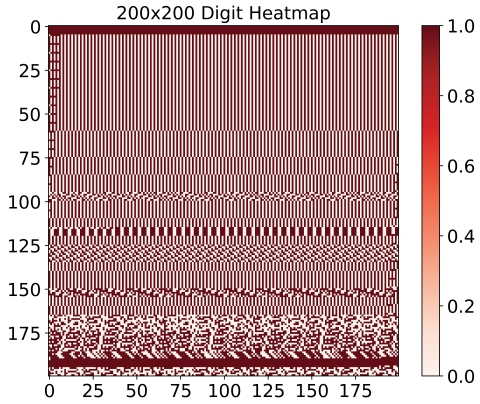

Figure 3: Heatmap of bit stream before training

4.2 NIST Statistical Tests for Randomness

NIST Statistical Test Suite We evaluated our pseudo-random sequences using the NIST Statistical Test Suite (NIST STS) (Smid et al. 2010), which includes 15 tests: Frequency, Block Frequency, Cumulative Sums, Runs, Longest Run of Ones, Rank, Discrete Fourier Transform, Nonperiodic Template Matchings, Overlapping Template Matchings, Universal Statistical, Approximate Entropy, Random Excursions, Random Excursions Variant, Serial, and Linear Complexity. Each test evaluates specific statistical properties using rigorous mathematical principles to detect poten-

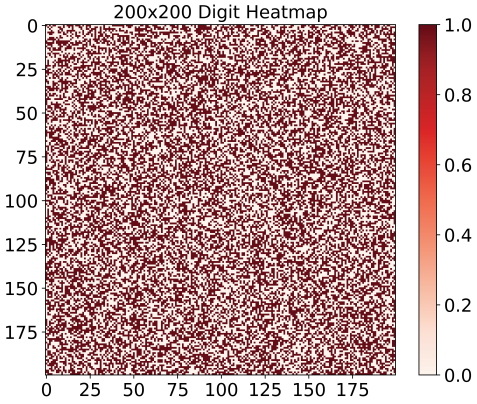

Figure 4: Heatmap of bit stream after training

tial non-random patterns and statistical deviations in binary sequences.

Our experimental methodology involved training a Transformer model on a 16-bit dataset, with model checkpoints systematically preserved at training accuracies thresholds of 0.63, 0.68, 0.81, and 0.89. The pseudo-random sequences generated from each checkpoint were subsequently subjected to the complete NIST STS battery of tests.

Our NIST testing passed 11 of 15 tests (corresponding to training accuracies of 0.63, 0.68, 0.81, and 0.89, respectively). The results of the NIST Statistical Test Suite that passed, as illustrated in Figure 5, indicate that the generated pseudo-random sequences exhibit satisfactory statistical properties. Specifically, All tests, except for the Block Frequency test, produced p-values greater than the significance level of 0.01, indicating that—with a 1% threshold for rejecting randomness—these sequences exhibit no statistically significant deviation from ideal randomness, suggesting that the sequences successfully passed the majority of the statistical tests. This outcome implies that the Transformer model effectively generates sequences that maintain randomness characteristics, thereby validating the robustness of the simulation process employed in this study.

The test failures reveal several limitations in our generator: RandomExcursions and RandomExcursionsVariant tests show insufficient random walk characteristics and missing P-values due to inadequate sequence variations; the Universal Test fails due to sequence compressibility issues from pattern reproduction in training data; and the Approximate Entropy Test fails due to limited sequence complexity caused by fixed-length training sequences.Notably, these results just achieved using merely a basic GPT-2 architecture, highlighting the considerable potential of Transformer-based approaches in pseudo-random number generation.

The heatmap presented in Figure 4 illustrates the transformation of the generated pseudo-random sequences before and after the training process. Figure 3 reveals a distinct pattern characterized by regularity and structure, indicative of a non-random distribution of values. Conversely, Figure 4 demonstrates a significant shift towards randomness, as evidenced by the chaotic and uniform distribution of values across the grid. This stark contrast underscores the effectiveness of the training regimen, which successfully enhances the stochastic properties of the output sequences, thereby validating the capability of the Transformer model to approximate true randomness in the generated bit stream.

4.3 Prediction Attack on Mersenne Twister

Building upon (Tao et al. 2025)'s findings on Transformers' ability to utilize prime factorizations and RNS representations in LCG sequence prediction, we employed a Transformer-based architecture to model and predict the output sequences of the Mersenne Twister (MT) algorithm. The experimental setup involved training the model with both 8-bit and 12-bit random number sequences, utilizing training-to-test set ratios of 1:10 and 1:20, respectively. Our empirical results demonstrate that the Transformer model achieves prediction accuracy ranging from 0.7 to 0.8 when inferring MT algorithm outputs. The detailed performance metrics and convergence characteristics are illustrated in Figure 6. These results not only advance our understanding of Transformer capabilities in sequence prediction but also provide valuable insights for developing more secure cryptographic algorithms.

5 Related Work

Our work investigates the expressive power and limitations of Transformers, with a specific focus on pseudo-random number generators. Transformer expressiveness research evolved in two main phases. Initially, (Yun et al. 2019) proved Transformers' universal approximation capabilities for continuous sequence functions, later extended to Sparse (Yun et al. 2020) and Linear Transformers (Alberti et al. 2023). Recent work focuses on in-context learning, with (Garg et al. 2022; Dai et al. 2022; Von Oswald et al. 2023) demonstrating basic function learning, while (Feng et al. 2024) showed Chain of Thought (Wei et al. 2022) capabilities for linear equations and dynamic programming. Further studies revealed their ability to solve P-class (Merrill and Sabharwal 2023a) and P/poly problems (Li et al. 2024). For a complete review, see (Yang et al. 2024).

From the perspective of constructing PRNGs, researchers have extensively explored various machine learning architectures to develop novel generators. These approaches can be categorized into three main frameworks: (1) Recurrent architectures have been widely adopted, leveraging their sequential processing capabilities. These include LSTM-based approaches that excel at capturing long-term dependencies (Jeong et al. 2018; Pasqualini and Parton 2020b; Jeong et al. 2020), traditional RNNs for basic sequence generation (Desai et al. 2012; Desai, Patil, and Rao 2012), Elman networks for their feedback mechanisms (Desai, Deshmukh, and Rao 2011), and Hopfield networks known for their stability properties (Hameed and Ali 2018; Lin et al. 2023; Bao et al. 2024). (2) Generative approaches using GANs have demonstrated promising results by employing adversarial training to improve random number quality (De Bernardi,

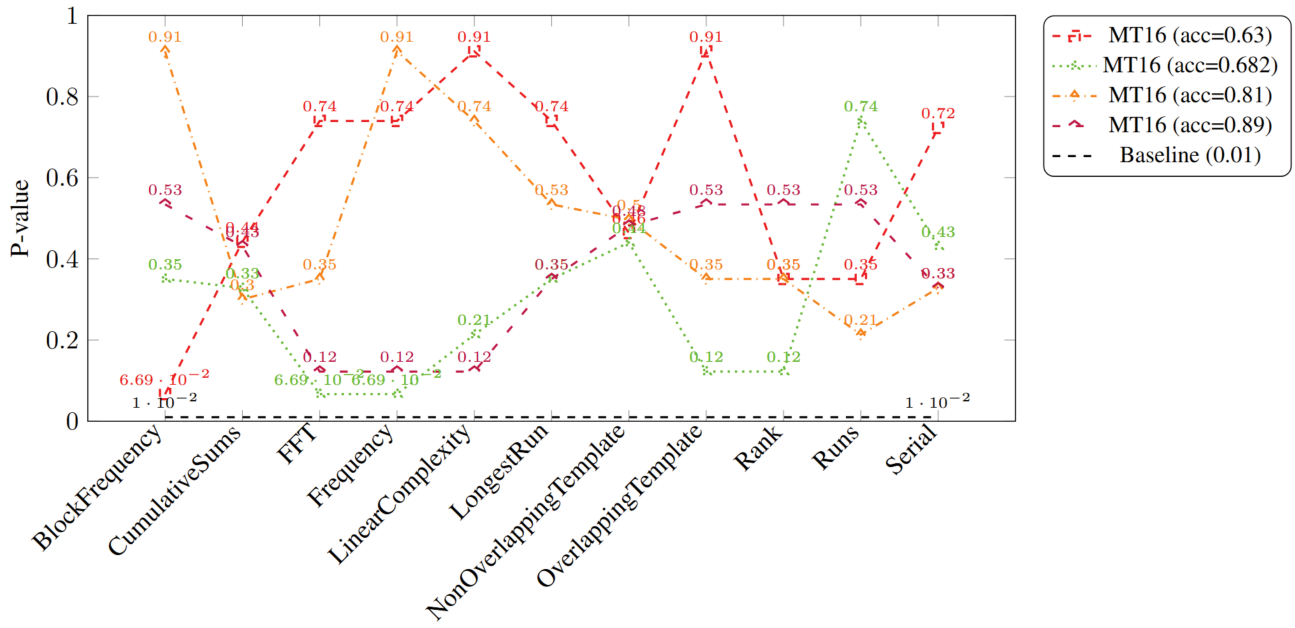

Figure 5: P-values for Different Statistical Tests

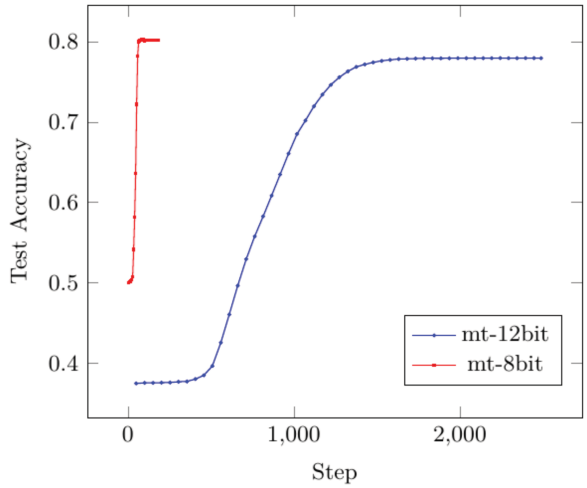

Figure 6: Prediction Accuracy

Khouzani, and Malacaria 2019; Oak, Rahalkar, and Gujar 2019; Wu et al. 2025a; Okada et al. 2023). The generator-discriminator architecture helps ensure the generated numbers exhibit desired statistical properties. (3) Reinforcement learning-based methods have emerged as an innovative approach, using reward mechanisms to optimize random number generation (Pasqualini and Parton 2020a,b; Park et al. 2022; Almardeny et al. 2022). These methods adapt their generation strategies based on feedback about the quality of produced sequences. For a comprehensive review of these approaches and their comparative advantages, see (Wu et al. 2025b).

Note that we use transformer model to simulate specific step of LCG and MT PRNGs. Our approach differs from previous work often to simulate the abstract model such as simulating the Turing machine (Merrill and Sabharwal 2023a) to solve P class problems and simulating the circuits (Li et al. 2024) to solve P/poly class problems. It is similar to the proof technique of (Feng et al. 2024).

6 Conclusion and Future Directions

Our theoretical analysis and experimental results indicate that the non-linear transformations and robust sequence modeling capabilities inherent in Transformer architectures can be effectively leveraged for both PRNGs construction and security analysis. Specifically, our experiments reveal that Transformer-based generators can produce high-quality pseudo-random sequences that satisfy the majority of NIST statistical criteria. Furthermore, the generated sequences demonstrate resistance to prediction attacks, suggesting that Transformers have potential applications both as pseudo-random number generators and as tools for evaluating PRNGs security.

Future research on Transformer-based PRNGs will explore three key directions : (1) Enhancing Transformers' capabilities to generate more secure and efficient pseudo-random numbers by leveraging their parallel architecture, (2) Developing Transformer-based frameworks for comprehensive PRNG security testing and evaluation, and (3) Theoretically analyzing Transformers' boundaries in fitting highly nonlinear functions, particularly identifying which types of functions remain beyond their computational reach.

Acknowledgements

We sincerely thank Xiongxin Yang and Meng Tan for their invaluable suggestions, and Professor Zhiguo Fu for his guidance.

References

Alberti, S.; Dern, N.; Thesing, L.; and Kutyniok, G. 2023. Sumformer: Universal approximation for efficient transformers. In *Topological, Algebraic and Geometric Learning Workshops 2023*, 72–86. PMLR.

Almardeny, Y.; Benavoli, A.; Boujnah, N.; and Naredo, E. 2022. A reinforcement learning system for generating instantaneous quality random sequences. *IEEE Transactions on Artificial Intelligence*, 4(3): 402–415.

Arora, S.; and Barak, B. 2009. *Computational complexity: a modern approach*. Cambridge University Press.

Bao, B.; Tang, H.; Su, Y.; Bao, H.; Chen, M.; and Xu, Q. 2024. Two-dimensional discrete bi-neuron Hopfield neural network with polyhedral hyperchaos. *IEEE Transactions on Circuits and Systems I: Regular Papers*.

Dai, D.; Sun, Y.; Dong, L.; Hao, Y.; Ma, S.; Sui, Z.; and Wei, F. 2022. Why can gpt learn in-context? language models implicitly perform gradient descent as meta-optimizers. *arXiv preprint arXiv:2212.10559*.

De Bernardi, M.; Khouzani, M.; and Malacaria, P. 2019. Pseudo-random number generation using generative adversarial networks. In *ECML PKDD 2018 Workshops: Nemesis 2018, UrbReas 2018, SoGood 2018, IWAISe 2018, and Green Data Mining 2018, Dublin, Ireland, September 10-14, 2018, Proceedings 18*, 191–200. Springer.

Desai, V.; Deshmukh, V.; and Rao, D. 2011. Pseudo random number generator using Elman neural network. In *2011 IEEE Recent Advances in Intelligent Computational Systems*, 251–254. IEEE.

Desai, V.; Patil, R.; and Rao, D. 2012. Using layer recurrent neural network to generate pseudo random number sequences. *International Journal of Computer Science Issues*, 9(2): 324–334.

Desai, V.; Patil, R. T.; Deshmukh, V.; and Rao, D. 2012. Pseudo random number generator using time delay neural network. *World*, 2(10): 165–169.

Feng, G.; Zhang, B.; Gu, Y.; Ye, H.; He, D.; and Wang, L. 2024. Towards revealing the mystery behind chain of thought: a theoretical perspective. *Advances in Neural Information Processing Systems*, 36.

Garg, S.; Tsipras, D.; Liang, P. S.; and Valiant, G. 2022. What can transformers learn in-context? a case study of simple function classes. *Advances in Neural Information Processing Systems*, 35: 30583–30598.

Gentle, J. E. 2003. *Random number generation and Monte Carlo methods*, volume 381. Springer.

Hameed, S. M.; and Ali, L. M. M. 2018. Utilizing hopfield neural network for pseudo-random number generator. In *2018 IEEE/ACS 15th International Conference on Computer Systems and Applications (AICCSA)*, 1–5. IEEE.

Jeong, Y.-S.; Oh, K.; Cho, C.-K.; and Choi, H.-J. 2018. Pseudo random number generation using LSTMs and irrational numbers. In *2018 IEEE international conference on big data and smart computing (BigComp)*, 541–544. IEEE.

Jeong, Y.-S.; Oh, K.-J.; Cho, C.-K.; and Choi, H.-J. 2020. Pseudo-random number generation using LSTMs. *The Journal of Supercomputing*, 76: 8324–8342.

Li, Z.; Liu, H.; Zhou, D.; and Ma, T. 2024. Chain of thought empowers transformers to solve inherently serial problems. *arXiv preprint arXiv:2402.12875*.

Lin, H.; Wang, C.; Yu, F.; Hong, Q.; Xu, C.; and Sun, Y. 2023. A triple-memristor Hopfield neural network with space multistructure attractors and space initial-offset behaviors. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 42(12): 4948–4958.

Liu, B.; Ash, J. T.; Goel, S.; Krishnamurthy, A.; and Zhang, C. 2022. Transformers learn shortcuts to automata. *arXiv preprint arXiv:2210.10749*.

MacLaren, M. D. 1970. The art of computer programming. Volume 2: Seminumerical algorithms (Donald E. Knuth). *SIAM Review*, 12(2): 306–308.

Matsumoto, M.; and Nishimura, T. 1998. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 8(1): 3–30.

Merrill, W.; and Sabharwal, A. 2023a. The expresssive power of transformers with chain of thought. *arXiv preprint arXiv:2310.07923*.

Merrill, W.; and Sabharwal, A. 2023b. The parallelism tradeoff: Limitations of log-precision transformers. *Transactions of the Association for Computational Linguistics*, 11: 531–545.

Oak, R.; Rahalkar, C.; and Gujar, D. 2019. Poster: Using generative adversarial networks for secure pseudorandom number generation. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2597–2599.

Okada, K.; Endo, K.; Yasuoka, K.; and Kurabayashi, S. 2023. Learned pseudo-random number generator: WGAN-GP for generating statistically robust random numbers. *PloS one*, 18(6): e0287025.

Park, S.; Kim, K.; Kim, K.; and Nam, C. 2022. Dynamical pseudo-random number generator using reinforcement learning. *Applied Sciences*, 12(7): 3377.

Pasqualini, L.; and Parton, M. 2020a. Pseudo random number generation: A reinforcement learning approach. *Procedia Computer Science*, 170: 1122–1127.

Pasqualini, L.; and Parton, M. 2020b. Pseudo random number generation through reinforcement learning and recurrent neural networks. *Algorithms*, 13(11): 307.

Press, W. H. 2007. *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press.

Radford, A.; Wu, J.; Child, R.; Luan, D.; Amodei, D.; Sutskever, I.; et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8): 9.

Smid, E. B.; Leigh, S.; Levenson, M.; Vangel, M.; David-Banks, A.; and JamesDray, S. 2010. A statistical test suite for random and pseudorandom number generators for cryptographic applications.

Tao, T.; Doshi, D.; Kalra, D. S.; He, T.; and Barkeshli, M. 2025. (How) Can Transformers Predict Pseudo-Random Numbers? *arXiv preprint arXiv:2502.10390*.

Vaswani, A. 2017. Attention is all you need. *Advances in Neural Information Processing Systems*.

Von Oswald, J.; Niklasson, E.; Randazzo, E.; Sacramento, J.; Mordvintsev, A.; Zhmoginov, A.; and Vladymyrov, M. 2023. Transformers learn in-context by gradient descent. In *International Conference on Machine Learning*, 35151–35174. PMLR.

Wei, J.; Wang, X.; Schuurmans, D.; Bosma, M.; Xia, F.; Chi, E.; Le, Q. V.; Zhou, D.; et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35: 24824–24837.

Wu, X.; Han, Y.; Zhang, M.; Li, Y.; and Cui, S. 2025a. GAN-based pseudo random number generation optimized through genetic algorithms. *Complex & Intelligent Systems*, 11(1): 31.

Wu, X.; Han, Y.; Zhang, M.; Zhu, S.; Cui, S.; Wang, Y.; and Peng, Y. 2025b. Pseudorandom number generators based on neural networks: a review. *Journal of King Saud University Computer and Information Sciences*, 37(3): 1–27.

Yang, K.; Ackermann, J.; He, Z.; Feng, G.; Zhang, B.; Feng, Y.; Ye, Q.; He, D.; and Wang, L. 2024. Do Efficient Transformers Really Save Computation? *arXiv preprint arXiv:2402.13934*.

Yun, C.; Bhojanapalli, S.; Rawat, A. S.; Reddi, S. J.; and Kumar, S. 2019. Are transformers universal approximators of sequence-to-sequence functions? *arXiv preprint arXiv:1912.10077*.

Yun, C.; Chang, Y.-W.; Bhojanapalli, S.; Rawat, A. S.; Reddi, S.; and Kumar, S. 2020. O (n) connections are expressive enough: Universal approximability of sparse transformers. *Advances in Neural Information Processing Systems*, 33: 13783–13794.