Evaluating the Quality of Randomness and Entropy in Tasks Supported by Large Language Models

Rabimba Karanjai
rkaranjai@uh.edu
Department of Computer Science
University of Houston
Houston, Texas, USA

Yang Lu
shisunny.yang@gmail.com
Department of Computer Science
University of Houston
Houston, Texas, USA

Ranjith Chodavarapu
rchodava@kent.edu
Department of Computer Science
Kent State University
Kent, Ohio, USA

Lei Xu
xuleimath@gmail.com
Department of Computer Science
Kent State University
Kent, Ohio, USA

Weidong Shi
wshi3@uh.edu
Department of Computer Science
University of Houston
Houston, Texas, USA

Abstract

The rapid advancement of large language model (LLM) technology has led to diverse applications, many of which inherently require randomness, such as stochastic decision-making, gaming, scheduling, AI agents, and cryptography-related tasks. However, the capabilities of LLMs in handling randomness, particularly in generating and utilizing random numbers effectively, remain unclear. This paper investigates the capacity of LLMs for handling tasks that involve randomness through a series of experiments. We designed a set of experiments that consider various factors that can influence an LLM's performance in tasks involving randomness, such as accessibility to external tools, types of tasks, model states (fresh vs. non-fresh), and prompting strategies. The experiments cover a range of tasks, including generating random numbers, generating random strings such as passwords, shuffling items, and evaluating the quality of randomness using entropy and the NIST randomness test-suite. Our findings reveal that while LLMs can generate outputs that exhibit some degree of randomness, their performance is inconsistent and often deviates significantly from the expected behavior. The analysis of the experimental results highlights key limitations and areas where improvement is needed for the LLMs to effectively handle tasks involving randomness.

Keywords

LLM, randomness, entropy, agents, NIST

ACM Reference Format:
Rabimba Karanjai, Yang Lu, Ranjith Chodavarapu, Lei Xu, and Weidong Shi. 2025. Evaluating the Quality of Randomness and Entropy in Tasks Supported by Large Language Models. In . ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Conference'17, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Large language models (LLMs [32]) have found numerous applications, such as natural language processing (NLP), machine translation, source code generation and translation, question-answering, chatbots [28], health-care training, customer service [20], time-series prediction, etc.

Recently, there has been increased interest in building autonomous agents using LLM, which is used as the agent's central computation engine. To support LLM based agents, there are several supporting components such as planning, memory, and action [18, 35].

LLM-based agents can be used for a wide range of applications such as co-piloting operating systems [23], playing games [17], Web agents [39], making API calls [13] and offering cyber security suggestions [38]. A recent news shows a surprising use case where LLMs are used to generate lottery tickets.

Random number generation plays a crucial role in various applications, including cryptography, Web protocols such as security tokens and session identifiers, science simulations, task scheduling, resource allocations, financial asset management, optimizations, and computer games. The quality of randomness is essential for ensuring security, fairness, and trustworthiness in these applications [10].

LLMs are not designed inherently for the generation of random numbers. Their underlying architecture is deterministic, which means that given the same input, they may produce the same output. However, recent studies have explored the potential of LLMs to exhibit randomness through techniques such as sampling from probability distributions and incorporating stochastic elements in their decision-making process [10, 34].

In this paper, we investigate the capability of LLMs to handle tasks that require randomness in the responses. We design a set of experiments that consider various factors that can influence the performance of an LLM in random tasks, such as types of tasks (directly or indirectly involving random sources), model states (fresh vs. non-fresh) and prompting strategies. Our analysis includes evaluating the quality of randomness using metrics such as entropy, and comparing LLM-generated outputs to those produced by established random number generators and algorithms.

Our research has shown essential insights into the abilities and limitations of LLMs in generating random outputs. We discovered

that while LLMs can mimic randomness to a certain extent, they still struggle to achieve high quality randomness due to its inherent limitation in the algorithms and potential biases. This finding has significant implications for the scientific community, as it highlights the need for further research and development to enhance the randomness capabilities of LLMs, especially for applications where true randomness or cryptographically strong random source is critical, such as cryptography-oriented AI agents and scientific simulations (LLMs for sciences). Moreover, our research provides valuable guidance for the developers and researchers working with LLMs, enabling them to understand better and control the quality of randomness in responses of these models for various applications.

Our main contributions include:

- A benchmarking suite with multiple tasks that involve runningdonmess to test the quality of randomness generated by LLMs[1].
- A comparative analysis of LLMs and local PRNGs, highlighting the performance differences between LLMs and local methods in creating high-quality randomness.
- An analysis of the impact of external tools and type of tasks on an LLM's ability to handle randomness reveals that LLMs can generate more random outputs when utilizing external tools.
- A discussion of potential solutions to improve the randomness of LLMs, such as incorporating entropy-based sampling and parallel chain-of-thought decoding.

2 Background

2.1 The Need for Entropy and Randomness

GenAI and Large Language Models (LLMs) have achieved extraordinary success across a wide range of application tasks such as question answering, document summarization, decision-making, code generation, reasoning, etc. Increasingly, GenAI models have been applied to tasks that demand entropy and randomness in the responses. Emerging use cases like LLM based game engines [36], simulations in various domains, GenAI for sciences, LLM based API agents [29], industrial optimizations, LLMs for making schedule decisions, all of these application tasks involve producing outputs with randomness. For example, random numbers are widely used in security protocols like challenge-response mechanisms, Web tokens, security nonce generation, session identifiers. For robustness and trustworthiness in security, it requires GenAI based Web agents to support high quality randomness within these security protocols. In the field of scientific simulations like biology and physics, random number generators are crucial for simulating various conditions or events. The quality of entropy used in GenAI process will have direct impacts on the accuracy of the scientistic modeling and simulation outcome. For the tasks that involve scheduling and statistical sampling, for instance, randomized clinical trials, randomized resource allocations, randomized task assignments, ensuring good quality randomness is essential to avoid biases and guarantee fairness. Many emerging finance and economic use case scenarios of GenAI like randomized order execution, asset management and risk management also require high quality randomness in the outputs.

[1]Code and artifacts in GitHub. The link will be updated upon acceptance.

2.2 Randomness as a Trustworthiness Problem

While LLMs have demonstrated generating human-like text, the inherent design of token generation still poses a challenge to producing high quality random outputs to satisfy the needs of many applications. LLMs are trained on massive datasets and learn to predict the most likely next word or token based on the input and their training data. Research has shown that LLMs face challenges in generating truly random distributions. Studies have suggested that LLMs often exhibit biases and struggle to produce true randomness when it is needed, especially in complex use case scenarios. The lack of supporting reliable and good quality entropy can become a major trustworthiness issue for any application aiming to leverage LLMs for tasks where randomness is crucial to ensure fairness, reduce bias, support robustness and provide security guarantee.

2.3 Evaluation of Randomness in LLM Responses

With the emergence of randomness-controlled models, the effectiveness of creating entropy in the related tasks has become a critical issue, as it may lead to biased or discriminatory outputs [19]. To address these challenges, various metrics have been proposed to evaluate the randomness of LLM. Hopkins and Renda [16] offer two sampling methods-non-autoregressive sampling (NARS) and autoregressive sampling (ARS), to evaluate the distribution sampling capabilities of LLMs in two controlled domains: uniform random number sampling and probabilistic context-free grammar (PCFG) sampling. Notably, NARS demonstrates superior performance over ARS in terms of error, variance, and containment metrics.However, Hopkins and Renda did not consider the access of external tools, like pseudo-random number generators (PRNGs), thus resulting in substantial interference to the final conclusions. Liu [21] extends this line of research by instructing GPT-4 to generate either a single number or a random sequence using varied prompts. It reveals that GPT-4 attempts to compensate for the uniformity of random numbers by sacrificing independence when functioning as a random number generator. Despite tasks directly instructing the LLMs to generate numerical values, common character-based and shuffling-related tasks should also be considered when testing the randomness of large language models. Hence, more types of tasks involving entropy should be considered, taking into account factors such as accessibility to external tools, types of tasks, model states, and prompts. Furthermore, new metrics should be proposed to accommodate the experiments of these different tasks.

3 Measuring LLMs' Capability on Handling Tasks Involving Randomness

Analyzing the way that an LLM handles tasks involving randomness directly (either static or dynamic) is challenging due to the system's complexity and limited access to its internal details when it is closed source. Therefore, we adopt an indirect approach. Specifically, we treat an LLM as a black box and design a series of experiments considering the major factors that may influence its performance in the tasks that involve randomness.

3.1 Factors Considered in the Experiments Design

We consider multiple factors in the design of our experiments to ensure a comprehensive measurement of the ability of LLMs to create randomness. These factors help to isolate the specific capabilities of LLMs and understand how different conditions might affect their performance in generating random outputs.

Accessibility to external tools. LLMs can be augmented with external tools, such as pseudo-random number generators (PRNGs), to enhance their abilities in various tasks [16]. In the context of randomness, access to a PRNG can significantly influence an LLM's performance. We evaluate LLMs both with and without access to a PRNG to investigate whether they can generate random numbers independently or benefit from a traditional PRNG's assistance. This factor is crucial for understanding the inherent capabilities of LLMs in generating random numbers and whether they can achieve reasonable randomness without relying on external tools.

Types of tasks. Tasks involving randomness vary significantly in complexity, the types of PRNG required (statistical PRNG, cryptographic PRNG, etc.), and how they utilize random numbers. It is often not obvious to the LLMs how randomness should be applied to meet the goals of specific tasks based on the prompt instructions. In this work, we consider two categories of tasks:

- *Direct tasks.* These tasks explicitly require the generation of random numbers. Examples include generating a sequence of random integers within a specified range or producing a random floating-point number between 0 and 1. This category focuses on the LLM's ability to generate random numbers directly, a fundamental aspect of randomness in LLMs.
- *Indirect tasks.* These tasks require an LLM to utilize random sources implicitly, often involving operations such as shuffling or sampling. Examples include shuffling a deck of cards, randomly selecting elements from a list, or generating random permutations of a sequence. This category assesses the LLMs' ability to apply randomness in more complex scenarios where random number generation is a means to achieve a specific outcome.

By evaluating LLMs on both the direct and indirect tasks, we aim to assess their ability to handle randomness across a spectrum of applications.

Model states. The internal state of an LLM, influenced by its previous interaction history, can affect its behavior on new tasks. To account for this, we consider both "fresh" (newly initialized) and "non-fresh" (previously used) LLM models in our experiments. This distinction allows us to investigate whether prior experiences influence an LLM's ability to generate random outputs.

Interestingly, research suggests that LLMs might not only inherit human biases in randomness generation but could potentially amplify them [34]. This highlights the challenges of achieving true randomness in LLMs and the need for careful evaluation.

Prompts. Prompts are the only source for an LLM to receive external task descriptions in the inference phase. Many works have demonstrated that carefully designed prompts can lead to better LLM responses [26]. For the same task involving randomness, it is very likely that different prompts result in different randomness handling.

3.2 Experiment Categories

Based on the factors outlined above, we systematically evaluate task types on an LLM's ability to handle randomness, while also considering the influences of model status. To ensure a comprehensive evaluation, we included a variety of LLMs in our study, namely OpenAI's GPT-4o, Google's Gemini 1.5 Pro, Mistral Large(2047), Gemma2 27b and Llama 3.1 8b. This selection allows us to evaluate randomness generation capabilities across different models and architectures.

To rigorously evaluate the capability of LLMs to handle tasks that involve randomness and generate the output based on the given task, we designed a series of experiments encompassing three distinct task categories: numerical, character-based, and shuffling-related. These categories were selected to represent different types of applications for randomness, from basic number generation to more complex tasks involving sequences and permutations. This methodology draws inspiration from the existing research on evaluating LLM sampling in controlled domains [16] and aims to provide a comprehensive assessment of LLMs' ability to exhibit randomness across various scenarios.

3.3 Numerical Value Related Tasks

Random number generation is widely recognized as the most fundamental task associated with randomness. Our evaluation considers LLM-based agents with and without access to a pseudo-random number generator (PRNG). This allows us to investigate whether LLMs can generate random numbers independently or benefit from a traditional PRNG's assistance, similar to the approach used in the previous studies [10]. Furthermore, we vary the scale of the task, including the size of the output (e.g., generating a single number versus a sequence of numbers) and the range of random numbers (e.g., integers within a specific interval, floating point values). This manipulation of scale allows us to assess the impact of these factors on the LLM's performance and identify potential limitations in its ability to generate random numbers across different magnitudes and data types.

We employ statistical tests commonly used to evaluate random number generators, such as the well-defined tests in the NIST randomness test suite [5], to assess the quality of the generated random numbers. These tests will help us determine if the distribution of the LLM-generated numbers significantly deviates from a truly uniform distribution, indicating potential biases or patterns in the LLM's outputs.

3.4 Characters Related Tasks

This category requires the LLM-based agent to manipulate characters randomly, such as generating random strings given a specified alphabet (e.g., generating random passwords, or random sequences of letters from the English alphabet). While this task can theoretically be reduced to random number generation by mapping characters to numerical values, it remains unclear whether LLMs can

effectively leverage this relationship. Our experiments aim to determine whether LLMs can exhibit randomness in character-based tasks without explicit reliance on numerical methods.

To evaluate the randomness of the generated character strings such as passwords, we analyze their statistical properties, such as the frequency distribution of individual characters and the presence of recurring patterns or sub-strings.

3.5 Shuffling Related Tasks

Shuffling is a classic application of randomness with extensive research in various domains, including card games, data analysis, and algorithm design. The problem of card shuffling, while seemingly a simple use case, has been intensively studied in the field of applied statistics, not only due to its significance to ensure fairness in card games, but also its applicability in diverse fields, ranging from game design, data analysis to scientific research.

In card shuffling, a well-shuffled deck ensures that each player has an equal chance of receiving any particular card or combination of cards. A strong uniform stopping time ensures that after the stop, the deck is in a truly random state, regardless of how it was initially arranged. Theoretical results have been obtained to understand stopping times in the context of conventional card shuffling techniques [7, 12, 33].

In this study, we evaluate LLMs on tasks analogous to shuffling cards, such as assigning patients to doctors or permuting a set of words. These tasks represent a higher complexity level than simple number or character generation, requiring the LLMs to understand and apply the concept of random permutations.

We analyze the generated permutations for uniformity and randomness to assess the LLMs' performance on shuffling tasks. We compare the LLM-generated permutations to those produced by a well-established shuffling algorithm to identify any biases or deviations from the expected randomness.

By conducting these experiments across a range of tasks and evaluating the results using appropriate statistical measures, we aim to provide a systematic assessment of LLMs' capacity for generating random output and identify potential areas for improvement in their ability to exhibit true randomness. The scoring method for the shuffle process is based on the minimum entropy of the distances between each pair of cards after the shuffle is applied. This effectively captures the idea that shuffling should maximize statistical entropy, which is a measure of randomness that can be calculated when the inner workings of the random system are unknown.

Below is how we compute statistical entropy as a percentage of the maximum possible entropy:

(1) N := number of cards in the deck
(2) K_{ijd} := after how many trials card i was distance d from card j
(3) H := percent of max entropy,
$\min\{-\sum_{d=1}^{N-1}(\frac{K_{ijd}}{\sum K_{ij}} log_{N-1}\frac{k_{ijd}}{\sum Kij})|0 < i, j < N\}$

The above method has been used to evaluate the quality of different card shuffling algorithms.

3.6 Direct Generation

We evaluate three specific domains to assess whether the LLMs can generate randomness in their responses. Our goal, however, was not only to deal with randomness generation, but also to investigate how the LLMs handle tasks that require randomness as part of their core function but were not asked to generate it directly like random number sequence creation.

We instructed the LLMs to generate random numbers and then apply the NIST randomness test-suite to evaluate whether they could pass the randomness tests. Each LLM was prompted to generate 10000 random 8-bit integers. The prompts used were as follows:

- **GPT-4o** "Can you please create 10000 random positive integers in decimal format, the highest of which is 255."
- **Gemini and Others** "Can you please create 10000 random positive integers in decimal format, the highest of which is 255." with System Prompt "You are a true random number generator. You will be asked to generate random numbers in JSON format. Do not give any code or ideas. Only the answer."

The generated sequences were then subjected to the Random Number Test Suite [5] to assess the quality of randomness [34].

3.7 Evaluation Metrics

We utilized the Random Test Tool (RTT) [37] to assess the quality of randomness in the number of sequences generated by the LLMs. This tool provides a comprehensive suite of statistical tests designed to evaluate various aspects of randomness, including uniformity, independence, and unpredictability.

The tool implements the popular NIST Randomness Test Suite [8]. Compared with similar tools, RTT is user-friendly, easy to manipulate, and capable of delivering clear results to users. The set of NIST Tests supported are.

Monobit (Chi2) This test is intended to see if the frequencies of 1 and 0 across the entire n-bit sequence are approximately equal, meaning that the proportion of 1s and 0s is close to half. If the number of 0s and 1s are not the same, it is intended to see if their difference falls within the limit of randomness.

Frequency in block This test is intended to ensure that frequencies of 1 and 0 are evenly distributed across the entire n-bit sequence.

Run Test This test is intended to see if the frequencies of runs of 1s and 0s of various lengths would be within the limits of randomness.

Longest run of Ones This test is intended to see if the frequencies of the longest run of 1s of various lengths appearing in the sequence are consistent with that expected for a random sequence.

Binary Rank This test is intended to see if the n-bit string has repetitive patterns across its entire sequence. The n-bit string is sequentially divided into N disjoint blocks, and it endeavors to see linear dependence among its fixed length substrings of each block.

Linear Complexity A long-bit string is usually obtained from an LFSR (Linear Feedback Shift Register). The bit sequence from which a longer LFSR is obtained can be termed as random, while the shorter LFSR indicates non-randomness. The linear complexity test looks for the length of the LFSR and determines if the bit sequence from which the LFSR is obtained is random or not.

Serial Test The serial test counts the frequency of all possible overlapping m-bit patterns across the entire n-bit sequence, and

Table 1: *p-value* Interpretation

Label	*p-value*	Interpretation
OK	$0.1 < p < 0.99$	Test successful
SUSPECT	$0.01 < p < 0.1$ or $0.9 < p < 0.99$	Suggest to re-test
KO	$p < 0.01$ and $p > 0.99$	Test failure

based on the deviations of each of the counts together, one intends to see if the sequence can be termed as random or not.

Spectral The focus of this test is the peak heights in the Discrete Fourier Transform of the sequence. The purpose of this test is to detect periodic features (i.e., repetitive patterns that are near each other) in the tested sequence that would indicate a deviation from the assumption of randomness.

Sign This test checks the equal repartition of the data around the median.

Given a random sequence, the tests calculate a *p-value*. The *p-value* represents the probability of obtaining a distribution at least as extreme as that observed. Then, the tool compares the *p-value* to a threshold (0.01). If the *p-value* is lower than this threshold, it implies that the probability of the observed behavior occurring by chance is 1 - 0.01 (99%). For *p-value* < 0.01, a perfectly random sample is expected to fail this test only 1% of the time. The tool follows the default classification:

Interpretation of p-values and classification of results adhere to the guidelines provided in the RTT documentation [5].

- **OK:** The test is successful if the p-value falls within the range [0.1, 0.99]. This indicates that the observed distribution is consistent with what would be expected from a truly random sequence.
- **SUSPECT:** If the p-value is in the intervals [0.01, 0.1] or [0.9, 0.99], it suggests a potential deviation from randomness. Further investigation and re-testing are necessary to confirm or reject the null hypothesis of randomness.
- **KO:** The test fails if the p-value is < 0.01 or > 0.99. This indicates a statistically significant deviation from randomness, suggesting that the sequence is likely not random.

3.8 Remark on Reproducibility

Reproducibility is a crucial property that facilitates trust in certain applications of GenAI, for instance, healthcare use cases, finance, legal applications, many scenarios in various science domains. In case that GenAI responses to application tasks depend on random sampling, and entropy, reproducibility should be interpreted within the context of the tasks. For instance, for card shuffling, it makes more sense to define reproducibility as producing similar random distribution of the shuffled cards after performing the same of number of card shuffles in each trial. For the tasks relying on random sampling, reproducibility needs to be defined as reproducible distribution of the outputs given the same experiment setting. This is the interpretation of reproducibility that we have taken in the context of this endeavor. During experiments, we conducted data collection in multiple iterations to ensure consistent entropy measurements across different trials.

4 Experimental Results and Analysis

For our experiments, we choose a combination of open-weight and closed-weight models. The open models allow us to dig more deeply into their generation strategies and allow us to play with the temperatures. Whereas for the closed models, we used both an official API and a web-based interface to collect data.

4.1 Analysis of NIST Randomness Tests

This analysis compares the randomness quality of various Large Language Models (LLMs) and local Python random number generators based on the results from the NIST randomness testsuit.

Table 2: Percentage of test outcomes for different random number generators

Generation Methods and LLM	OK	SUSPECT	KO
Local 8-bit numbers	87.78%	11.11%	1.11%
Local 1M data sample	91.11%	7.78%	1.11%
random.SystemRandom()	87.78%	11.11%	1.11%
secrets.randbelow()	88.89%	8.89%	2.22%
Gemini 15	30.56%	13.89%	55.56%
Phi-3 (Run 1)	22.22%	0%	77.78%
Phi-3 (Run 2)	25%	12.5%	62.5%
Gemma 2 27B	11.11%	0%	88.89%

4.1.1 Key Observations.

(1) **Local Python Generators:** The local Python random number generators (8-bit numbers, 1M data sample, SystemRandom, and secrets.randbelow) performed significantly better than the LLMs, with over 87% of tests passing (OK) for all four methods.
(2) **LLM Performance:** The LLMs (Gemini 15, Phi-3, and Gemma 2 27B) showed poor randomness qualities, with a high percentage of failed (KO) tests.
(3) **Gemini 15:** Performed the best among the evaluated LLMs, passing 30.56% of tests, but still significantly underperformed compared to the local Python generators.
(4) **Phi-3:** Showed inconsistent results between two runs, with Run 2 performing slightly better than Run 1.
(5) **Gemma 2 27B:** Demonstrated the poorest performance among the measurable models, failing 88.89% of the tests.
(6) **Honorable Mention:** We tried to run these tests with llama 3.1 8b model as well. That did not pass any test.

The analysis reveals a clear distinction between the randomness quality of the local Python random number generators and LLMs. Local Python methods consistently produced high-quality random numbers, passing most statistical tests. In contrast, the LLMs struggled to generate truly random sequences, with Gemma 2 27B performing particularly poorly.

These results suggest that LLMs, in their current state, are not suitable for applications requiring high-quality random number generation.

4.1.2 Qualitative Analysis. While the quantitative results provide a statistical assessment of randomness, a qualitative analysis can reveal further insights. Observing the generated sequences, we

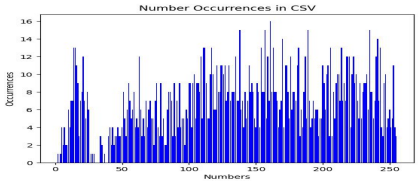

Figure 1: Gemini 1.5 Pro distribution of random numbers.

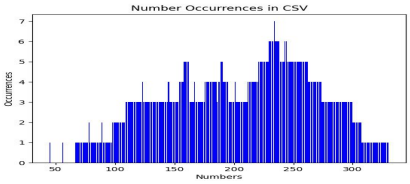

Figure 2: Mistral Large(2047) distribution of random numbers.

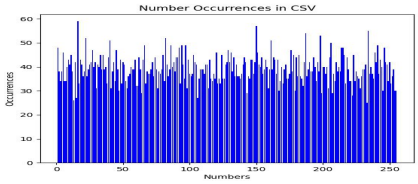

Figure 3: Gemini 1.5 with function calling (10k).

noticed certain patterns and tendencies that deviate from ideal randomness. For example, some LLMs exhibited a slight bias towards generating certain numbers or ranges of numbers more frequently than others. This observation aligns with the previous research that highlights the limitations of LLMs in accurately mimicking target distributions [16]. This is specially relevant when talking about the LLMs that try to generate random numbers on their own without relying on external tools.

Figure 1 and Figure 2 show the distribution of random numbers between 0 to 255 when generated 10000 times. These generations were done using the described method in Section 3 and do not use any function call.

Even without the randomness tests, we can see the distribution is not random, and each model favors a specific set of numbers. For Gemini 1.5 Pro Fig: 1, the top 3 Preferred Numbers are 161 (16 occurrences),138 (15 occurrences), and 235 (15 occurrences). For Mistral Fig:2, this behavior is even more clustered, with the cluster being in the middle with 234 (7 occurrences), 243 (6 occurrences), and 230 (also 6 occurrences).

However, when we have Gemini with function calling enabled, we see in Fig 3 the numbers being almost evenly distributed for 10,000 random number generation. This is where we observe that the model is defaulting to the workflow as depicted in Figure 4. Things take a bit different turn when we have 100,000 random numbers generated with Gemini using function calling, as we see in

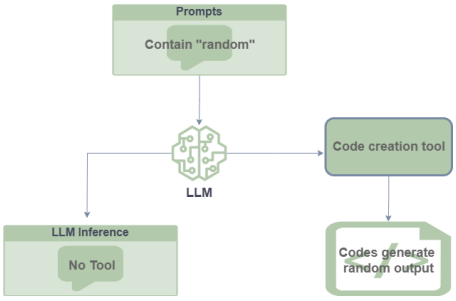

Figure 4: LLM function call workflow.

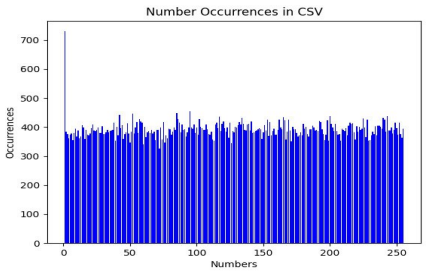

Figure 5: Gemini 1.5 with function calling (100k).

Figure 5 where almost the whole spectrum of numbers is evenly distributed apart from Number 1, which disproportionately is chosen 730 times.

4.2 Shuffling Results

This section analyzes the results of the shuffle method obtained through our experimental process.

4.2.1 Method. We focus on evaluating the effectiveness of different shuffling methods and quantifying the degree of randomness achieved using an entropy-based metric. We aim to gain insights into the factors influencing randomness generation in these models by analyzing the entropy values obtained from various LLMs and shuffling scenarios.

To assess the randomness generation capabilities of LLMs, we designed experiments centered around shuffling a deck of playing cards. This task provides a well-defined and quantifiable measure of randomness, allowing for a systematic evaluation of different LLMs and shuffling methods. We employed two primary shuffling scenarios: (1) Local shuffle, where the LLM generates Python code executed to shuffle the deck, and (2) LLM shuffle, where the LLMs directly shuffle the deck without external code.

To quantify the degree of randomness achieved in the shuffled decks, we utilized the entropy measure described earlier. Our entropy definition grades the quality of the unknown shuffling method by measuring pairwise distances between two cards. In the context of card shuffling, higher entropy values correspond to greater randomness in the sequence of cards. We varied the number of shuffle rounds (128 to 2048) and recorded the corresponding entropy values for all the tested scenarios.

Table 3: Entropy results for the card shuffling experiments.

Shuffle rounds	Local	GPT	Gemini
128	0.874451358681846	0.861846152487087	0.87337156361593
256	0.897806026358848	0.876740097767342	0.899403147801315
512	0.899562541724958	0.895595748483194	0.908432001491059
786	0.916425927773246	0.905871556497765	0.910063521198634
1024	0.915989771924721	0.909819923462218	0.912599250807944
1280	0.914072526105781	0.915665610283595	0.913379487780489
1536	0.918177419740495	0.918220137760354	0.917924657467474
1792	0.921352692988455	0.921004938696014	0.923022991807562
2048	0.923277055991329	0.924444147944848	0.92356798431639

We conducted experiments with several LLMs, including variants of GPT, Gemini, and Llama. For each LLM, we systematically varied the number of shuffle rounds (from 128 to 2048) in both Local and LLM shuffle scenarios. This allowed us to observe how the degree of randomness changed with increased rounds of shuffling, and identify potential convergence patterns.

Table 3 summarizes the entropy results for the card shuffling experiments conducted with GPT, Gemini, and the local shuffle scenarios.

As the number of shuffle rounds increases, the entropy values for the local, GPT, and Gemini shuffles exhibit a clear trend of convergence toward a high entropy value. The entropy values for all three are remarkably close across all the shuffle rounds. This suggests that the LLMs demonstrate a comparable ability to generate random shuffles given sufficient number of shuffling rounds.

To further investigate the generalizability of our findings, we conducted a card shuffling experiment using the Llama LLM. Table 4 presents the entropy results for Llama using the GPT shuffle scenario.

Table 4: Entropy results of Llama card shuffles.

Rounds	128	256	512	1024
Entropy	0.7869377	0.7885018	0.805581	0.8087705

The Llama 3.1 8b exhibits consistently lower entropy values compared to GPT and the local shuffles, suggesting potential limitations in its ability to generate random sequences. This discrepancy could be attributed to differences in model architecture, training data, or specific prompting techniques used.

Using the defined entropy measurement, we compare the entropy of GPT card shuffles with the result of the locally shuffled cards. To reduce the token count, we asked GPT to shuffle ten cards up to 2048 times (rounds). Local card shuffles were performed using the Python card shuffle code returned from GPT. The entropy results indicate that there is almost no difference between the result of GPT and the result of local shuffles.

It is often assumed that adjusting the temperature parameter in LLMs can significantly improve the randomness of their outputs. However, our experiments suggest that this might not always be the case. For specific tasks, tweaking the temperature settings may not lead to substantial changes in the underlying concepts generated by the LLM, even if the specific words used vary. This observation highlights the need for alternative approaches to enhance randomness in LLMs, potentially focusing on refining prompting techniques, incorporating stochastic elements in the model architecture, or exploring different sampling methods.

4.3 Analysis of LLM Generated Passwords

This analysis compares the randomness quality of the password sequences created by the LLMs. In these test scenarios, the LLMs were instructed to generate random passwords of a certain length with characters chosen from the English alphabet (both the lower and uppercase letters), decimal digits, and a set of special characters. Randomness of the generated passwords were tested with the same NIST randomness test tools.

Table 5: Percentage of test outcomes for LLM generated password sequences.

Generation Methods and LLM	**OK**	**SUSPECT**	**KO**
GPT	44.4%	11.1%,	44.5%
Gemini 15	33.3%	11.1%	55.6%
Phi-3	0%	11.1%	88.9%
Gemma 2 27B	0%	0%	100%

The results indicate that the password sequences generated by the LLMs exhibit poor entropy and low quality randomness. The percentage of the failed NIST randomness tests was high for all the evaluated LLMs. Examining the generated passwords by Gemma 2 and Phi-3 show that the generated password sequences contained repeated passwords, or repeated substrings.

4.4 Summary of Factors Affecting LLMs Capability on Handling of Randomness

While LLMs can exhibit variability in their outputs, achieving true randomness remains challenging due to their deterministic training process and inherent biases[1]. These models are trained on massive datasets, learning to predict the next word in a sequence based on patterns and relationships within the data[30]. While enabling impressive language generation capabilities, this process can limit their ability to produce truly random outputs [34].

Several factors contribute to LLMs limitations in handling tasks involve randomness:

- **Bias in the training data**: If the training data contains biases or predominantly reflects specific patterns, the model's output may be skewed, even when randomness is desired [31].
- **Limited knowledge update**: LLMs cannot update their knowledge base in real-time, hindering their ability to incorporate new information and adapt to changing contexts, which is crucial for generating random outputs based on the latest information [3].
- **Lack of true understanding**: LLMs do not possess genuine comprehension of the text they generate, which can lead to non-sensical or irrelevant output, especially when dealing with complex or nuanced concepts that require randomness [6].

Despite these limitations, LLMs have shown surprising capabilities in certain scenarios, such as generating numbers from specific distributions without explicit programming. This suggests potential for improvement and further research to enhance their handling of randomness.

Potential solutions include improving the diversity of training data [10], developing new algorithms to generate random sequences [2], incorporating feedback mechanisms to refine the performance of the model [4], and fine-tuning task mixtures to improve generalization. Additionally, embracing and leveraging the inherent randomness of LLMs, particularly in creative applications, can be a viable approach [9].

Furthermore, controlling randomness with parameters such as temperature and top p allows users to fine-tune the balance between predictability and variability in LLM outputs [9]. These parameters provide a degree of control over the generation process, enabling users to influence the randomness and creativity of the model's responses.

4.5 Ability to Follow Prompt Instructions

During experimentation, we made the following observations. Firstly, for some models, particularly those with small size, the ability to precisely follow prompt instructions is sometimes poor compared to larger models. For example, the output may not always be in the format requested by the prompt instructions. To fix this, in certain cases, it requires post-processing of the outputs to convert them into format that can be processed by the randomness analysis tools. Secondly, when not emphasized in the prompts, LLMs may output programming code for the requested task instead of producing the outputs based on the inputs. For example, when asked to shuffle cards, a LLM may give a card shuffle Python code as response. We mitigated this issue with more specific prompts to indicate that the output should not be code. Thirdly, some models occasionally do not perform the requested number of randomization steps. For instance, when asked to shuffle cards 1,000 times, a model may shuffle only 50 times. Since we repeated data collection with many trials and computed entropy over a large number of outputs across trials, this has been less than a problem because we can resume from where it left off.

5 Reproducibility Framework

The stochastic nature of Large Language Models (LLMs) presents significant challenges to the scientific principle of reproducibility. To address this, we established a rigorous framework to ensure the experiments presented in this paper can be reliably and accurately replicated, combining methodological constraints, task-specific interpretations of reproducibility, and a commitment to open artifact availability. A primary source of non-determinism in LLMs is the sampling strategy; therefore, to create a consistent baseline, all interactions were conducted with the **temperature parameter set to 0**. This greedy decoding approach makes the generation process deterministic for a given prompt and model state, which is essential for evaluating core capabilities. We posit that reproducibility for tasks involving randomness is achieved not through identical outputs, but through consistent statistical properties. Accordingly, for direct generation tasks like random numbers and strings, reproducibility is assessed by applying the NIST Statistical Test Suite (`sts-2.1.2`) [25] to generated sequences, where a successful replication involves passing or failing the same statistical tests. For indirect tasks like card shuffling, reproducibility is defined by the convergence of statistical entropy, using a metric that measures pairwise distances between items (detailed in Section 3.5), with successful replication indicated by convergence to values statistically indistinguishable from those in Table 3 and Table 4.

6 Limitations

Our study has several limitations that should be acknowledged. First, the evaluation is based on a specific set of LLMs and tasks, and the results may not be generalizable to other LLMs or tasks. Another limitation is the inherent nature of transformer architectures. We run all our experiments with temperature 0 to facilitate reproducibility. However, that also introduces a certain determinism when generating random numbers by prompting. That makes our results and work, dependent on prompting. Finally, since our evaluation includes closed-weight models, the lack of access to the internal workings of these models makes it difficult to fully understand and control the factors that influence their randomness generation capabilities. Future work will explore the use of open-weight models and more transparent evaluation methods and provide deeper insights into the mechanisms underlying randomness in LLMs.

7 Related Works

The evaluation of randomness generation in LLMs is an emerging field of study. Hopkins and Renda [14] provided one of the first empirical evaluations of LLMs as distribution samplers, establishing key metrics and highlighting performance differences between autoregressive and non-autoregressive sampling. Liu [19] extended this by focusing on GPT-4's ability to generate random numerical sequences, revealing that the model often compensates for uniformity by sacrificing independence. More recently, Harrison [13] compared LLM and human performance on random number generation, finding that models may still not match human-level capabilities in this specific task. Our work builds directly on these foundational studies by providing a more comprehensive evaluation framework that includes a wider variety of direct and indirect randomness tasks (numerical, character, and shuffling), applies the full NIST suite of randomness tests for a more rigorous assessment, and analyzes a broader set of contemporary LLMs.

Our focus on a rigorous reproducibility framework also situates this paper within the broader conversation on the challenges of reproducibility in machine learning research. The difficulty of replicating results in computational science, often termed the "reproducibility crisis," is particularly acute in deep learning due to numerous sources of non-determinism [15]. Even with fixed random seeds, subtle variations in software libraries (e.g., cuDNN versions), hardware (e.g., GPU architecture), and the non-deterministic nature of certain parallelized floating-point operations can lead to divergent outcomes [27].

In response, the machine learning community has proposed various best practices to mitigate these issues. Pineau et al. [24] introduced a widely recognized checklist for reproducibility, encouraging the publication of not only code but also model weights, hyperparameters, and detailed execution environments. The use of containerization technologies like Docker has also been advocated as a method to encapsulate the full software stack, ensuring that dependencies and system configurations can be perfectly replicated

[15]. Our work contributes to this effort by not only adhering to these principles through the public release of our code and experiment artifacts but also by proposing a task-specific definition of reproducibility for stochastic LLM evaluations, where statistical consistency, rather than identical output, serves as the benchmark for a successful replication.

8 Ethical Implications and Responsible Disclosure

The findings of this study, which highlight significant deficiencies in the ability of Large Language Models (LLMs) to generate high-quality randomness, carry substantial ethical implications. As LLMs are increasingly integrated into a wide array of applications, from consumer-facing tools to enterprise-level systems, a misunderstanding of their limitations in stochastic processes could lead to predictable, insecure, and biased outcomes. The demonstrated weakness in generating random passwords, for instance, poses a direct security risk. If developers or end-users mistakenly trust an LLM to generate cryptographic material or unique identifiers, the resulting outputs could be vulnerable to adversarial prediction and compromise system security. This aligns with concerns raised by Bourtoule et al. [11] regarding the unforeseen failure modes of machine learning systems when deployed in critical environments.

Furthermore, the tendency of LLMs to exhibit biases, as noted in Section 4.4, can be amplified when randomness is expected but not properly delivered. For example, in a system designed to randomly assign resources or opportunities (e.g., in randomized clinical trials or automated scheduling), a biased "random" process could lead to systematically unfair or inequitable outcomes, perpetuating societal biases present in the training data [22]. This underscores the ethical imperative for developers and researchers to be transparent about the capabilities and limitations of their models.

Consequently, we have a responsibility to disclose these findings in a manner that informs without causing undue alarm or enabling malicious actors. Our approach to responsible disclosure involves two key actions. First, by publishing this research in a peer-reviewed venue, we aim to alert the academic and industrial communities to these potential vulnerabilities, encouraging the development of more robust systems and best practices. Second, we advocate for clear guidelines and warnings within developer documentation for LLMs, explicitly cautioning against their use as a primary source of entropy for security-sensitive or fairness-critical applications. This aligns with the principle of "transparency" in AI ethics, which calls for clear communication about how AI systems operate and where they might fail [14]. We believe that a proactive and transparent approach is the most effective way to mitigate the risks associated with the misuse of LLMs in contexts requiring true or high-quality randomness.

9 Conclusions

To gain a better understanding of LLM-based agents' capabilities in handling tasks that involve randomness, we developed a set of experiments and tested several LLMs. Our analysis included evaluating the quality of randomness using metrics like entropy and well established NIST randomness test-suite. The results show that while LLMs can mimic randomness to a certain extent, they still struggle to achieve high quality randomness. This study contributes valuable insights into the capabilities and limitations of LLMs in generating random outputs.

Acknowledgments

This research was supported in part by the Google Developer Experts program, which provided Google Cloud research credits to enable our experiments.[2] Additionally, we gratefully acknowledge the financial and collaborative support of the NATO Science for Peace and Security (SPS) Programme[3], which fosters international cooperation in scientific research.

References

[1] 2023. *AI's Dicey Reputation: Are LLMs Really Just Random Stochastic Machines?* https://promptengineering.org/ais-dicey-reputation-are-llms-really-just-random-stochastic-machines/

[2] 2023. *How to Get Better Outputs from Your Large Language Model | NVIDIA Technical Blog.* https://developer.nvidia.com/blog/how-to-get-better-outputs-from-your-large-language-model/

[3] 2024. *10 Biggest Limitations of Large Language Models.* https://www.projectpro.io/article/llm-limitations/1045

[4] 2024. *LLM Challenges in Development: Key Insights.* https://www.labellerr.com/blog/challenges-in-development-of-llms/

[5] 2024. *xmco: A python tool used to run statistical tests on random data.* https://github.com/xmco/

[6] Novita AI. 2024. *All You Need to Know about the Limitations of Large Language Models.* https://medium.com/@marketing

[7] David Aldous and Persi Diaconis. 1986. Shuffling Cards and Stopping Times. *The American Mathematical Monthly* 93, 5 (May 1986), 333–348. doi:10.1080/00029890.1986.11971821

[8] Lawrence E. Bassham, Andrew L. Rukhin, Juan Soto, James R. Nechvatal, Miles E. Smid, Elaine B. Barker, Stefan D. Leigh, Mark Levenson, Mark Vangel, David L. Banks, Nathanael Alan Heckert, James F. Dray, and San Vo. 2010. *SP 800-22 Rev. 1a. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications.* Technical Report. Gaithersburg, MD, USA.

[9] Marie-Alice Blete. 2023. *LLMs: Determinism & Randomness. TL;DR: While working with GPT-3.5 Turbo.* https://medium.com/@mariealice.blete/llms-determinism-randomness-36d3f3f1f793

[10] Andrew Bouras. 2024. Integrating Randomness in Large Language Models: A Linear Congruential Generator Approach for Generating Clinically Relevant Content. *arXiv preprint arXiv:2407.03582* (2024).

[11] Lucas Bourtoule, Varun Chandrasekaran, Christopher A Choquette-Choo, Hengrui Jia, Adelin Travers, Baiwu Zhang, David Lie, and Nicolas Papernot. 2021. Machine unlearning. In *2021 IEEE symposium on security and privacy (SP)*. IEEE, 141–159.

[12] Persi Diaconis, R.L Graham, and William M Kantor. 1983. The mathematics of perfect shuffles. *Advances in Applied Mathematics* 4, 2 (1983), 175–196. doi:10.1016/0196-8858(83)90009-X

[13] Yu Du, Fangyun Wei, and Hongyang Zhang. 2024. AnyTool: Self-Reflective, Hierarchical Agents for Large-Scale API Calls. arXiv:2402.04253 [cs.CL] https://arxiv.org/abs/2402.04253

[14] Luciano Floridi and Josh Cowls. 2019. A unified framework of five principles for AI in society. Harvard Data Science Review 1 (1)(2019).

[15] Odd Erik Gundersen and Sigbjørn Kjensmo. 2018. State of the art: Reproducibility in artificial intelligence. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 32.

[16] Aspen K Hopkins and Alex Renda. 2023. Can LLM generate random numbers? evaluating llm sampling in controlled domains. Sampling and Optimization in Discrete Space (SODS) ICML 2023 Workshop.

[17] Sihao Hu, Tiansheng Huang, Fatih Ilhan, Selim Tekin, Gaowen Liu, Ramana Kompella, and Ling Liu. 2024. A survey on large language model-based game agents. *arXiv preprint arXiv:2404.02039* (2024).

[18] Wenyue Hua, Xianjun Yang, Mingyu Jin, Zelong Li, Wei Cheng, Ruixiang Tang, and Yongfeng Zhang. 2024. Trustagent: Towards safe and trustworthy llm-based agents through agent constitution. In *Trustworthy Multi-modal Foundation Models and AI Agents (TiFA)*.

[19] Dong Huang, Qingwen Bu, Jie Zhang, Xiaofei Xie, Junjie Chen, and Heming Cui. 2023. Bias assessment and mitigation in llm-based code generation. *arXiv preprint arXiv:2309.14345* (2023).

[2] This material is based upon work supported by the Google Cloud Research Credits program.

[3] See https://www.nato.int/cps/en/natohq/78209.htm

[20] Alekya Jonnala. 2024. How Large Language models (LLM) help enterprises enhance customer experiences. *Journal Homepage: http://www. ijmra. us* 13, 11 (2024).

[21] Qiang Liu. 2023. Does gpt-4 play dice? *Chinaxiv* (2023).

[22] Ninareh Mehrabi, Fred Morstatter, Nripsuta Saxena, Kristina Lerman, and Aram Galstyan. 2021. A survey on bias and fairness in machine learning. *ACM computing surveys (CSUR)* 54, 6 (2021), 1–35.

[23] Kai Mei, Xi Zhu, Wujiang Xu, Wenyue Hua, Mingyu Jin, Zelong Li, Shuyuan Xu, Ruosong Ye, Yingqiang Ge, and Yongfeng Zhang. 2025. AIOS: LLM Agent Operating System. arXiv:2403.16971 [cs.OS] https://arxiv.org/abs/2403.16971

[24] Joelle Pineau, Philippe Vincent-Lamarre, Koustuv Sinha, Vincent Larivière, Alina Beygelzimer, Florence d'Alché Buc, Emily Fox, and Hugo Larochelle. 2021. Improving reproducibility in machine learning research (a report from the neurips 2019 reproducibility program). *Journal of machine learning research* 22, 164 (2021), 1–20.

[25] Andrew Rukhin, Juan Soto, James Nechvatal, Miles Smid, Elaine Barker, Stefan Leigh, Mark Levenson, Mark Vangel, David Banks, Alan Heckert, et al. 2001. *A statistical test suite for random and pseudorandom number generators for cryptographic applications*. Vol. 22. US Department of Commerce, Technology Administration, National Institute of ....

[26] Pranab Sahoo, Ayush Kumar Singh, Sriparna Saha, Vinija Jain, Samrat Mondal, and Aman Chadha. 2024. A systematic survey of prompt engineering in large language models: Techniques and applications. *arXiv preprint arXiv:2402.07927* (2024).

[27] David Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-Francois Crespo, and Dan Dennison. 2015. Hidden technical debt in machine learning systems. *Advances in neural information processing systems* 28.

[28] Samaneh Shafee, Alysson Bessani, and Pedro M Ferreira. 2024. Evaluation of llm chatbots for osint-based cyber threat awareness. *arXiv preprint arXiv:2401.15127* (2024).

[29] Yifan Song, Weimin Xiong, Dawei Zhu, Wenhao Wu, Han Qian, Mingbo Song, Hailiang Huang, Cheng Li, Ke Wang, Rong Yao, Ye Tian, and Sujian Li. 2023. RestGPT: Connecting Large Language Models with Real-World RESTful APIs. arXiv:2306.06624 [cs.CL] https://arxiv.org/abs/2306.06624

[30] Andreas Stöffelbauer. 2023. *How Large Language Models Work. From zero to ChatGPT*. https://medium.com/data-science-at-microsoft/how-large-language-models-work-91c362f5b78f

[31] Nguyen Ha Thanh. 2023. *Bias, Randomness, and Risks of Large Language Models in High-stakes Domains | by Nguyen Ha Thanh | Medium*. https://medium.com/@nguyenthanh.asia/bias-randomness-and-risks-of-large-language-models-in-high-stakes-domains-987bc2c1517c

[32] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971* (2023).

[33] L. N. Trefethen and L. M. Trefethen. 2000. How Many Shuffles to Randomize a Deck of Cards? *Proceedings: Mathematical, Physical and Engineering Sciences* 456, 2002 (2000), 2561–2568. http://www.jstor.org/stable/2665604

[34] Katherine Van Koevering and Jon Kleinberg. 2024. How Random is Random? Evaluating the Randomness and Humaness of LLMs' Coin Flips. *arXiv preprint arXiv:2406.00092* (2024).

[35] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. 2024. A survey on large language model based autonomous agents. *Frontiers of Computer Science* 18, 6 (2024), 186345.

[36] Hongqiu Wu, Yan Wang, Xingyuan Liu, Hai Zhao, and Min Zhang. 2024. Instruction-Driven Game Engines on Large Language Models. arXiv:2404.00276 [cs.AI] https://arxiv.org/abs/2404.00276

[37] XMCO Team. 2024. Random Test Tool. GitHub repository. Available at: https://github.com/xmco/random [Accessed 12 May 2024].

[38] Yikuan Yan, Yaolun Zhang, and Keman Huang. 2024. Depending on yourself when you should: Mentoring LLM with RL agents to become the master in cybersecurity games. *arXiv preprint arXiv:2403.17674* (2024).

[39] Ke Yang, Yao Liu, Sapana Chaudhary, Rasool Fakoor, Pratik Chaudhari, George Karypis, and Huzefa Rangwala. 2024. AgentOccam: A Simple Yet Strong Baseline for LLM-Based Web Agents. arXiv:2410.13825 [cs.AI] https://arxiv.org/abs/2410.13825