Exploring Random Data Generation Capabilities of LLMs

Zhen Tang, Kevin Zhang, Qilong Wu, Xiao Zhuang
University of Illinois, Urbana-Champaign
Champaign, Illinois
{zhent6,kevinez2,qilong3,xhuang8}@illinois.edu

ABSTRACT

In the ever-evolving domain of artificial intelligence, this study explores the capabilities and limitations of large language models (LLMs) in generating random sequences. Our research focuses on the intricate behaviors of LLMs when prompted to generate random data, revealing significant insights into the effects of the "temperature" hyperparameter on data distribution. Traditionally, increasing the temperature is expected to enhance randomness and diversity in the outputs. However, some of our experiments show a counter-intuitive introduction of bias and a sharper probability distribution at higher temperatures, contradicting typical expectations.

This study employs a series of experiments, including the generation of integers, floating-point numbers, phrases, and names, to investigate the stochastic decoding dynamics of LLMs. Our findings indicate that while lower temperatures lead to a predictable and biased output, an increase in temperature does not consistently increase randomness as previously assumed. In the case of name generation, the model exhibited unexpected biases, favoring less popular names despite higher temperatures.

These results underscore the complex nature of LLMs' random generation processes and suggest that inherent biases in training data or model architecture may influence outcomes. The study highlights the need for careful consideration of these models in applications requiring high levels of randomness, such as watermarking and cryptography systems. Further research is necessary to better understand the underlying mechanisms and to optimize LLMs for unbiased and uniform random generation.

KEYWORDS

LLMs, Random Generation

1 INTRODUCTION

In the rapidly evolving field of artificial intelligence, the exploration of large language models' (LLMs) capabilities and limitations in generating random sequences has carved out a new niche at the intersection of computational linguistics and statistical analysis [1, 4]. Although LLMs have shown exceptional aptitude in numerous tasks requiring human-like reasoning, their ability to generate random content is comparatively less known. This capability plays a crucial role in various domains, such as LLM watermark algorithms used to identify AI-generated content, which are fundamentally based on this ability [5]. This study is motivated by recent findings that highlight deviations from uniformity when LLMs are prompted to generate random outputs, raising profound questions about inherent model biases and the impact of prompt engineering on the randomness of generated sequences [6].

Moreover, this research is driven by the desire to explore the underlying principles behind such generation behavior—whether there are rules similar to those governing human randomness [2]. This not only poses a challenge to our understanding of LLM mechanisms but may also provide a novel metric for evaluating their performance in applications that rely on random outputs. By extensively analyzing the discrepancies in the random numbers generated by LLMs, the distribution characteristics of the data produced, and how prompting affects the outcomes, this report aims to add new insights to the field of artificial intelligence research and address the challenges of using LLMs in scenarios that require high levels of randomness [3].

2 BACKGROUND

As artificial intelligence rapidly evolves, the capabilities and limitations of large language models (LLMs) in generating random sequences have gained increasing attention. Traditionally, random number generation is a foundational function in computer science, extensively utilized in cryptography, data analysis, and algorithm design. Compared to traditional random number generators (such as linear congruential generators), LLMs offer unique approaches and challenges in handling randomness.

Within this context, an important application of AI detection involves LLM watermarking, a technique that embeds invisible marks in content generated by LLMs to verify its authenticity. By modifying the model's output strategy, such as adjusting the logits during generation, specific biases can be introduced so the model generates content containing certain markers. For instance, a state-of-the-art method developed by Kirchenbauer et al. manipulates logits during the decoding process to introduce model biases, enhancing the generation of specific tokens, which is crucial for identifying LLM-generated content [1].

Past research has indicated that while LLMs excel in handling complex language tasks, they may not perform as well as specially designed random number generators in producing random numbers. For example, researchers Nisan and Zeldovich explored the potential problems and limitations of using LLMs to generate cryptographically secure random numbers. Additionally, studies by Tang et al. (2021) investigated the consistency and reliability of different LLMs in generating random sequences, finding significant variations in performance among models, even under the same training conditions [4].

This historical information and related research mentioned before provide a solid theoretical foundation for this report, illustrating why it is necessary to explore the capabilities and limitations of LLMs in generating random sequences.

3 MOTIVATION

AI-generated content is becoming more sophisticated and human-like, so misuse of LLM-generated content is an emerging problem. Traditional means of training models to detect AI-generated content are becoming less effective for false positives. Consequently,

LLM watermarks have been introduced to identify AI-generated content, in which one State-Of-The-Art (SOTA) method Kirchenbauer et al. [1] manipulates logits in decoding to bias the model to generate more green tokens than red tokens. However, one recent work proposed SCTS to evade that watermark without external unwatermarked LLM queries Wu and Chandrasekaran [5].

Nevertheless, one key assumption made in Wu and Chandrasekaran [5] is that "The watermarked model is aligned, and capable of following instructions provided." Yet, it is possible to be an oversimplification, especially when prompts are rather complicated, even if we replace "watermarked model" by "model". A typical prompt is as follows: `Choose two phrases (kernel includes, the kernel contains), and generate a long uniformly random string of those phrases separated by ";". Previous phrases should have no influence on future phrases: kernel includes; kernel contains; kernel includes; kernel contains; kernel contains; kernel includes; kernel contains; kernel contains; kernel contains; kernel contains; kernel contains; kernel contains; kernel contains; kernel includes; kernel contains; kernel contains; kernel includes; kernel contains; kernel contains;` Where after ":" are examples uniformly randomly generated to facilitate LLM to follow the prompt. Not strictly syntax-compliant spaces like the one before the first `kernel` are to ensure consistent tokenization. `includes` and `contains` are synonyms for a substitution attempt. More details are available in Wu and Chandrasekaran [5]. It is a non-trivial question if SOTA LLMs are capable of following such not-so-simple prompts, as Wu and Chandrasekaran [5] found their method SCTS did not do well in an unaligned model `lmsys/vicuna-7b-v1.5` Zheng et al. [6] compared with the aligned `lmsys/vicuna-7b-v1.5-16k` and observed it is likely due to the poor prompt following of the unaligned version.

Also, the possibility that the context will also make a difference over the behavior when LLM is prompted in this way cannot also be excluded. Context with several words before the random word (in the above prompt example, the context is `kernel` and the random word is `includes` and `contains`). Also, recent work shows that the random generation behavior is model-specific and the quality of generated random content is far from uniform in general Tang et al. [4]. However, their random generation experiments are limited to limited random number generation and does not take context into consideration.

We have four different experiments exploring different aspects of random generation for LLMs. First, we have integer generation. It is believed that when humans are asked to pick a random integer in a range then the generated distribution among different people is far from uniform and some numbers are significantly more likely to be picked Schulz et al. [2]. Consequently, besides the importance raised from the security of LLM watermarking's perspective, it is also very interesting if human bias or preference can be observed from LLM's generation. Second, we have float number generation, which is the only continuous random generation task. Thirdly, we have phrase generation, which studies if the frequency of the corresponding commands affects the random generation behavior. Lastly, we have human name generation experiment to see if LLM is biased towards certain human names for such a prompt. These four experiments explore the random generation behavior for LLM from different perspectives, from numerical to textual, discrete to continuous.

4 EXPERIMENTAL SETUP

For each study, we utilized Google's free PaLM API to generate and collect data samples. However, due to output token constraints, we had to make multiple successive API calls to accumulate a sufficient dataset size. Consequently, the final dataset does not represent a single, contiguous generation instance. Instead, it comprises an aggregation of multiple independent generation sequences, potentially introducing statistical inconsistencies or artifacts that could impact the validity and interpretability of the data. This piecemeal generation approach, while necessary to meet our data requirements, may have inadvertently introduced unintended biases or discontinuities within the dataset that should be considered when analyzing the results.

To investigate the influence of temperature, a crucial hyperparameter that modulates the stochasticity of the decoding process, on the characteristics of the generated data distributions, we conducted a systematic exploration across a range of temperature values. Specifically, for each experimental condition, we generated six distinct datasets, each corresponding to a temperature value of 0.0, 0.2, 0.4, 0.6, 0.8, and 1.0, respectively. The temperature parameter governs the entropy regularization during the neural network's output sampling, with lower values promoting deterministic, greedy predictions and higher values enabling greater randomness and diversity in the generated samples. By encompassing this spectrum of temperature settings, our study aimed to unveil potential regularities or disparities in the emergent data patterns, thereby elucidating the intricate interplay between the model's stochastic decoding dynamics and the resultant data manifolds.

4.1 Study 1: Integer Number Generation

This study investigates the proficiency of large language models (LLMs) in generating uniformly distributed random whole numbers within a specified range. By prompting the LLM to generate a sequence of random decimal whole numbers adhering to a uniform distribution, we aim to assess the model's ability to follow complex instructions and produce statistically sound random data.

The experimental design involves issuing a carefully constructed prompt to the LLM which is presented as follows: `Generate 10000 uniformly distributed random whole numbers between 0 and 10000. Make them decimal whole numbers by adding ".0" at the end of every number generated. Previous numbers should have no influence on future numbers: . . . (20 random numbers in [0,10000) generated by NumPy as an example).`

The prompt essentially requests the LLM to generate 10,000 uniformly distributed random whole numbers between 0 and 10,000, with each number represented as a decimal value. The prompt includes a specific formatting instruction to append ".0" to each generated number to circumvent content filtering policies. To ensure continuity and avoid potential biases, the prompt explicitly specifies that previous numbers should have no influence on future numbers. Additionally, the prompt includes a set of 20 random

numbers within the specified range, generated using NumPy, as an illustrative example.

The experiment is conducted across a range of temperature values (0.0, 0.2, 0.4, 0.6, 0.8, and 1.0) to investigate the impact of this hyperparameter on the randomness and distribution of the generated data. Multiple API calls are made at each temperature value, and the resulting data is aggregated to form the final dataset for analysis.

After collecting the data, a chi-square test is performed to determine whether the generated data follows a uniform distribution or deviates from the expected pattern. The observed counts correspond to the frequencies of the generated numbers, while the expected counts are derived from a theoretical uniform distribution with a count of 1 for each value in the range from 0 to 10,000. The null hypothesis for this study, denoted by H_0 is that the generated data follows a uniform distribution, indicating that the LLM can produce random numbers without significant deviations from the expected uniform pattern. Conversely, the alternative hypothesis, denoted by H_a, states that the generated data does not follow a uniform distribution, suggesting potential biases or limitations in the LLM's random data generation capabilities. By comparing the observed and expected counts, the chi-square test provides a statistical measure of the goodness-of-fit, enabling the evaluation of the LLM's ability to generate truly random and uniformly distributed data.

4.2 Study 2: Float Number Generation

This study investigates the proficiency of large language models (LLMs) in generating uniformly distributed random float numbers within a specified range. By prompting the LLM to generate a sequence of random float numbers adhering to a uniform distribution, we aim to assess the model's ability to follow complex instructions and produce statistically sound random float numbers, and observe its specific behaviors.

The prompt is as follows: Generate 10,000 uniformly distributed random numbers between 0 and 1. Previous numbers should have no influence on future numbers: ...(20 random numbers in [0,1) generated by Numpy as an example)

The prompt essentially requests the LLM to generate 10,000 uniformly distributed random float numbers between 0 and 1. Additionally, the prompt includes a set of 20 random numbers within the specified range, generated using NumPy, as an illustrative example.

After collecting the data, a Kolmogorov–Smirnov (K-S) test [3] is performed to determine whether the generated data follows a uniform distribution or deviates from the expected pattern. Null hypothesis H_0 and alternative analysis H_a are similarly defined as in the previous study. K-S test provides a statistical measure of the goodness-of-fit, enabling the evaluation of the LLM's ability to generate truly random and uniformly distributed data in the continuous case.

4.3 Study 3: Phrase Generation

To see how popular phrase affect the process of random generation of LLMs, we design this phrase generation experiment. We provide a word and ask LLM to randomly pick another word in a provided list to form a meaningful phrase. All combinations should be meaningful otherwise it will not provide any interpolation for watermarks. In particular, we ask LLM to generate import statements using the following prompt:
Randomly pick 10000 words, w, from the list of words: ["os", "torch", "numpy", "Kubernetes"] to form a phrase "import w". Previous choices should have no influence on future choices. Here are some examples: ... (20 random examples)
At the end of the prompt, we provided 20 examples of import statements that are randomly chosen using Numpy to give LLM some guidance to generate proper phrases.

We carefully picked the packages in order to show the potential biases in the LLMs. The packages os, torch, and NumPy are the most popular packages used in almost all large projects written in Python. Although the Kubernetes package is used in every Kubernetes operator, there is a huge quantitative gap between it and the other three hottest packages.

After the generation, we perform a chi-square goodness-of-fit test to evaluate whether generations are uniformly distributed with the null hypothesis H_0 to be "The random generation of phrases is under uniform distribution".

4.4 Study 4: Name Generation

To explore whether bias exists in the process of random generation by LLMs towards certain words, we designed a human name generation experiment. We crafted two sets of name pairs, each containing names with four letters. The first set comprised two popular names: Emma (the top popular baby name for girls in 2023, US) and Noah (the top popular baby name for boys in 2023, US). The second set included one popular name and one less popular name: Emma and Luna (the 96th most popular baby name for girls in 2023, US). For each pair, the LLM was tasked with randomly selecting one name. This setup allowed us to investigate if the frequency of a word's daily usage, reflected by the popularity of names, influences any potential bias in the LLM. The focus solely on names helped eliminate the impact of word semantics and context on the experiment.

Additionally, we employed Numpy to generate 20 random name choices as examples, aiding the LLM in proper name generation.

The final experimental prompt was as follows:
"Randomly choose between Emma and Noah for 10,000 iterations. Previous choices should not influence future selections. For example: . . . (20 random name choices are given as examples)."

We utilized a chi-square goodness-of-fit test to evaluate whether the results were uniformly distributed. The null hypothesis H_0 posited that the randomly chosen names would follow a uniform distribution.

5 RESULTS

5.1 Integer Number Generation

The chi-squared test results presented in the table provide valuable insights into the large language model's ability to generate uniformly distributed random whole numbers across different temperature settings. By comparing the generated data to a baseline dataset of truly uniform random numbers produced using NumPy, we can evaluate the deviations from the expected uniform distribution.

Table 1: The table presents the chi-squared test results evaluating whether the data generated in Study 1 follows a uniform distribution across different temperature values. The third column shows the p-value difference compared to a uniformly distributed random whole number dataset generated using NumPy which achieved a p-value of 0.2282.

Temp	P>|0.05|	% Diff from baseline
0.0	0.0000	-0.2282
0.2	0.0107	-0.2175
0.4	0.0133	-0.2149
0.6	0.0338	-0.1944
0.8	0.4197	+0.1915
1.0	0.4364	+0.2082

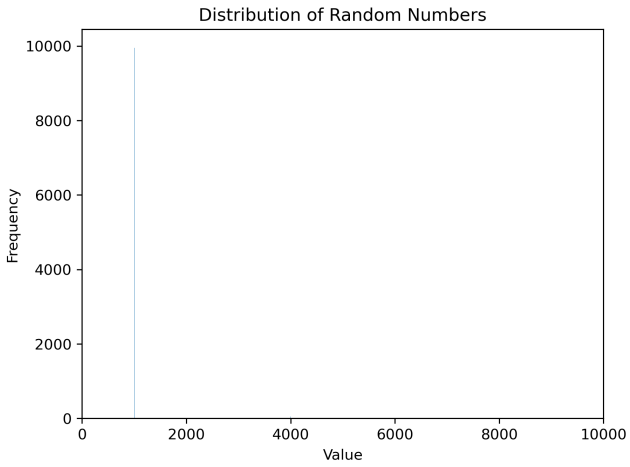

Figure 1: Distribution of Random Whole Numbers at 0 Temperature.

At lower temperature values (0.0, 0.2, and 0.4) as shown in table 1, the data generated by the LLM exhibits significant deviations from the uniform distribution, as indicated by the extremely low p-values (< 0.05). This suggests that the LLM struggles to produce truly random and uniformly distributed data when operating in a more deterministic, greedy decoding mode with limited stochasticity.

Interestingly, as the temperature increases to 0.6, the p-value improves to 0.0338, indicating a reduced but still statistically significant deviation from the uniform distribution. This implies that introducing moderate levels of randomness during the decoding process may enhance the LLM's ability to generate more uniformly distributed data, albeit with some residual biases.

A notable finding emerges at higher temperature values of 0.8 and 1.0, where the p-values (0.4197 and 0.4364, respectively) exceed the 0.05 significance threshold. Remarkably, these p-values are higher than the baseline NumPy dataset, suggesting that the LLM's random data generation capabilities may even surpass those of traditional random number generators when operating in a highly stochastic decoding regime.

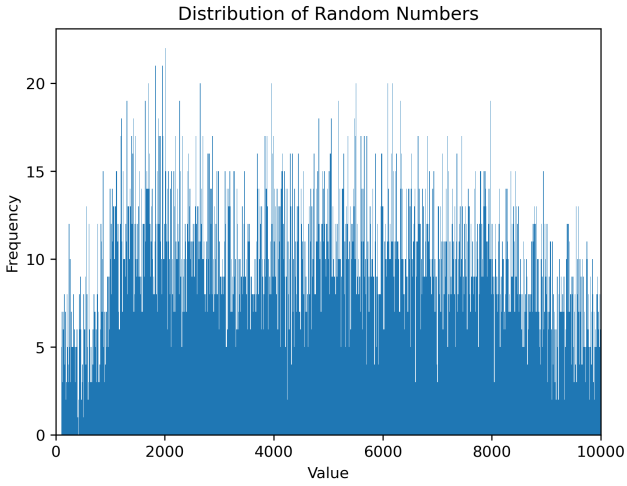

Figure 2: Distribution of Random Whole Numbers at 0.2 Temperature.

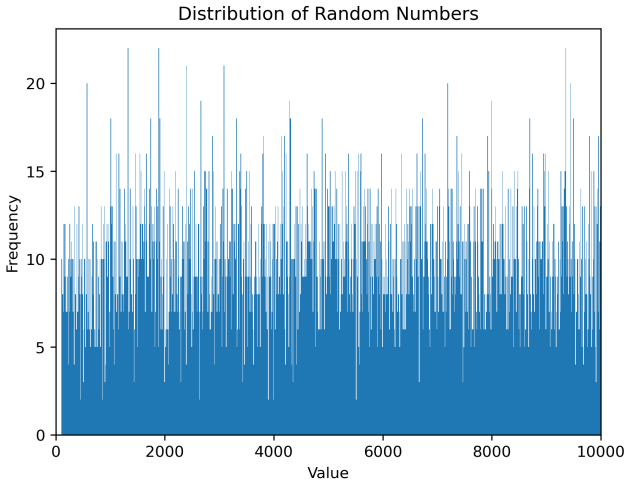

Figure 3: Distribution of Random Whole Numbers at 1.0

Table 2: K-S Test Result

Temperature	K-S statistics	p-value
0	0.0585	2.696×10^{-15}
0.2	0.0112	0.557
0.4	0.0095	0.758
0.6	0.0106	0.628
0.8	0.0129	0.376
1	0.0133	0.339
(NumPy)	0.0000	1.000

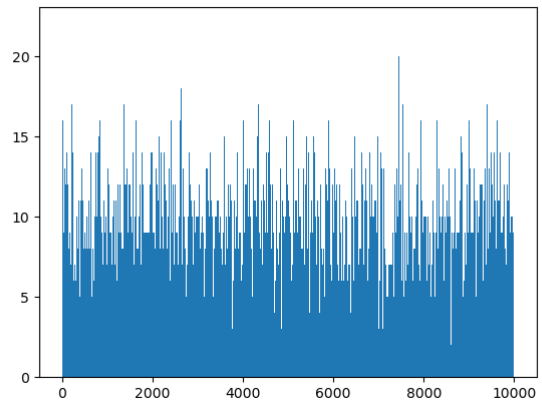

Figure 4: Distribution of Random Whole Numbers Generated by NumPy.

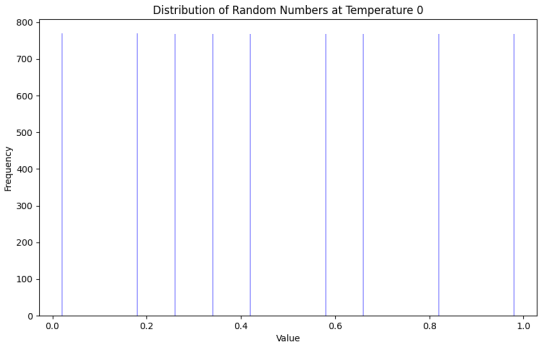

Figure 5: Distribution of Random Float Numbers at 0 Temperature.

5.2 Float Number Generation

The K-S test results presented in the table provide valuable insights into PALM's ability to generate uniformly distributed random float numbers across different temperatures. Random

At 0 temperature, the data generated by the LLM exhibits significant deviations from the uniform distribution, as indicated by the extremely low p-values (2.696×10^{-15}). This suggests that PALM fails to produce truly random uniformly distributed data when operating in a more deterministic, greedy decoding mode.

Interestingly, as the temperature becomes positive, the p-value improves to 0.3 or higher, indicating a reduced deviation from the uniform distribution. Consequently, the uniformity cannot be disproved. This implies that introducing moderate levels of randomness during the decoding process may enhance the LLM's ability to generate more uniformly distributed data, albeit with some residual biases. Also, higher temperatures do not further increase the uniformity from 0.4. Visualization results are consistent with $K - S$ tests, with 0 temperature only having several discrete bins while

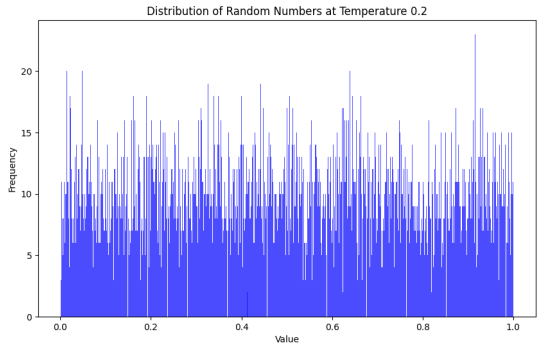

Figure 6: Distribution of Random Float Numbers at 0.2 Temperature.

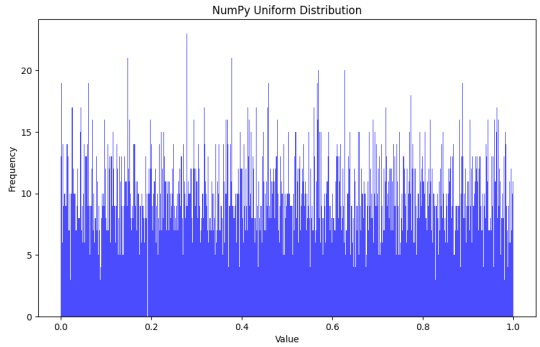

Figure 7: Distribution of Random Float Numbers Generated by NumPy.

positive temperature ones are not visually distinguishable from the NumPy baseline.

In general, PALM is able to generate reasonable random float numbers when the temperature is 0.2 or higher, and may only generate from several fixed numbers when the temperature is 0.

5.3 Phrase Generation

Table 3: Chisquare Test Result for Random Phrases Generations

Temperature	Chi-Squre Statistics	p-value
0	0.40764	0.9386
0.2	1055.27	1.8407×10^{-228}
0.4	1724.818	0.0
0.6	7204.2538	0.0
0.8	5801.506	0.0
1	6427.176	0.0
(NumPy)	3.0872	0.3783

Table 3 shows the result of chi-square tests of Random Phrases Generation. The LLM generated comparable results only at temperatures equal to 0. Figure 8 and Figure 9 show similar shapes which are nearly under uniform distribution. For other temperature settings, the Chi-Square Statistic is very high and the p-value is too small and under the significant level. Figure 10 shows the distribution at a temperature equals to 0.8 which is unlike uniformly distributed.

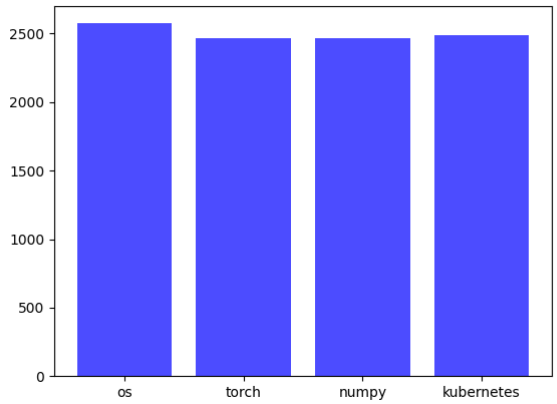

Figure 8: Distribution of Phrases Generated by NumPy.

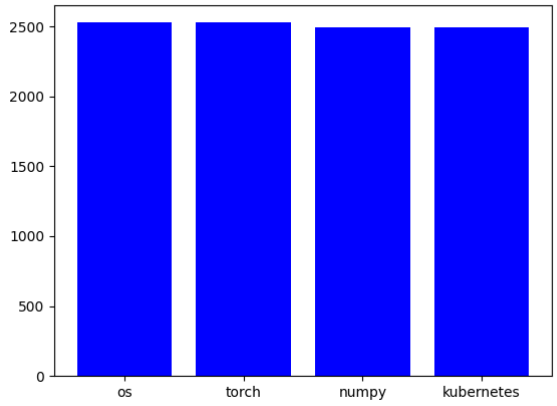

Figure 9: Distribution of Phrases Generated at 0 Temperature.

This result is quite interesting. We expected more randomness as the temperature increases, but the model gives the most likely uniform distribution only at temperature 0. We carefully inspected the raw data and found that the model generated all 4 words given in the list in order and repeats which gives a uniform distribution. In other words, there is no randomness involved in the generation process at temperature 0.

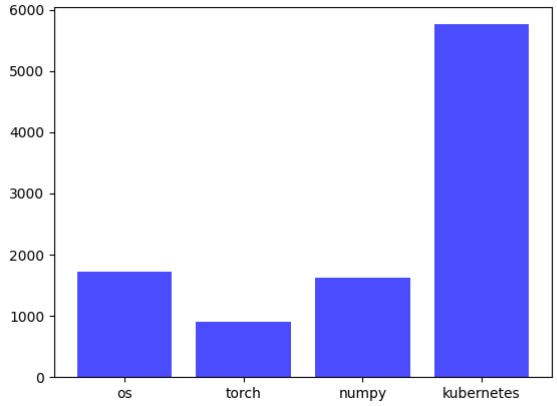

Figure 10: Distribution of Names Generated at 0.8 Temperature.

5.4 Name Generation

Table 4: Chisquare Test Result for Two Popular Names

Temperature	Statistics	p-value
0	0.0000	1.0000
0.2	0.0162	0.8985
0.4	9.9314×10^{-5}	0.9920
0.6	19.7512	8.8204×10^{-6}
0.8	47.2324	6.3047×10^{-12}
1	73.9936	7.8369×10^{-18}
(NumPy)	0.0000	1.000

The results from the experiment reveal significant variations in the behavior of the Large Language Model (LLM) when randomly choosing between two popular names, "Emma" and "Noah," under different temperature settings. As shown in Table 3, at a temperature of 0, the LLM's selections perfectly conform to a uniform distribution, with a statistic of 0.0000 and a p-value of 1.0000, indicating no deviation from uniformity. However, as the temperature increases, starting from 0.2, the statistic gradually rises, and the p-value decreases, suggesting a growing deviation from random selection. Notably, at a temperature of 0.6, the statistic jumps to 19.7512, and the p-value drops to a markedly low 8.8204×10^{-6}, clearly indicating significant bias in the random selection process under this setting. This contrasts sharply with the baseline.

The experimental outcomes for the chi-square test results comparing a popular and less popular name pair, "Emma" and "Luna," under various temperature settings, are detailed in Table 4. Similar to the experiment above, at a temperature of 0 and 0.2, the LLM's random choices align with a uniform distribution. However, a dramatic shift is observed at a temperature of 0.4, where the statistic surges to 106.3445 with a p-value of6.1967×10^{-25}, indicating significant biases in the random selection process. This trend of increased bias continues as the temperature rises, with statistics peaking at

425.3961 and the p-value reaching its lowest at 1.6328×10^{-94} at a temperature of 0.8.

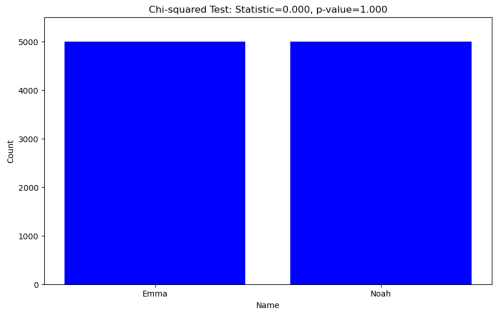

Figure 11: Distribution of Names Generated by NumPy.

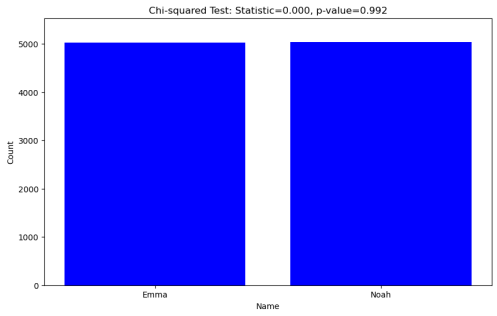

Figure 12: Distribution of Names Generated at 0.4 Temperature.

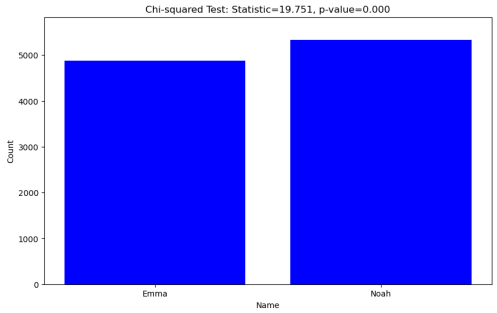

Figure 13: Distribution of Names Generated at 0.6 Temperature.

The results from the two experimental groups exhibit similar trends which suggest that higher temperatures introduce significant biases into the LLM's choice generation. However, bias is introduced later when names of similar popularity are used (0.6 for the pair with popular-popular names and 0.4 for the mixed popularity pair). Moreover, the deviation is more pronounced in the experimental group where there is a greater disparity in popularity. Interestingly, across both sets of experiments, the name "Emma," despite being the more popular name in the second set, is consistently selected less frequently. This consistent under-selection of "Emma," even when it is the more popular name in the mixed popularity group, suggests complexities in the LLM's decision-making process that are not solely influenced by the popularity of the names.

Table 5: Chisquare Test Result for Popular-Less Popular Names

Temperature	Statistics	p-value
0	0.0000	1.0000
0.2	0.0162	0.8985
0.4	106.3445	6.1967×10^{-25}
0.6	224.7732	8.2273×10^{-51}
0.8	425.3961	1.6328×10^{-94}
1	275.0253	9.1135×10^{-62}
(NumPy)	0.0000	1.000

6 DISCUSSION

The findings from the whole number generation experiment have significant implications for the broader motivation of this study - understanding the random generation behavior of large language models (LLMs) and its impact on tasks such as evading LLM watermarks. The results demonstrate that the temperature parameter plays a crucial role in determining the degree of randomness in LLM outputs. At lower temperatures, the LLM struggles to produce uniformly distributed random data, deviating significantly from the expected uniform distribution. This aligns with the concern raised in the motivation that LLMs may not be capable of following complex prompts involving random generation, particularly in the context of evading watermarks like the SOTA method described. However, the results also reveal that at higher temperatures, LLMs can surprisingly surpass traditional random number generators in producing uniformly distributed data. This unexpected finding suggests that, when appropriately tuned, LLMs may possess the capability to generate highly random content, potentially enabling more effective watermark evasion techniques. Consequently, these results underscore the importance of thoroughly investigating LLM random generation behavior, as it could have significant implications for the robustness and effectiveness of watermarking methods designed to detect AI-generated content.

The float number generation suggests that PALM can reasonably generate uniformly distributed numbers at typical positive temperatures. It is an open problem on how PALM did this. It is not clear whether there is such training data, such as random number tables so that the model learns from that. It is also possible this is achieved because the PALM understands the prompt and successfully made out-of-distribution generation. Also, arbitrary random generation tasks can be done by transforming from the uniform distribution so this experiment is beneficial for all random generation tasks. Besides, it is an open problem to see the behavior of LLMs when they are prompted to generate from a non-uniform distribution. Will they avoid generation with low probabilities (like beyond 3σ from the mean)? Can they do more complicated, correlated, $2 - d$

or higher generations? All these are not easy for humans yet are not explored for LLMs as far as we know.

Different from the previous two experiments, we found PaLM struggled on the random phrases generation tasks. No matter what parameters we provided to the PaLM, there are patterns or biases shown in the results. At temperature 0, it repeats the words in order without any randomness involved. This can be easily seen as a watermark and detected by humans or other models. As temperature increases, the model starts to show its preferences. However, in contrast to the intuition, the model doesn't have a bias toward popular words. Instead, it generated much less-used words. This kind of counter-intuitive preferences might not be easily detected by humans but can be captured by some well-trained models focusing on the watermarks.

The results of the name generation experiment contrast with those of the previous number generation experiments and common understanding. Typically, as the temperature parameter increases, the probability distribution of the outputs becomes flatter and broader, enhancing the likelihood of selecting even less probable words. This contributes to increased diversity and creativity in generated text, resulting in richer and more unpredictable content. However, the name generation experiment shows that when the temperature is raised, significant bias is introduced, and the probability distribution becomes sharper, reducing diversity. A possible reason is that the model was exposed to data with specific biases during training, such as some names appearing more frequently than others, causing the model to favor certain options even at higher temperature settings.

The same reasoning might explain why "Emma," the most popular name, was less frequently selected in both experiments. However, a more rigorous explanation may require statistical analysis of PALM's training dataset. We can only conclude that the popularity of names does not wholly determine the emergence and direction of biases.

The presence of bias is good news for the design of LLM watermarks. However, whether this bias is widespread and stable, and the mechanisms behind it, cannot be fully explained by the content of this experiment alone. Future experiments need to be more rigorously designed, including testing more name pairs, quantitatively assessing popularity, and stricter control of variables. These are the steps we can take moving forward.

In all, the results of this experimental design are significant for understanding how LLMs handle generation tasks. They not only reveal the impact of the temperature parameter on output biases but also emphasize the need to carefully consider the quality and structure of the model training data when using LLMs for content generation. For future research, we recommend a more rigorous experimental design, including testing a broader array of name samples, finer adjustments to temperature settings, and employing a variety of statistical methods to assess the results.

7 CONCLUSION

In conclusion, this study has provided valuable insights into the capabilities and limitations of large language models (LLMs) in generating random sequences. The experiments conducted across different domains, including whole numbers, floating-point numbers, phrases, and human names, have revealed that LLMs exhibit a complex interplay between temperature settings and the uniformity of their random generation. While higher temperatures were initially expected to increase randomness, the results painted a more nuanced picture. Notably, the whole number generation experiment demonstrated that at lower temperatures, LLMs struggle to produce uniformly distributed random integers, often converging on specific values or exhibiting significant biases. However, as the temperature increased, the distributions became more uniform, even surpassing traditional random number generators in some cases. This finding aligns with the motivation of understanding LLM behavior in the context of watermark evasion techniques, where the ability to generate highly random content is crucial. Interestingly, the phrase and name generation experiments revealed instances where higher temperatures did not necessarily lead to more uniform distributions, challenging the assumption of increased randomness. These results underscore the complex nature of LLM random generation and the potential influence of factors such as prompt engineering, model architecture, and domain-specific biases. Overall, this research contributes to the growing body of knowledge on the strengths and limitations of LLMs, particularly in tasks requiring high levels of randomness. The findings not only advance our understanding of these models' inner workings but also have practical implications for domains relying on random generation, such as watermarking algorithms and simulations. As the field of artificial intelligence continues to evolve, further exploration of LLM random generation capabilities will be crucial for leveraging their full potential while mitigating potential biases and ensuring robust and reliable performance.

REFERENCES

[1] John Kirchenbauer, Jonas Geiping, Yuxin Wen, Jonathan Katz, Ian Miers, and Tom Goldstein. 2023. A watermark for large language models. In *International Conference on Machine Learning*. PMLR, 17061–17084.

[2] Marc-André Schulz, Barbara Schmalbach, Peter Brugger, and Karsten Witt. 2012. Analysing humanly generated random number sequences: a pattern-based approach. *PloS one* 7, 7 (2012), e41531.

[3] Peter Sprent and Nigel C Smeeton. 2016. *Applied nonparametric statistical methods*. CRC press.

[4] Leonard Tang, Gavin Uberti, and Tom Shlomi. 2023. Baselines for Identifying Watermarked Large Language Models. *arXiv preprint arXiv:2305.18456* (2023).

[5] Qilong Wu and Varun Chandrasekaran. 2024. Bypassing LLM Watermarks with Color-Aware Substitutions. *arXiv preprint arXiv:2403.14719* (2024).

[6] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. 2024. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems* 36 (2024).

A APPENDIX

A.1 Whole Number Generation

A.2 Floating Number Generation

The visualization results from 0.4, 0.6, and 0.8 temperatures are listed here as they are visually indistinguishable from 0.2 temperature and the NumPy random generation baseline.

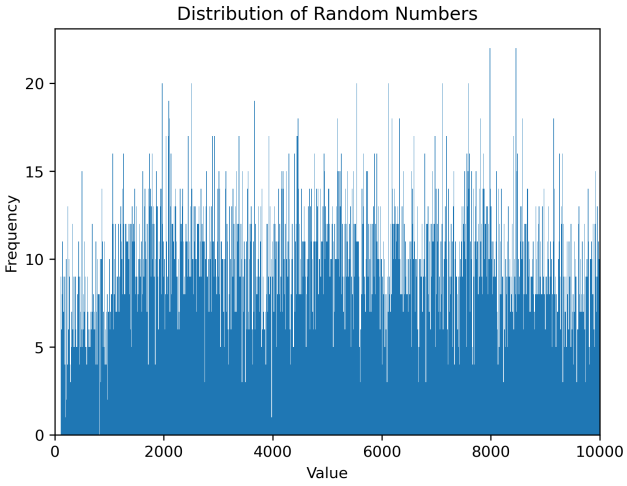

Figure 14: Distribution of Random Whole Numbers at 0.4 Temperature.

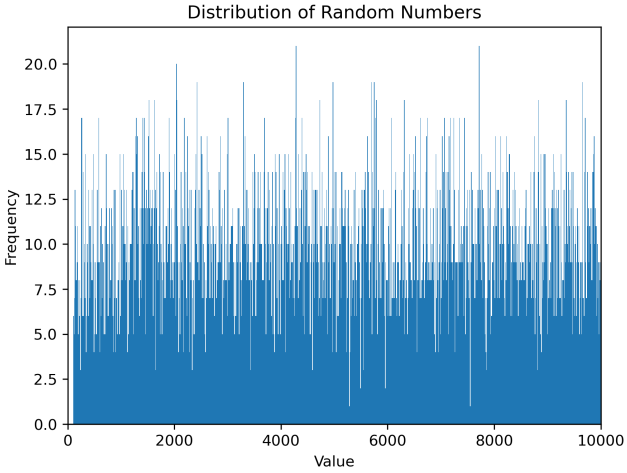

Figure 15: Distribution of Random Whole Numbers at 0.6 Temperature.

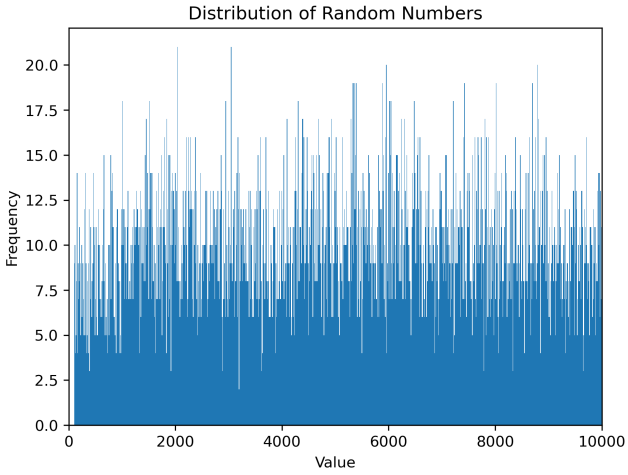

Figure 16: Distribution of Random Whole Numbers at 0.8 Temperature.

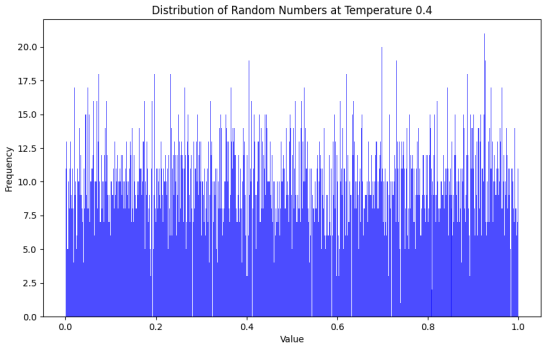

Figure 17: Distribution of Random Float Numbers at 0.4 Temperature.

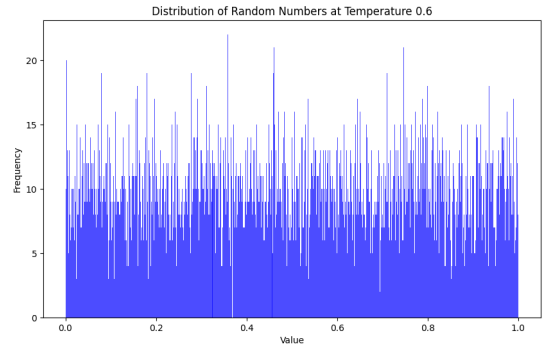

Figure 18: Distribution of Random Float Numbers at 0.6 Temperature.

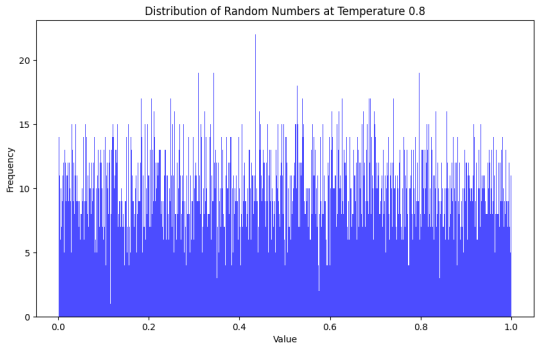

Figure 19: Distribution of Random Float Numbers at 0.8 Temperature.

A.3 Name Gerneration

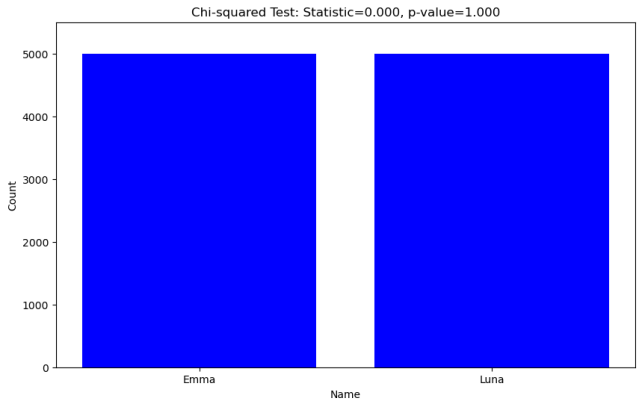

Figure 20: Distribution of Names Generated by NumPy.

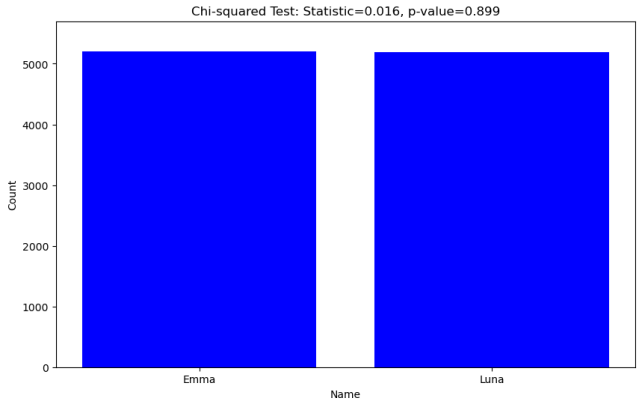

Figure 21: Distribution of Names Generated at 0.2 Temperature.

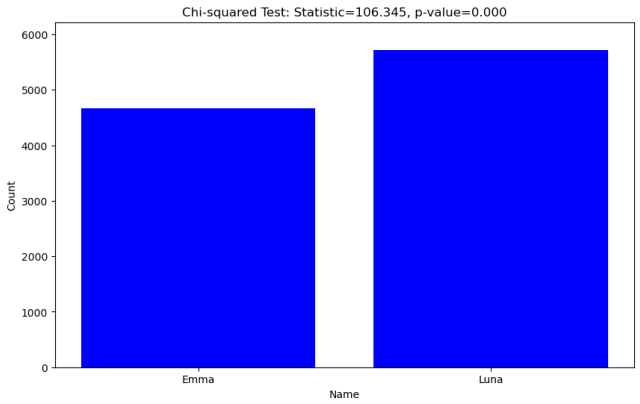

Figure 22: Distribution of Names Generated at 0.4 Temperature.