Are Language Models Any Good at Density Modeling?

Sriram Ranga[1], **Sai Shashank Bedampeta**[2], **Rui Mao**[1,*], **Anupam Chattopadhyay**[1]

[1]Nanyang Technological University, Singapore
[2]Vellore Institute of Technology, Vellore
sriram011@e.ntu.edu.sg, saishashank.bedampeta2021@vitstudent.ac.in, {rui.mao, anupam}@ntu.edu.sg

Abstract

Large Language Models (LLMs) surprised the world with their ability to mimic humans in writing and are starting to be used as simulations of human writers for various kinds of linguistic analyses. However, these analyses rest on the belief that LLMs are good density models that accurately capture the underlying probability distribution of the language. In this paper, we question this basic assumption and try to evaluate language models on their density modelling capabilities. Since a ground truth does not exist for the probability distribution of any natural language, we come up with a synthetic language made up of decimal numbers written in words in English. We train language models from scratch on various probability distributions over this synthetic language and compare the distributions learned by the models with the original distributions. Experiments show that language models can learn underlying probability distributions across a wide range of cases, but they fail when those distributions depend on deep semantic properties of numbers that cannot be inferred from syntactic patterns. Additionally, we observed a strong bias in the models towards numbers that frequently occur as substrings within other numbers. This suggests that such a bias possibly exists in real-world natural language models as well, and negatively impacts downstream tasks and analyses that rely on model-generated probabilities.

1 Introduction

Language models are autoregressive probabilistic models pre-trained to maximize the probability of generating the training data, along with some regularization that helps them to generalize well to new text that is reasonably close to the training data. Before they are trained to become chat bots using instruction tuning, these models are expected to capture the probability distribution of the whole language, thus acting as density models. While linguists debate whether something distinct and complete called a natural language exists and whether it can be wholly quantified and captured by data (Birhane and McGann 2024), engineers continue to train these language models on massive amounts of data, and Natural Language Processing (NLP) researchers have started using them as a joint simulation of all human writers for different kinds of analyses (Meister et al. 2021). Whether such a true distribution exists or not, pre-trained (as opposed to instruction-fine-tuned) language models are assumed to be good at density modelling, but it is difficult to evaluate LLMs trained on English and other natural languages on this particular capability of theirs, since a known ground truth for the distribution does not exist.

Small-scale experiments in the work of Ilia and Aziz (2024) involving human participants show a low level of calibration between next-word probability distributions given by the human participants and by the language models. Meister and Cotterell (2021) took another approach and used corpus-level metrics like word rank frequencies, sentence length distribution, stop word rates, etc., to show a misalignment between the corpus properties and those of LLM-generated text. Mao et al. (2025) argued that humans and ChatGPT rely on different conceptual frameworks, leading to a misalignment between them. These observations challenge the assumed density modelling capabilities of language models and beg for a proper answer to the question: *How good are language models at density modelling?*

It is difficult to answer the question for natural languages with all their complexities and a lack of an accepted ground truth. Therefore, to gain clearer insights, we instead analyse a controlled setting involving a synthetic language and a range of popular probability distributions. We create a language out of numbers from "zero" to "two million" represented in words, which makes it easy to order the members of the language on an axis and assign them probabilities in meaningful ways. Additionally, this representation has a good amount of syntactic and semantic complexity that could help with the generalizability of the results into natural languages. We train a GPT-2-like language model (Radford et al. 2019) from scratch on these numbers drawn from various probability distributions and compare the distributions learned by the model with them. The choice of distributions include commonly occurring one-dimensional discrete distributions like the uniform distribution, a linear distribution, the binomial distribution (see Fig. 1), as well as distributions that are specially crafted to test the ability of the model to understand deep semantic properties of numbers, for instance, their divisibility by 3.

We find that models we trained are able to understand the syntax of the language fairly well (close to 0.5% error rate), and are able to learn the distributions as long as they can

*Corresponding author: Rui Mao

use syntactical hints to guess the probabilities. When that is not possible, models collapse to an approximate distribution that fails to capture the preferences of the original distribution. We also observe that models have a strong bias for numbers like "one" that are frequently seen as a part of other numbers like "one hundred and ...". This means that models discount the fact that these numbers are observed more often in the presence of certain suffix contexts and bump up their probabilities even in the absence of them. This can have an impact in studies which use model probabilities for downstream analysis tasks. This perspective links to an assumption made in our previous work (Ranga et al. 2025) that the distribution of sentences in the English language can be approximated as the probabilities that the model assigns to the sentences for being at the beginning of passages. To explore this link, we modify the dataset into passages of numbers and find that in the case of our synthetic language, this assumption still holds, thus providing support for the validity of the original assumption as well.

Our contributions through this paper can be summarized as follows: (1) We evaluate language models trained from scratch on their density estimation capabilities for the first time, and find out the conditions in which the models are able to learn the distributions accurately and those in which they fail to do so. (2) We highlight a strong bias in the models for members of the language that frequently appear as substrings in other members, which cautions against their use for human preference analysis but simultaneously provides evidence for the validity of an assumption we made in our previous work (Ranga et al. 2025).

2 Related Work

Using LLM probabilities. Many NLP works use probabilities from LLMs to analyse their preferences and learn about the data they are trained on or can generalize to. Hu and Levy (2023) make the distinction between the model probabilities for text completion and question answering, and claim that the latter are not reliable. Wagner and Abend (2025) observe that NLP research often misinterprets LLM output probabilities by assuming that preference-tuned model probabilities, where models are doing response prediction, would be the same as for pre-trained models, where they would be doing density estimation. Conversational models fine-tuned to provide correct answers tend to move away from the source distribution towards the "correct" generations that maximize the objective function values.

Testing LLM probabilities. Meister and Cotterell (2021) analysed the accuracy of probability distributions from language models by comparing corpus-level metrics of human-generated data and model generated data. They observed major differences in a lot of metrics like word rank frequencies, sentence length distribution, stop word rates, etc., which shows that the models cannot perfectly capture the distribution underlying their training data. Ilia and Aziz (2024) on the other hand, looked at next-word distributions for a set of passages from human participants and language models, and found out that they are not well aligned with each other. Gupta et al. (2025) show evidence for biased priors in LLMs by exploring their preferences for heads in coin tosses. They even observed a tokenization level bias for heads (single token) over tails (multiple tokens), which indicates more instances of heads than tails in the training data, thus being able to link the model bias to a bias in the training data. Biased predictions can also result from the design and position of prompts, the way label space is divided, and the label taxonomy (Mao et al. 2023). Gu et al. (2025), Hopkins, Renda, and Carbin (2023) explore a related ability of LLMs as random generators that generate samples according to distributions described using natural language and mathematical expressions.

The above works that are related to using LLM probabilities mainly highlight the lower accuracy of the probabilities of instruction-tuned models compared to pre-trained models, but they do not question the validity of the pre-trained model distributions in the first place. Those that work on testing these model probabilities, use alternative or approximate metrics that do not give us enough evidence to form an understanding about how capable models are at learning widely different distributions.

3 Preliminaries

Large Language Models (LLMs) are deep generative models trained to model a probability distribution over sequences of text. Given sequences of tokens like $x = \{x_1, x_2, \ldots, x_T\}$, LLMs are trained to maximise the log-likelihood of the observed data:

$$\mathcal{L}(\theta) = \sum_{t=1}^{T} \log P_\theta(x_t \mid x_{<t})$$

Here, θ denotes the learnable model parameters and P_θ is the learned conditional probability distribution. This objective encourages the model to assign higher probabilities to token sequences that are similar to those observed in natural language. LLMs trained in this fashion can be used to generate text *autoregressively*, predicting each token sequentially conditioned on the tokens preceding it. At each time step t, the next token is sampled from the conditional distribution as:

$$x_t \sim P_\theta(x_t \mid x_{<t})$$

This process continues until a termination condition is met, such as generating a special end-of-sequence token $\langle \text{eos} \rangle$. The probability that a model generates a sequence x is thus given by:

$$P_\theta(x) = \prod_{t=1}^{T} P_\theta(x_t \mid x_{<t})$$

In this work, we calculate probabilities assigned by models to numbers following the same formula as above by tokenizing the number into a sequence of words.

Uniform Distribution: A discrete random variable X is said to follow a *uniform distribution* over a finite set $\{a, a+1, \ldots, b\}$ if its probability mass function (PMF) p_X follows:

$$p_X(x) = \frac{1}{b-a+1}, \quad \text{for } x \in \{a, \ldots, b\}.$$

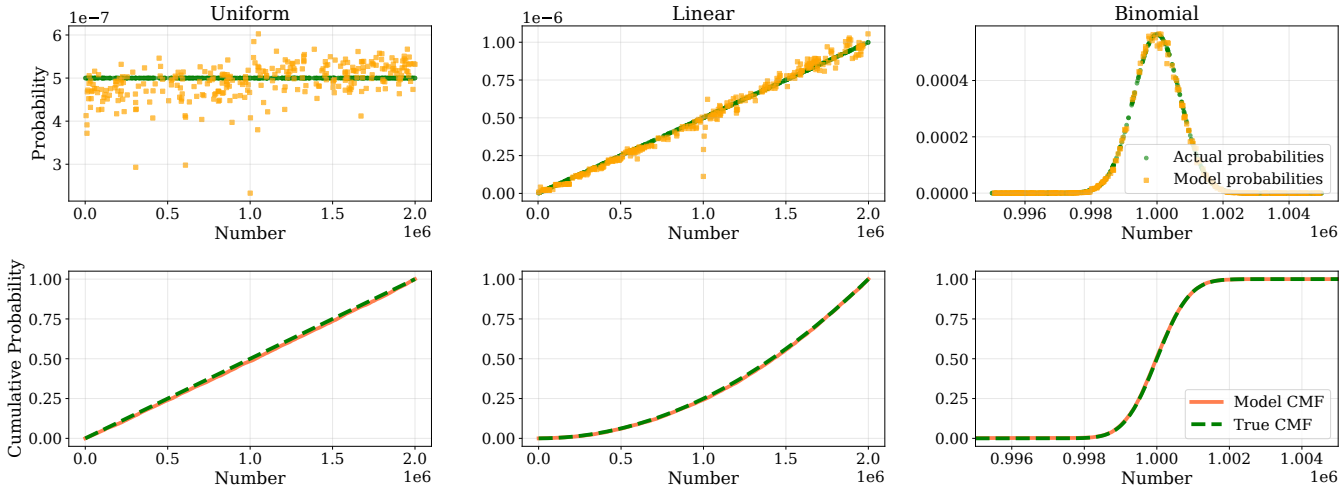

Figure 1: PMF (top) and Cumulative Mass Function (CMF) (bottom) graphs for uniform, linear and binomial distributions comparing the distribution used to train the model and the distribution captured by the model. A zoomed in version focusing around the mean is shown for the binomial distribution since the probability mass is highly concentrated in the centre. The near 0 value of the CMF for lower numbers and near 1 value for higher numbers indicates that the model captures the probabilities for the rest of the areas accurately.

Binomial Distribution: The *binomial distribution* models the number of successes in n independent Bernoulli trials, each with success probability $p \in [0, 1]$. A binomial random variable $X \sim \text{Bin}(n, p)$ has the PMF:

$$p_X(x) = \binom{n}{x} p^x (1-p)^{n-x}, \text{ for } x = 0, 1, \ldots, n.$$

Apart from the above standard distributions, we define two new ones in this paper to test the learning capabilities of language models:

Linear Distribution: A discrete random variable X is said to follow a *linear distribution* over a finite set $\{a, a+1, \ldots, b\}$ if:

$$p_X(x) = \frac{2(x-a)}{(b-a+1)(b-a)}, \quad \text{for } x \in \{a, \ldots, b\}.$$

Sinusoidal Distribution: A discrete random variable X is said to follow a *sinusoidal distribution* with a middle value P_0 and period T over a finite set $\{a, a+1, \ldots, b\}$ if:

$$p_X(x) = P_0 \times (1 + sin(2\pi x/T)), \quad \text{for } x \in \{a, \ldots, b\}.$$

Once a time period T is selected, P_0 is set to a value that normalizes the PMF values and makes them sum up to 1.

KL Divergence: Kullback–Leibler divergence is a measure of how much an approximating probability distribution p is different from a true distribution q. It is mathematically defined as:

$$D_{\text{KL}}(q\|p) = \sum_x q(x) \log \frac{q(x)}{p(x)} = \mathbb{E}_{x\sim q}\left[\log \frac{q(x)}{p(x)}\right]$$

When the sample space cannot be covered exhaustively to compute the quantity, a standard Monte Carlo estimator can be employed by sampling $x_i \sim q$ to approximate it as

$$\hat{D}_{\text{KL}}(q\|p) = \frac{1}{N} \sum_{i=1}^{N} \big(\log q(x_i) - \log p(x_i)\big)$$

However, this estimate can have a high variance. Therefore, we use a low variance unbiased estimator (Schulman 2020) for a better approximation:

$$\tilde{D}_{\text{KL}}(q\|p) = \frac{1}{2N} \sum_{i=1}^{N} \big(\log q(x_i) - \log p(x_i)\big)^2.$$

4 Methodology

The primary aim of this work is to measure how well a language model captures the probability distribution underlying its training data. However, there is no known ground truth distribution for natural language, therefore we consider artificially crafted languages. Ideally, we would prefer to use a language very close to a natural language and assign probabilities to its members according to various distributions. However, assigning meaningful probabilities to members of a natural language is challenging in its own regard. This is partly due to the infinite size of usual natural languages, but mainly due to the fact that there is no straightforward way to order all the members of the language in one or more dimensions in a meaningful fashion that allows us to use well defined PMFs on them. Therefore, we propose a synthetic language whose members can be meaningfully ordered, which lets us evaluate models on how well they can learn a variety of probability distributions over this language. We define the

Number	In Words
12	twelve
109	one hundred and nine
46,707	forty six thousand seven hundred and seven
154,029	one hundred and fifty four thousand and twenty nine
2,000,000	two million

Table 1: Samples from the synthetic language used in the paper: numbers and their word representations.

language as the collection of numbers from 0 to 2,000,000, written in words. A few examples are given in Table 1.

We chose to represent the numbers in words as opposed to the standard Hindu-Arabic (Britannica 2025) notation, because we wanted to hide the values and relationships among the numbers behind some syntactic and semantic complexity. This was done to bring the synthetic language's complexity closer towards that of natural language. For example, identifying that both "three thousand one hundred and forty" and "three hundred and one thousand and forty" are correctly formed numbers, but "three hundred one thousand and forty" is incorrectly formed requires the model to learn the syntax rules of the language. However, when written in the Hindu-Arabic notation, there is not much scope for the model to commit syntactical mistakes. Additionally, consider that we are assigning probabilities to numbers according to a smooth distribution, and numbers close by have similar PMF values, and those that are far away have different PMF values. Taking the same previous example of the first two numbers, they differ only in the order of "hundred" and "thousand" (and an additional "and" required for correct syntax), but the numerical representations "3,140" and "301,040" very clearly reveal that the numbers are very different. By only giving this "in words" representation of the numbers to the models, we are making them work harder to be able to learn the same information, closer to the complexity they need to handle in natural language scenarios. That way, we hope, it would be more likely for the findings obtained for this synthetic language to generalize to natural ones.

Let L denote this synthetic language containing $N_L = 2,000,001$ numbers in words. Firstly, a word-level tokenizer tk is used to identify the vocabulary V, containing n_V words. In each experiment, n numbers out of the N_L numbers in the language are sampled using a carefully chosen distribution P_{true} and are used to train a language model M that is initialized with random weights. Let the trained model represent a distribution P_{model} of all the possible strings that can be created by concatenating tokens from V. We compare P_{true} with P_{model} to understand how well the model learned the distribution underlying its training data.

5 Experiments

In this section, we first describe the setup used for the experiments in Sec. 5.1. We then see in Sec. 5.2 whether a model is able to learn the syntax of this synthetic language with a limited number of examples. Next, in Section 5.3, we test the ability of the model to capture various distributions involving patterns of different levels of complexity. In Section 5.4 we discuss our observation of a bias in models that makes them assign higher probabilities to samples which appear frequently as substrings of other samples. Finally in Section 5.5, we verify the validity of an assumption made in the work of Ranga et al. (2025) regarding the distribution of sentences in the English language, by testing it on the synthetic language.

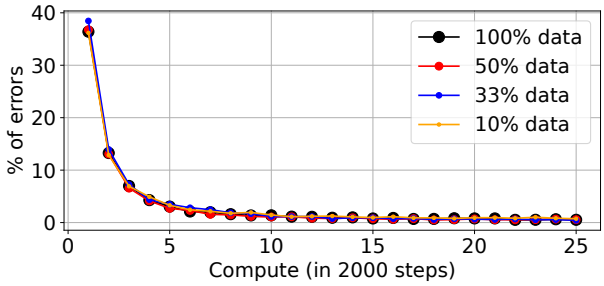

Figure 2: Percentage of incorrectly formed numbers generated by the model, as a function of the amount of compute used to train the model. The four different lines correspond to different amounts of the dataset sampled from the uniform distribution that the model is trained on.

5.1 Experimental Setup

Dataset. The synthetic dataset consisting of numbers from 0 to 2 million, was constructed using the Python package "num2words" (Ogawa and Contributors 2025). The reconstruction of the numbers from their word representations was done using a custom grammar and parser written in Python. The vocabulary size was calculated to be 37, which includes 32 words that make up the numbers ("zero" to "nineteen", multiples of ten from "twenty" till "ninety", "hundred", "thousand", "million" and "and"), a "." used as a separator between numbers (when using the passage mode described in Section 5.5), ⟨bos⟩, ⟨eos⟩ to mark the beginning and end of the text, ⟨pad⟩ for padding and ⟨unk⟩ to handle errors due to any unknown symbols. A simple word-level tokenizer was used with white space characters for delimitation. 10% of the numbers from the language were sampled (following i.i.d. sampling), to build the dataset for training.

Model and training. The HuggingFace transformers library was used to define and train the tokenizer as well as the model. A model closely following the GPT-2 architecture, initialized with random weights, was used for all the experiments. The model uses the embedding dimension of 256, has 4 heads, and is made up of 8 layers of decoder blocks, which comes up to over 6.3 million parameters in total. A context window of 20 tokens was selected based on the maximum tokens observed for a single number in the dataset. For experiments in passage mode (see Section 5.5), the context window was scaled up corresponding to the number of numbers in the passage. Training was done with the standard cross-entropy loss minimization objective, with the default

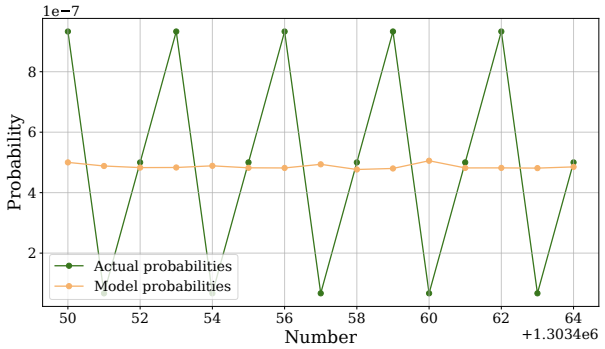

Figure 3: It can be observed that models fail to capture distribution patterns that require some semantic processing of the members of the language. For example, in this case, where the probabilities of samples depend on their divisibility by 3, the syntactic structure of numbers (like "twenty seven" or "one hundred and five") does not help to tell multiples of 3 apart from those which leave a remainder of 1.

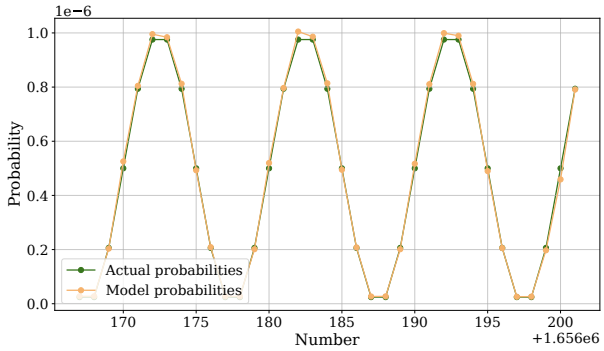

Figure 4: Distribution patterns that can be derived from syntactical clues are being captured by the model accurately. In this case, the phase of a sine wave with a periodicity of 10 can be captured using the sample's suffix that corresponds to its units part. For example, "one hundred and *twelve*" and "one hundred and thirty *two*") can be matched based on their last words *twelve* and *two*.

dropout rate of 0.1 for regularization. All the experiments were run on a Linux based system using a single NVIDIA RTX 6000 GPU that has 48 GB of RAM.

5.2 Learning the Syntax of the Language

One of the main reasons for choosing the "in words" representation of numbers for the language over the Hindu-Arabic representation is that it has the additional component of syntactical complexity. The members of the language strictly follow a defined grammar, and some combinations of words from the vocabulary resemble members of the language very closely, but are incorrect constructions as per the grammar. Hence, we first try to understand how well models are able to follow the syntax of the language. We train models on various percentages of the language, sampling from these datasets of various sizes using a uniform distribution.

We observe as shown in Figure 2 that the percentage of incorrectly formed numbers drops rapidly to less than 1% but training them for multiple epochs was still insufficient to eliminate syntax errors entirely. Additionally, we found that increasing the percentage of the language used for training, while maintaining the number of training steps as it is, did not contribute significantly to a drop in the error rate. Even the model trained with 10% of the data generates incorrectly formed samples only less than 1% of the time, therefore it is used for the rest of the experiments in the paper. We exclude this small portion of errorful generations from the analyses in the rest of the sections and move ahead to find out whether models are able to capture the underlying distributions of their training data or not.

5.3 Learning Distributions

To understand how a model might be able to learn a given probability distribution, we need to take a look at how models are trained. Language models are autoregressive next-token predictors and are trained for text generation in an unsupervised manner. To train a model, data is sent to it in mini-batches (of a few samples each), and its parameters are tuned slightly in the direction that maximizes the chance that when we sample from the model, it generates the same data back. The individual samples in a mini-batch influence the parameter update steps based on their relative frequencies in it, and since they come from a certain distribution, each of them has an influence on the model that corresponds to their source distribution's PMF value. Therefore, if an entire finite language is used to train a sufficiently large model for a sufficient number of steps, assuming that the deviations from the distributions in mini-batches get averaged out over a few epochs, the model should approximate the distribution effectively. However, in practical cases, the entire language is not available/used for training the model. The model has to learn patterns from a few samples and learn to generalize to the whole dataset.

To test this generalizability in the specific case of our synthetic language, where all the members can be ordered in a single dimension based on their values, probabilities can be assigned to them based on patterns involving their values. These patterns can be of different levels of complexity, and in this paper, we choose to experiment with two kinds of patterns - one that can be recognized using the syntactic aspects of the language and one that additionally requires the model to understand some semantic aspects as well. To explain the differences between these patterns, we take the help of some example distributions.

First, consider the distributions in which numbers that are closer to each other have closer PMF values, compared to those that are farther away. The numbers that are part of the training dataset get their PMF values boosted relative to their frequency in the dataset. But for the model to generalize and to assign accurate PMF values to unseen data, it needs to learn their relationships with samples that it has already seen. One way they get to know this could be through syntactic clues. For example, the model can learn that "*three hundred and two thousand* two hundred and twenty three"

Distribution	Initial KL	Final KL	Improvement (%)
Uniform	0.2282 ± 0.0011	0.0038 ± 0.0010	98.33
Binomial	82.0749 ± 0.3315	24.8572 ± 0.3247	69.68
Sine - Period 3	0.6880 ± 0.0030	0.7323 ± 0.0136	-6.43
Sine - Period 10	0.4274 ± 0.0043	0.0070 ± 0.0009	98.36
Linear	0.3874 ± 0.0137	0.0290 ± 0.0055	92.51

Table 2: KL Divergence Improvements Across Distributions (Mean and standard deviation over sample sets with different random seeds are reported in the table).

and "*three hundred and two thousand* three hundred and forty one" are farther to each other compared to "*three hundred and two thousand two hundred and* four" and "*three hundred and two thousand two hundred and* ten", based on the length of their common prefix. We trained models on a variety of distributions that have this property of numbers that are close to each other having similar PMF values. Figure 1 shows the model PMF and CMF curves when trained on uniform, linear and binomial distributions. Comparing their plots with those of the true distributions, we can confirm that the models have learned to capture the training distributions accurately.

Next, consider the distributions in which the PMF values are directly related to a semantic property of the number, like the remainder we are left with when it is divided by a particular divisor. For example, if the divisor is 3, the model has to somehow internally calculate the sum of the digits and compare it with 0, 3, 6, or 9, or perform another equivalent test involving implicit computations. As one can observe in Fig. 3, we find that models are unable to identify these kinds of patterns and end up collapsing into a uniform distribution.

Alternatively, consider the case in which the distribution deviates from the uniform distribution in a sine wave-like fashion, and its periodicity is a number that makes the phase of the curve reflect directly in any syntactical element of the number. For instance, if the periodicity of the curve is set to 100, it can compare the sub-hundred portions of "three thousand two hundred and *twenty three*" and "one million six hundred and *twenty three*" and assign similar probabilities to each other. This can be extended to other periodicities which divide numbers like 10, 100, 1000, etc. without leaving a remainder. We experiment with different such periodicities and find that models can learn to estimate probabilities using common suffixes in the numbers. Fig. 4 contains such an example with a periodicity of 10 (which divides all powers of 10).

In order to numerically measure how well the models have learnt the distributions they were trained on, we calculate the KL divergence of the models from the true distributions. Table 2 summarizes the results across different distributions with a test size of 10,000 samples. We observe that there is a consistent reduction in divergence from the initial to the final checkpoint for most distributions, indicating improved alignment between the model's learned and target distribution over successive training steps. However, for sine distribution with period 3 we observe that the model has not been able to learn the distribution, and the KL divergence

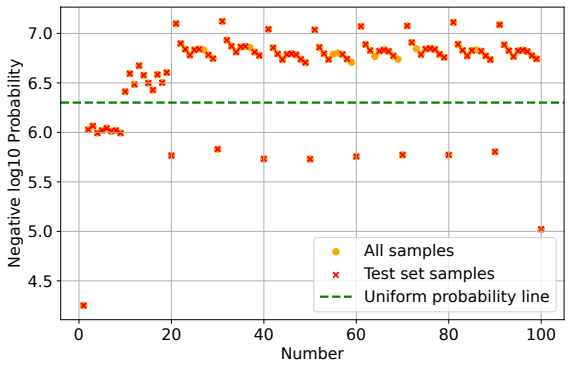

Figure 5: Probabilities assigned by the model trained on all numbers 0 - 2 million for one epoch to the numbers "one" to "one hundred". There is a clear correlation between the dips in this graph and the spikes in Fig. 6. This indicates that the model assigns disproportionately high probabilities to numbers that are frequently encountered as substrings.

values align with the visual indication of the same in Fig. 3. KL divergence for the Binomial distribution is much higher for the Binomial distribution compared to others, because KL divergence values are very sensitive for probabilities that are close to 0. The improvement in the value indicates good learning, but a better understanding can be gained for this scenario from the visual in Fig. 1.

5.4 Substring Bias

In all the above experiments, we observed anomalies in the model probabilities for the numbers from "one" to "one hundred". Specifically, if we take the case of a model trained on data sampled from the uniform distribution, we found that members representing certain numbers had their probabilities consistently and significantly boosted over the ideal uniform value of $N_L^{-1} = 10^{-6.301}$, and also above others in the neighbourhood. There was a pattern in the numbers with abnormal PMF values, as it can be seen in Fig. 5. All the numbers from "one" to "ten", and those that are multiples of ten had higher probabilities assigned to them, with "one" having the highest of all. This prompted us to look at the number of times each of these members appears as substrings in all other numbers in the dataset (see Figure 6). There exists a very strong correlation between the abrupt jumps in the number of substring occurrences and the probability values. Additionally, the number "one hundred" has a much higher probability than the rest of the multiples of ten, and notably the word "hundred" appears in more than 99.99% of the members of the dataset.

These two observations led us to conclude that the model exhibits a strong bias for frequently appearing substrings, even when they are not presented in the exact same context. Note that this behaviour is not just limited to models trained on the uniform distribution. For example, a model trained on the sinusoidal distribution with a periodicity of 10 also exhibits the same properties (see Fig. 7).

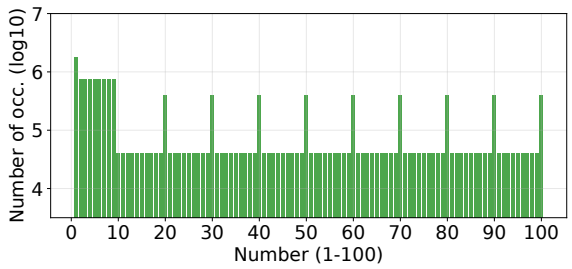

Figure 6: Number of times each number from "zero" to "one hundred" appear in the numbers "zero" to "two million". Observe the correlation between this graph and Fig. 5.

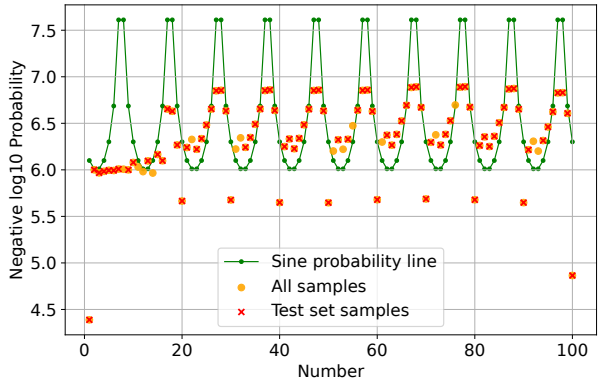

Figure 7: When trained on a sinusoidal distribution with period=10, the model distribution mimics the training distribution, but is influenced by the common substring bias.

5.5 Context Independence of the Model Distribution

In our previous work (Ranga et al. 2025), we used the distribution of probabilities assigned by the original GPT-2 model to sentences of the English language to make predictions about a potential saturation of content on the internet. Our analysis was based on two assumptions: (1) The GPT-2 model, trained using documents on the internet, has correctly learned the true distribution of the sentences of the English language. (2) This true distribution can be approximated as the distribution of sentences being at the beginning of the passage/document without any preceding context.

For Assumption 2 to be true, models should be flexible about the context that precedes a sentence in a passage. The bias observed in Section 5.4 can be interpreted as the relaxation given by the model to the context of substrings while assigning them probabilities. In order to link this with Assumption 2 mentioned above, we repeat some of the experiments of Section 5.3 with the data given to the model in the **passage mode**. Here, instead of feeding individual numbers to the models, we group them into a passage, by concatenating them with a "." symbol as a separator. When we trained models with different passage lengths, we found out that their first number distributions remained largely similar to the original distributions, thus providing support for the assumption in question. The same can be observed in Fig. 8 where the probabilities for different numbers to appear at the beginning of the passage are shown in a histogram.

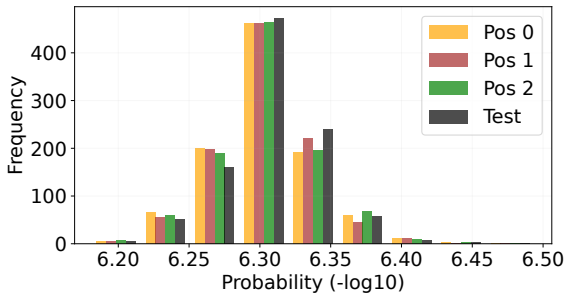

Figure 8: The probabilities assigned to numbers by a model trained to learn the uniform distribution in the passage mode with three numbers per passage. Colours indicate the positions of numbers in the training passages.

6 Concluding Remarks

In this paper, we explored the ability of language models to learn various kinds of probability distributions depending on different levels of complexity of the data and its properties. We derived insights on the mechanisms using which models are able to learn these distributions and highlighted some cases in which this mechanism does not work and models fail to learn the patterns in the distribution. We also identified the "common substring" bias in models that calls for caution when using model probabilities for any downstream analysis tasks, and used our setup to provide evidence in support of an assumption in our previous work (Ranga et al. 2025).

The experiments conducted and analysis performed in this paper are based on a synthetic dataset, but the insights gained are applicable to natural languages as well. While it is a positive result that models are able to learn a wide range of probability distributions accurately, the finer limitations must not be overlooked. Firstly, our observation that trained models do fairly well at generating syntactically correct samples, but are still not perfect, aligns with the observation made about various commercial code generation LLMs struggling to guarantee the syntactical correctness of generated code (Wang et al. 2025). This calls for novel approaches for augmenting the language models with a reliable source of truth, such as the grammar of the language.

Secondly, our observation of models' substring bias has implications in real-life scenarios. If a question-answering model has seen more samples in its training data where the answer is yes, the probability that it answers yes to any question would be higher than what the correct probabilities should be. A concrete example of this was given in the work of Gupta et al. (2025), where they observe that models are biased towards heads compared to tails, while also noting that there are a higher number of heads in the training data compared to tails. Researchers using language model probabilities to make inferences about the training data/language as a whole should be wary of such biases.

Acknowledgments

We would like to acknowledge Prof. Erik Cambria for his valuable suggestions for improving this paper. We would also like to thank Alka Luqman for initial discussions on the ideas that led to the paper as well as for her review. We gratefully acknowledge Google for partial support of this work.

References

Birhane, A.; and McGann, M. 2024. Large models of what? Mistaking engineering achievements for human linguistic agency. *Language Sciences*, 106: 101672.

Britannica, E. 2025. Hindu-Arabic numerals. https://www.britannica.com/topic/Hindu-Arabic-numerals. Accessed 24 July 2025.

Gu, J.; Pang, L.; Shen, H.; and Cheng, X. 2025. Do LLMs Play Dice? Exploring Probability Distribution Sampling in Large Language Models for Behavioral Simulation. In *Proceedings of the 31st International Conference on Computational Linguistics*, 5375–5390.

Gupta, R.; Corona, R.; Ge, J.; Wang, E.; Klein, D.; Darrell, T.; and Chan, D. M. 2025. Enough Coin Flips Can Make LLMs Act Bayesian. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics*, 7634–7655.

Hopkins, A. K.; Renda, A.; and Carbin, M. 2023. Can LLMs Generate Random Numbers? Evaluating LLM Sampling in Controlled Domains. In *ICML 2023 Workshop: Sampling and Optimization in Discrete Space*.

Hu, J.; and Levy, R. 2023. Prompting is not a substitute for probability measurements in large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 5040–5060. Singapore: Association for Computational Linguistics.

Ilia, E.; and Aziz, W. 2024. Predict the Next Word: <Humans exhibit uncertainty in this task and language models _____>. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 2: Short Papers)*, 234–255. St. Julian's, Malta: Association for Computational Linguistics.

Mao, R.; Chen, G.; Li, X.; Ge, M.; and Cambria, E. 2025. A Comparative Analysis of Metaphorical Cognition in ChatGPT and Human Minds. *Cognitive Computation*, 17(35): 1–12.

Mao, R.; Liu, Q.; He, K.; Li, W.; and Cambria, E. 2023. The Biases of Pre-Trained Language Models: An Empirical Study on Prompt-Based Sentiment Analysis and Emotion Detection. *IEEE Transactions on Affective Computing*, 14(3): 1743–1753.

Meister, C.; and Cotterell, R. 2021. Language Model Evaluation Beyond Perplexity. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, 5328–5339. Association for Computational Linguistics.

Meister, C.; Pimentel, T.; Haller, P.; Jäger, L.; Cotterell, R.; and Levy, R. 2021. Revisiting the Uniform Information Density Hypothesis. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 963–980. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics.

Ogawa, T.; and Contributors. 2025. num2words: Convert numbers to words in multiple languages. https://pypi.org/project/num2words/. Version 0.5.14. Accessed: 2025-07-29.

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

Ranga, S.; Mao, R.; Cambria, E.; and Chattopadhyay, A. 2025. The Plagiarism Singularity Conjecture. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, 10245–10255. Albuquerque, New Mexico: Association for Computational Linguistics.

Schulman, J. 2020. Approximating KL Divergence. http://joschu.net/blog/kl-approx.html. Accessed: 2025-11-17.

Wagner, E.; and Abend, O. 2025. What do Language Model Probabilities Represent? From Distribution Estimation to Response Prediction. arXiv:2505.02072.

Wang, Z.; Zhou, Z.; Song, D.; Huang, Y.; Chen, S.; Ma, L.; and Zhang, T. 2025. Towards understanding the characteristics of code generation errors made by large language models. In *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*, 717–717. IEEE Computer Society.