Can LLMs Generate Random Numbers? Evaluating LLM Sampling in Controlled Domains

LLM Sampling Underperforms Expectations

Anonymous Author(s)
Affiliation
Address
`email`

Abstract

Practitioners frequently take multiple samples from large language models (LLMs) to explore the distribution of completions induced by a given prompt. While individual samples can give high-quality results for given tasks, collectively there are no guarantees of the distribution over these samples induced by the generating LLM. In this paper, we empirically evaluate LLMs' capabilities as distribution samplers. We identify core concepts and metrics underlying LLM-based sampling, including different sampling methodologies and prompting strategies. Using a set of controlled domains we evaluate the error and variance of the distributions induced by the LLM. We find that LLMs struggle to induce reasonable distributions over generated elements, suggesting that practitioners should more carefully consider the semantics and methodologies of sampling from LLMs.

1 Introduction

Practitioners frequently take multiple samples from large language models (LLMs) to explore the distribution of completions induced by a given prompt. This broad methodology surfaces in many areas, from sampling synthetic data for training machine learning models [14] to sampling multiple candidate solutions to a given task [1], to ensuring that completions satisfy certain constraints [4]. In each of these tasks, LLMs show great promise: their generated outputs are often more realistic than those of other synthetic data generation techniques [14] or more accurate than other machine learning approaches. There are now instances of LLMs producing prototyped interview responses for HCI research [10], unit testing for software [16, 20], and even training data for ERM-based learning algorithms [5, 19].

LLMs as distribution samplers. A core assumption in repeatedly sampling with LLMs is that they induce a consistent distribution over output generations such that the sampling yields useful results for a given task. However, there is comparatively little evidence, in either academic papers or in folk wisdom, on how well LLMs abide by this assumption. Indeed, they often do not: Figure 1 presents

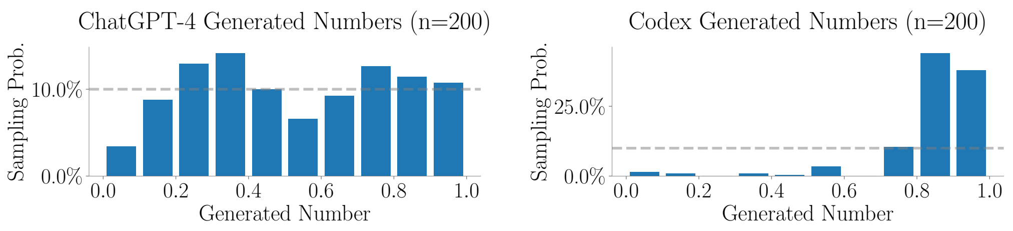


Figure 1: Histogram of results when prompting ChatGPT-4 (left) and Codex (right) to generate a uniform distribution over $[0, 1)$. The resulting distributions are not uniform.

```
Here are 100 samples from [0, 1):
* 0.23
* 0.12
* [SINGLE COMPLETION]
```

(a) Non-autoregressive (NARS) sampling, in which a single completion is generated at a time. To evaluate the induced distribution we evaluate the perplexity of each candidate completion (e.g., 0.00, 0.01, . . . 0.99).

```
Here are 100 samples from [0, 1):
* 0.23
* 0.12
* [FIRST COMPLETION]
* [SECOND COMPLETION]
* [THIRD COMPLETION]
```

(b) Autoregressive (ARS) sampling, in which multiple completions are generated at once.

Figure 2: Non-autoregressive v.s. autoregressive sampling methodologies.

an illustrative example of the distributions generated by prompting state-of-the-art LLMs for uniform distributions of numbers between 0 and 1. The induced distributions are far from uniform, motivating the underlying question of this work: can we trust LLMs to produce a given distribution?

There are no established best practices for sampling data from LLMs to generate data from a desired distribution, nor are there established metrics of success. The closest measure is the notion of *calibration*, which is the degree to which the probabilities output by a classification model match the probabilities of that class being correct in test data. However, this concept alone is insufficient to fully understand the quality of a distribution induced by an LLM.

Contributions. Our contributions are as follows. First, we introduce new vocabulary distinguishing methodological approaches to distribution sampling with LLMs: *non-autoregressive* sampling (NARS) and *autoregressive* sampling (ARS) (Figure 2). With NARS sampling, a user presents a fixed prompt then repeatedly draws individual samples from that fixed prompt. In contrast, with ARS sampling a user presents a prompt then has the LLM autoregressively generate multiple samples. We also identify two additional methodological ingredients that substantially affect the quality of generated distributions in practice. The first is whether the model is *instruction fine-tuned*, trained on additional examples of instruction commands and responses [6]. The second is the number of *prompt examples*.

Second, we present a comprehensive evaluation of LLMs as distribution samplers in two controlled domains: uniform random number sampling and PCFG sampling. By focusing on these controlled domains where we know the expected ground-truth distribution, we are able to evaluate the quality of the samples generated by the LLM.

We propose a suite of analyses that compare sampling methodologies along three primary axes: the *error* of the LLM's induced distribution against the ground truth, the *variance* of the induced distribution across different prompts, and the *containment* of generated samples in the domain of the ground-truth distribution. We also present individual case studies of the distributions induced by different methodological choices.

In general, we find that many LLMs struggle to generate low-error distributions: while the largest models with the right experimental setup have low error in the simplest case (uniform number sampling), all models struggle to beat baselines in harder cases (PCFG sampling). We find high variance in the induced distributions across different choices of prompts. Despite these challenges, we do find high containment of generated samples, indicating that the LLMs are perfectly capable of producing samples within the domain. Regarding the specific methodologies studied in this paper, we find that NARS sampling outperforms ARS sampling, that instruction fine-tuning increases the error of the induced distribution, that larger models generate better distributions than smaller models, and that providing sufficient prompt examples is critical.

Our results demonstrate multiple discrepancies between ground-truth and LLM-induced distributions, emphasizing the need for additional evaluation when introducing LLMs as data generators. The concepts and experiments laid out in this paper work lay the foundations for future work in understanding the capabilities, limitations, and methodologies of distribution sampling using LLMs.

$S ::= NP\ VP \quad (100\%)$	$N ::=$ **cat** (40%)
$NP ::= Det\ N \quad (60\%)$	$N ::=$ **dog** (33%)
$NP ::= N \quad (40\%)$	$N ::=$ **mouse** (20%)
$VP ::= V\ NP \quad (80\%)$	$N ::=$ **book** (10%)
$VP ::= V \quad (20\%)$	$V ::=$ **liked** (50%)
$Det ::=$ **the** (70%)	$V ::=$ **ate** (30%)
$Det ::=$ **a** (30%)	$V ::=$ **read** (20%)

(a) PCFG grammar, showing nonterminals in *italics*, terminals in **bold**, and probabilities in parentheses.

dog liked dog (0.6%)
cat ate a cat (0.3%)
a mouse read (0.1%)
a cat liked dog (0.3%)
a dog liked dog (0.3%)
dog liked cat (0.8%)
cat liked the dog (0.8%)
the cat liked the cat (1.1%)
cat liked cat (1.0%)
a dog ate (0.3%)

(b) PCFG samples (and their associated probabilities of being sampled).

Figure 3: PCFG grammar and samples. Note that not all samples are grammatical English.

2 Experimental Methodology

We evaluate two domains, the *uniform number* domain and the *PCFG* domain. In each domain, we evaluate two sampling methodologies, *non-autoregressive* (NARS) and *autoregressive* (ARS). We evaluate each across a range of models and prompting prompt contexts.

Domains. We evaluate two domains, the uniform number domain and the PCFG domain. In the uniform number domain, we generate samples uniformly from the interval $[0, 1)$ (with two digits; e.g., 0.42 or 0.10). In the PCFG domain, the task is to generate samples from a probabilistic context-free grammar (PCFG), a grammar with associated for each production rule which induces a distribution over sentences in the language. Figure 3 presents the definition of the PCFG, along with some samples and their associated probabilities. This PCFG was generated by querying a language model (ChatGPT) for a simple example of a PCFG.[1]

Sampling methodologies. We evaluate two sampling methodologies in each domain, non-autoregressive (NARS) and autoregressive (ARS). In the NARS methodology, we evaluate the perplexity of each possible sample; assuming a sampling temperature of 0 (and no other changes to the sampling methodology such as nucleus sampling [11] or a frequency penalty), this gives the probability of generating this sample as the first completion after the prompt text (conditioned on generating a sample in the domain).

In the ARS methodology, we allow the model to generate multiple samples autoregressively from a single prompt. We use the model's default temperature settings as a representative example of how the model would be deployed in practice. To ensure that generated samples stay within the expected format, we generate one sample at a time, and manually insert newlines and separators to indicate the next sample as appropriate. In this setting, we run 10 rollouts of 10 generated samples each for each trial; we found that longer rollouts caused significant mode collapse in the smaller models.

Prompting methodologies. In each experiment, we provide a description of the task and a set of examples sampled from the ground-truth distribution. We sweep over the number of ground-truth examples provided, ranging from 0 to 10. See Appendix A for the exact prompts used in each experiment.

Models and hyperparameters. We evaluate the LLaMa model [18] and its derivatives (several of our experiments required fine-grained knowledge of the output logits, which closed-source models like GPT-4 [15] do not provide). Specifically, we evaluate LLaMa-7B, LLaMa-13B, and LLaMa-30B to investigate the effects of model scaling; and Alpaca-7B [17] to investigate the effects of instruction fine-tuning. We evaluate the models using the llama.cpp software project [8] at commit `2e6cd4b` using the llama-cpp-python Python bindings [3] at version 0.1.55. We quantize each model to 8 bits. This software stack includes default sampling parameters (for the ARS experiments) of a temperature of $(\star)$ 0.8, a top-p sampling rate of 0.95, a top-k sampling rate of 40, and a repetition penalty of 1.1.

[1]`https://chat.openai.com/share/d1562920-f38e-48ba-a031-fe2685bbb359`. We use "liked" rather than "chased" in the PCFG to enforce that all words are a single token in our evaluated models.

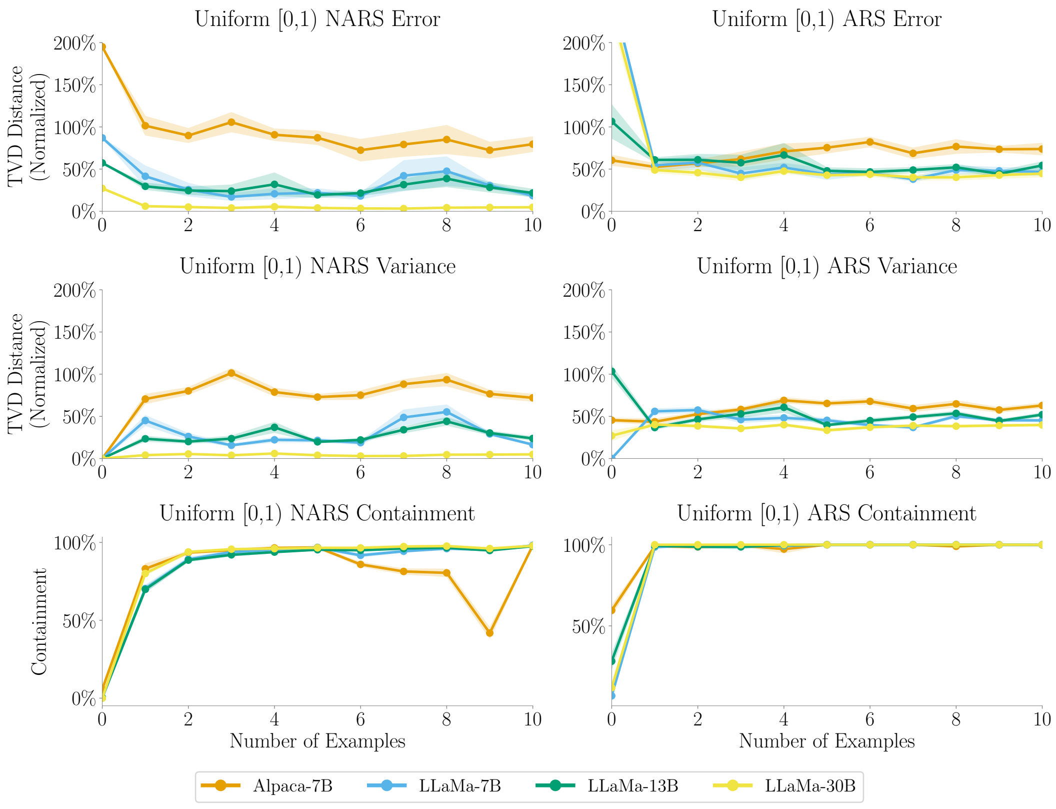


Figure 4: Results for the numbers domain.

Error, variance, and containment metric. For error and variance we use the total variation distance (TVD) metric, which for discrete probability distributions is the L1 distance between probability vectors. We choose this metric as an intuitive distance metric which is well defined for both continuous and discrete distributions.

For NARS experiments, containment is the sum of the probability of each candidate generation (i.e., the probability that a given generation is in the domain). For ARS experiments, containment is the fraction of generated samples that are in the domain.

For each experiment, we run 9 trials with distinct random seeds; all plots show the mean $\pm$ the standard error of the mean of the respective metric over 9 trials.

Baseline errors and variances. We present all results relative to baselines of the error of random distributions (presented as 100% on each plot). That is, for each distribution under study, we sample distributions uniformly at random from the simplex, compute their error and variance, and average across several samples.

3 Results and Analysis

We outline the results for each experiment below, then analyze trends observed across experiments.

3.1 Results

Figure 4 and Figure 5 for the uniform number and PCFG domains respectively show the error (top), variance (middle), and containment (bottom) metrics for NARS sampling (left) and ARS sampling (right). In each plot, the x axis shows the number of examples included from the ground-truth distribution with the prompt. In each plot, the x-axis shows the number of examples included from the ground-truth distribution with the prompt. For the error plots, the y-axis shows the error between the generated distribution

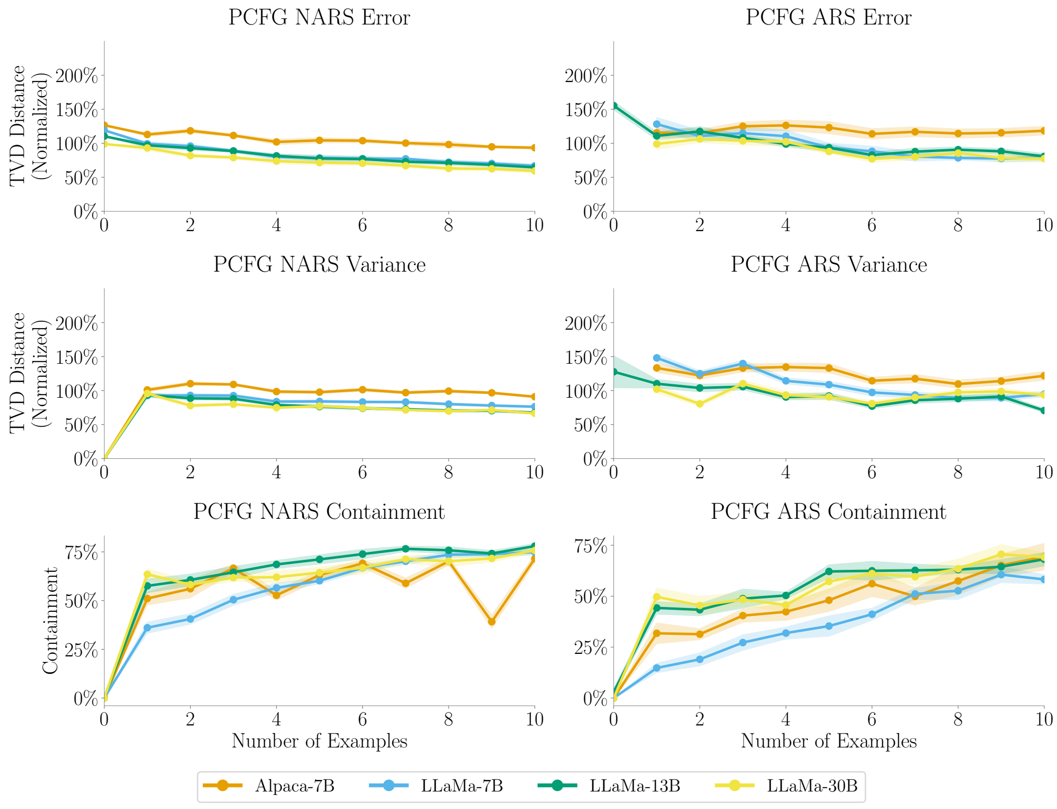


Figure 5: Results for the PCFG domain.

and the ground-truth distribution. For the variance plots, the y-axis show the average error between each different trial of the experimental setting. For the containment plots, the y-axis shows the fraction of generations that are correctly-formatted elements of the domain. For the error plots, the y axis shows the error between the generated distribution and the ground-truth distribution. For the variance plots, the y axis show the average error between each different trial of the experimental setting. For the containment plots, the y axis shows the fraction of generations that are correctly-formatted elements of the domain.

For uniform number experiments, results are discretized to the first digit of the generated number (i.e., 0.0, 0.1, $\ldots$, 0.9). For the PCFG ARS experiments, we do not have sufficient samples to approximate the actual distribution induced by the LLM (there are 468 total classes); instead, the generated distribution is computed by inferring the probabilities of each PCFG rule from the generated samples and using the induced distribution over sentences (note that there is no ARS point at 0 examples, as no models able to generate correctly-formatted sentences).

Figure 6 presents case studies from the uniform number domain, and Figure 7 presents case studies from the PCFG domain.

3.2 Analysis

NARS outperforms ARS sampling. Figures 4 and 5 present the respective results for the uniform number and PCFG domains using the NARS sampling paradigm (left) and ARS sampling paradigm (right). In both the top row–which shows the error between the generated distribution and the ground-truth distribution for each experimental setup–and the middle–showing the error between different trials for each experimental sequence–of each, we see a consistent trend in performance: that NARS sampling outperforms ARS sampling in this experimental setup.

We anecdotally find that ARS sampling easily succumbs to *mode collapse*. Mode collapse occurs when a generator maps different input values to the same output [9]. As with other generation domains, this is an important issue to address in the context of ARS dataset sampling. Figure 7 shows evidence

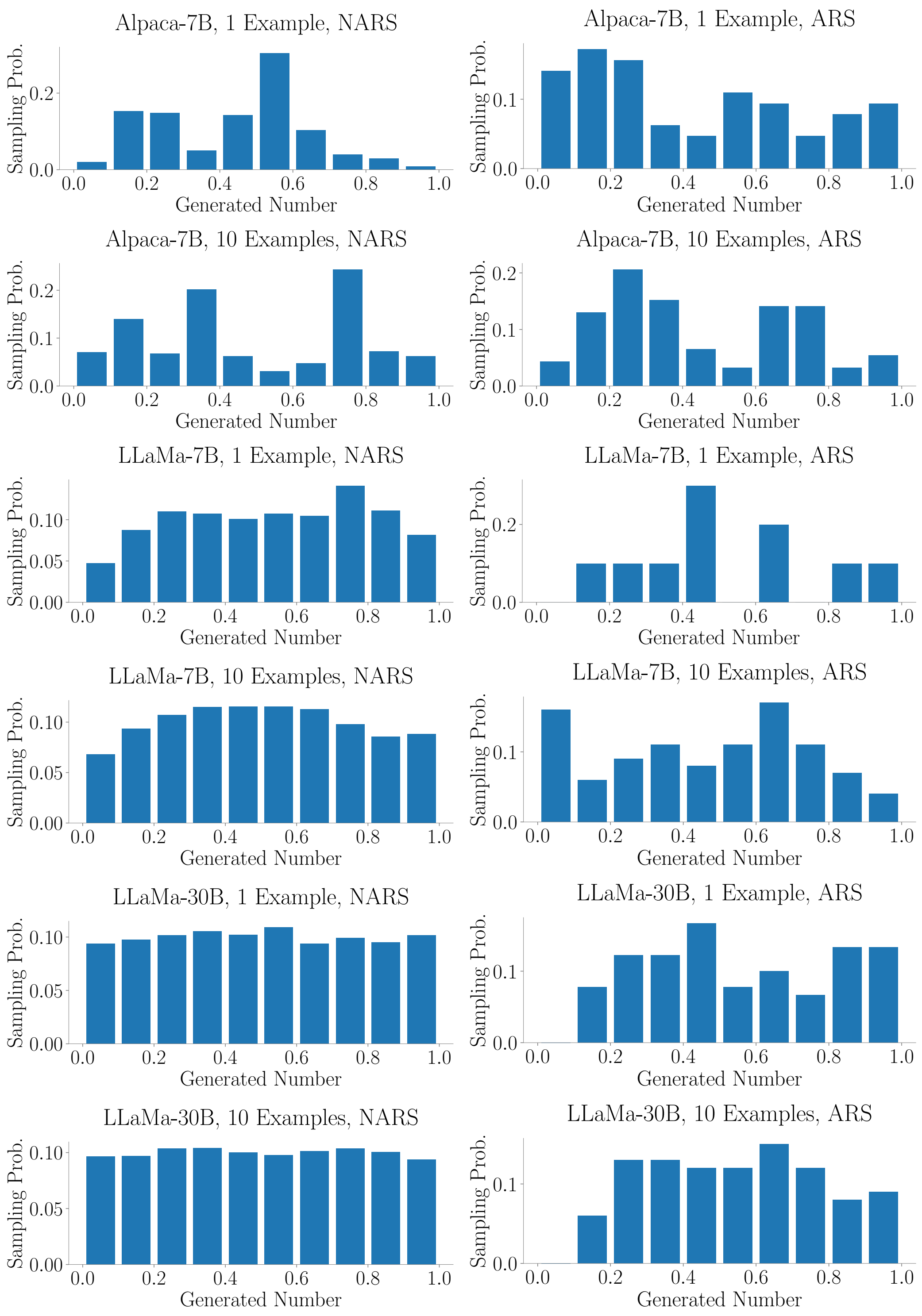


Figure 6: Case studies for the numbers domain, presenting empirical distributions of the median-error trial across a number of experimental settings. Each plot shows a histogram of generated numbers from a single trial, which is chosen as the trial with the median TVD error in that configuration.

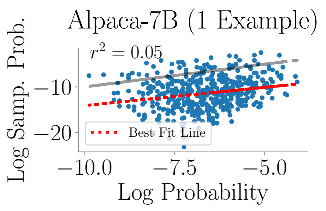
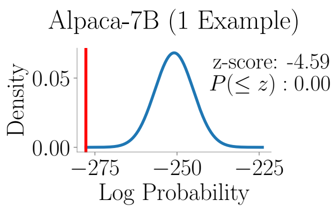
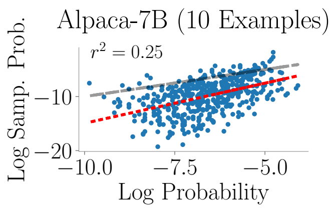
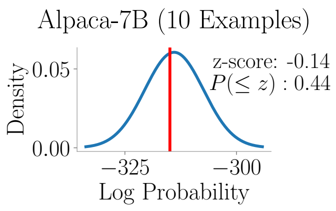
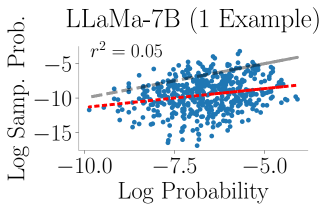
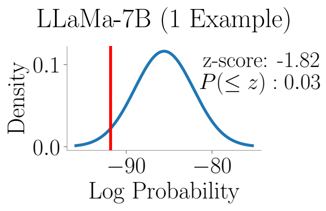
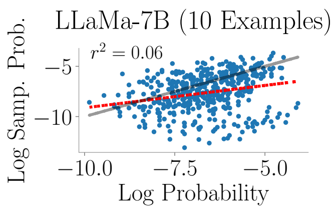
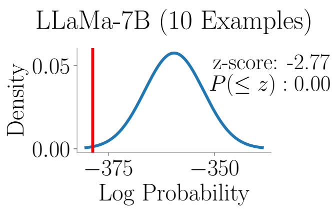
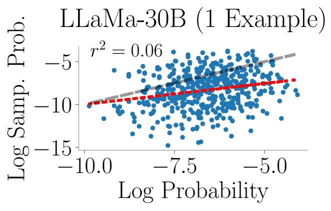
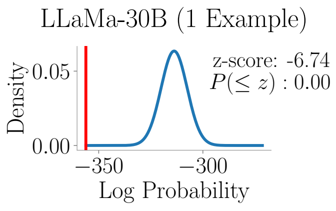
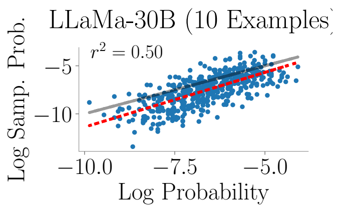
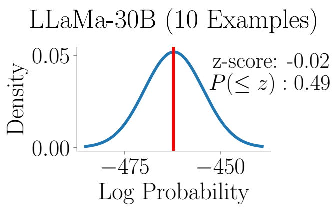
NARS Correlations	ARS Examples	ARS Probabilities
	cat ate book (0.15%), mouse cat (N/A), book read dog (0.08%), cat liked cat (1.02%), book read book (0.03%), dog ate cat (0.46%), cat liked mouse (0.51%), book read cat (0.10%), mouse read mouse (0.10%), dog ate book (0.12%)	
	the cat liked the mouse (0.56%), the mouse liked the cat (0.56%), the book liked the cat (0.28%), the book liked the mouse (0.14%), the book liked the cat and the mouse (N/A), the dog liked the cat and the mouse (N/A), the cat liked the cat and the mouse (N/A), the mouse liked the cat and the mouse (N/A), ate the cat (N/A), liked the cat (N/A)	
	cat ate book (0.15%), the book mouse likes and read (N/A), the cat liked the dogs (N/A), the cats ate the book (N/A), the book ate and read the mouse's (N/A), the mouse ate the cat's dog (N/A), the cat the dogs ate (N/A), the book ate the dog (0.13%), a the cat like and eat the (N/A), the cat liked the dog (0.85%)	
	the dog ate dog (0.36%), the cat eating a book (N/A), the dog liking the dog (N/A), the mouse eating a book (N/A), the cat liked the book (0.28%), dog read (the) book (N/A), the dog liked dog (0.60%), the book liked book (0.07%), dog ate cat (0.46%), book ate dog (0.12%)	
	a book read the dog (0.04%), a mouse ate the mouse (0.07%), the dog ate the dog (0.38%), the cat read the cat (0.45%), a book liked the cat (0.12%), the dog liked the mouse (0.42%), a mouse read the book (0.02%), the mouse liked the cat (0.56%), the cat read the mouse (0.23%), a cat read the mouse (0.10%)	
	the cat liked mouse (0.54%), the cat liked dog (0.81%), the dog ate the cat (0.51%), the dog ate a dog (0.16%), the cat read a book (0.05%), the dog read the cat (0.34%), the cat read a cat (0.19%), the cat read the dog (0.34%), the dog read the cat (0.34%), the cat liked the mouse (0.56%)	

Figure 7: Case studies for the PCFG domain. Each row shows results from the median error trial in that configuration. The left column shows the correlations between ground-truth frequency (x-axis) and generation probability (y-axis); each point represents a sentence in the grammar. Each plot includes Pearson correlation (r^2), line of best fit (in red), and the ideal $y = x$ (in light grey). The middle column shows example sentences generated from ARS generation, along with the ground-truth frequency of those sentences (N/A if not in language). The right column shows the joint probability of generated sentences compared to the distribution of expected probabilities. ARS generation produces infrequent sentences more often than expected.

of mode collapse for Alpaca-7B: in the 1-example context, all generated sentences take the same form (noun, verb, noun); in the 10-example context, most examples past the fifth use a conjunction ("and") which is not in the PCFG.

However, we do not claim that NARS sampling categorically outperforms ARS sampling. ARS sampling requires tuning many more sampling hyperparameters, and requires other design choices; it is possible that ARS sampling could outperform NARS sampling with the right hyperparameters and design choices. For instance, ARS sampling may help recover good quality sampling for a model that is poorly calibrated for NARS sampling.

More prompt examples help. In most contexts, including more examples in the prompt results in lower error and variance. This is illustrated in multiple cases: in the uniform number domain, Figure 6 shows that just how much improved distributions produced with 10 examples are to those with only 1; in Figure 7, even with the 30B parameters, LLaMa-30B cannot produce a reasonable distribution with only one sample; in the ARS paradigm, we see variance, error and containment somewhat converge across model architectures when at least more than 6 samples are presented within the prompt. There is a caveat to this level of detail, however; including these additional examples increases the length of the prompt, resulting in a more expensive inference.

The primary exception to this is with the NARS Alpaca-7B model at 9 prompt examples, at which containment consistently decreases. We hypothesize that this is because the instructions that Alpaca-7B is fine-tuned on include round numbered lists of items (e.g., 10), causing a discontinuity in behavior at this point (in which Alpaca-7B is being prompted to complete the tenth example).

The choices of prompts matter. Each trial uses a different random seed to generate examples included in the prompt, inducing different distributions from the LLM. The variance exhibited in both the uniform number and PCFG domains show these different choices of examples in the prompt result in significantly different induced distributions.

Language models struggle to generate low-error distributions The top rows of Figures 4 and 5 show the error of each generated distribution against the ground-truth expected distribution. The only instance of low-error ($< 10\%$) generation is the NARS LLaMa-30B in the uniform number domain with at least 1 prompt example. In all other experimental contexts, all models fail to accurately model the ground-truth distribution. We expect these struggles to be exacerbated in contexts where the model or its user do not have a firm understanding of the distribution that is being sampled from.

Modeling decisions impact performance The containment and error plots in each of Figures 4 and 5 show that that instruction fine-tuning improves output quality but hurts calibration. In all domains, instruction fine-tuning (Alpaca) results in higher containment (i.e., generating more in-domain samples) but has worse error and variance. This property has been observed in other domains: for example, OpenAI [15, Figure 8] show that instruction fine-tuning of GPT-4 hurts calibration on a multiple choice exam dataset; our findings confirm that this affects the quality of induced data distributions.

Relatedly, the size of model impacts both error, precision and containment metrics. In all contexts, larger LLaMa models have equivalent or better performance than smaller LLaMa models.

4 Discussion

Throughout our evaluation, there has remained one critical high-level takeaway: LLMs do not always generate the prompted distribution. These results are in particular sensitive to the expected distribution itself, the sampling methodology, and the choice of the model architecture and dataset. As a result, before drawing multiple samples from an LLM practitioners should ask, "What does it mean to draw a sample from my LLM? What is the distribution I expect? How will I evaluate the resulting outputs?" While we currently lack systematic ways to express or evaluate these questions, this work acts as a first step towards reducing this ambiguity. For example, as we have shown, practitioners can probe these questions by evaluating the perplexity of example generations that they would expect to be in-distribution.

There are many existing limitations to LLMs that we do not directly evaluate: tokenization, biases in training data, and mode collapse all offer novel avenues for future research to explore within the context of LLMs as distribution samplers. Further, memorization and cloning within LLMs remains a deep concern for users [7]. Janus [12] demonstrated that some LLMs have a favorite number: 42 (a popular reference to Douglas Adam's Hitchhiker's Guide to the Galaxy series). A sampled distribution should likely not contain a high count of repeated values, nor should those values be regurgitated from an uncited source. LLMs produce hallucinations framed as reasonable, real-world facts [13]. Despite work suggesting ways to reconcile this misinformation [2], such approaches are far from entering the mainstream. These missteps break fundamental user expectations (particularly in discrete applications) and may thus harm the sampled distribution's quality.

Future work should explore these in detail, contributing benchmark tasks, datasets, and baselines to calibrate LLM-produced distributions against.

References

[1] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models, 2021.

[2] Amos Azaria and Tom Mitchell. The internal state of an llm knows when its lying, 2023.

[3] Andrei Betlen et al. llama-cpp-python. `https://github.com/abetlen/llama-cpp-python`, 2023.

[4] Luca Beurer-Kellner, Marc Fischer, and Martin Vechev. Prompting is programming: A query language for large language models. *PLDI '23*, 2022.

[5] Vadim Borisov, Kathrin Seßler, Tobias Leemann, Martin Pawelczyk, and Gjergji Kasneci. Language models are realistic tabular data generators, 2023.

[6] Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Alex Castro-Ros, Marie Pellat, Kevin Robinson, Dasha Valter, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Zhao, Yanping Huang, Andrew Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. Scaling instruction-finetuned language models, 2022.

[7] Matteo Ciniselli, Luca Pascarella, and Gabriele Bavota. To what extent do deep learning-based code recommenders generate predictions by cloning code from the training set? In *Proceedings of the 19th International Conference on Mining Software Repositories*, MSR '22, page 167–178, New York, NY, USA, 2022. Association for Computing Machinery.

[8] Georgi Gerganov et al. llama.cpp. `https://github.com/ggerganov/llama.cpp`, 2023.

[9] Ian J. Goodfellow. NIPS 2016 tutorial: Generative adversarial networks. *CoRR*, abs/1701.00160, 2017.

[10] Perttu Hämäläinen, Mikke Tavast, and Anton Kunnari. Evaluating large language models in generating synthetic hci research data: A case study. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, CHI '23, New York, NY, USA, 2023. Association for Computing Machinery.

[11] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. In *International Conference on Learning Representations*, 2020.

[12] Janus. Mysteries of mode collapse. `https://www.alignmentforum.org/posts/t9svvNPNmFf5Qa3TA/mysteries-of-mode-collapse`, 2022.

[13] Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. Survey of hallucination in natural language generation. *ACM Computing Surveys*, 55(12):1–38, 2023.

[14] Sergey I Nikolenko. *Synthetic data for deep learning*, volume 174. Springer, 2021.

[15] OpenAI. Gpt-4 technical report, 2023.

[16] Max Schäfer, Sarah Nadi, Aryaz Eghbali, and Frank Tip. Adaptive test generation using a large language model, 2023.

[17] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford alpaca: An instruction-following llama model. `https://github.com/tatsu-lab/stanford_alpaca`, 2023.

[18] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023.

[19] Chenxi Whitehouse, Monojit Choudhury, and Alham Fikri Aji. Llm-powered data augmentation for enhanced crosslingual performance, 2023.

[20] Zhiqiang Yuan, Yiling Lou, Mingwei Liu, Shiji Ding, Kaixin Wang, Yixuan Chen, and Xin Peng. No more manual tests? evaluating and improving chatgpt for unit test generation, 2023.

A Prompts

This section contains the prompts used for each domain. In all domains we vary the number of examples given depending on the experimental context.

Uniform numbers. The uniform number prompt is as follows:

```
The following is a list of uniform random numbers in the interval [0, 1]:

1. 0.16
2.
```

PCFG. The PCFG prompt is as follows:

```
The following is a list of samples from the following PCFG (note that they are
not necessarily grammatical English):

‘‘‘
S -> NP VP [1.0]
NP -> Det N [0.6] | N [0.4]
VP -> V NP [0.8] | V [0.2]
Det -> "the" [0.7] | "a" [0.3]
N -> "cat" [0.4] | "dog" [0.3] | "mouse" [0.2] | "book" [0.1]
V -> "liked" [0.5] | "ate" [0.3] | "read" [0.2]
‘‘‘

1. cat liked the dog
2.
```

Normal numbers. The normal number prompt is as follows:

```
The following is a list of uniform random numbers in the interval [0, 1]:

1. 0.16
2.
```

B Bit Sampling Experiments

Figures 8 to 10 present results for a bit sampling domain. In this domain, the objective is to sample individual bits (0 or 1) according to a range of distributions. Figure 8 presents the uniform bits domain, in which the objective is to sample bits from the uniform distribution over bits. Figure 8 presents the nonuniform bits domain, in which the objective is to sample bits from a nonuniform distribution over bits where 0 is sampled with probability 75% and 1 is sampled with probability 25%. Figure 8 presents the nonuniform bits with bad prompting domain, in which the objective is again to sample bits from a nonuniform distribution over bits where 0 is sampled with probability 75% and 1 is sampled with probability 25%; however in this domain, the prompt examples are drawn from a distribution where 0 is sampled with probability 25% and 1 is sampled with probability 75%.

We find similar results to the domains evaluated in the main body of the paper. Despite the uniform bit domain being conceptually simple, most models struggle to generate uniform distributions over $\{0, 1\}$. The nonuniform bits domains have even higher error. However, the nonuniform bits with bad prompting domain only marginally increases the error compared to the regular nonuniform bits domain, suggesting that the models are not learning the distribution from the prompt examples.

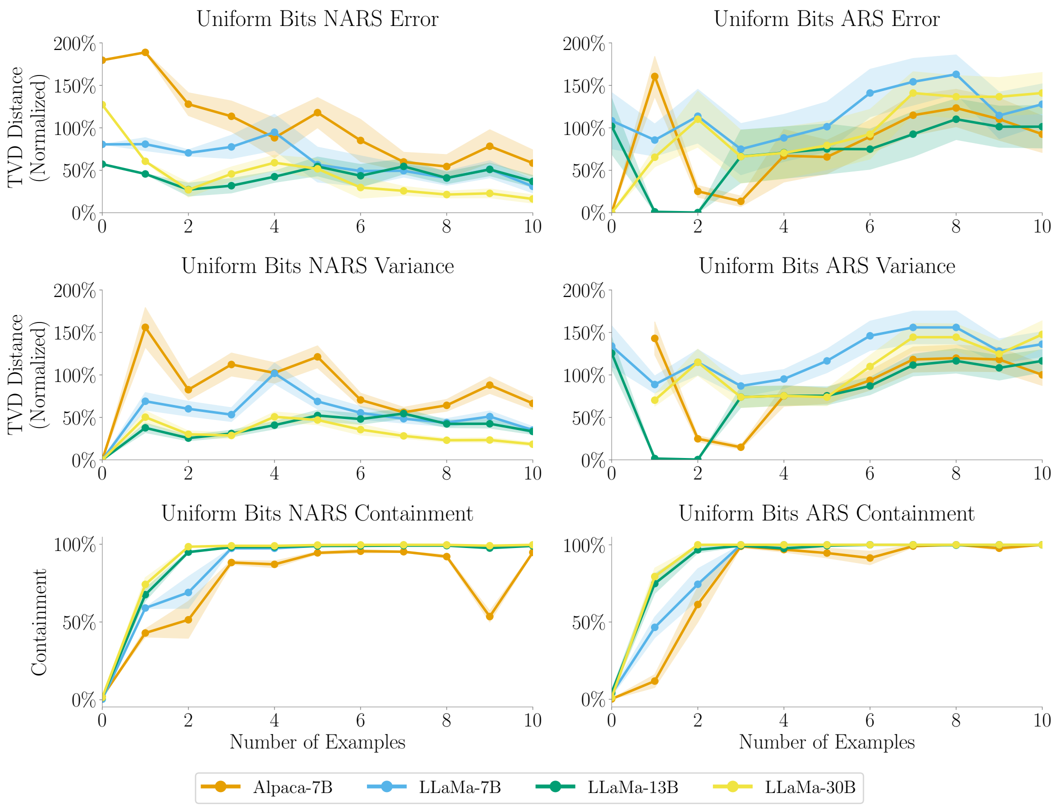


Figure 8: Results for the uniform bits domain.

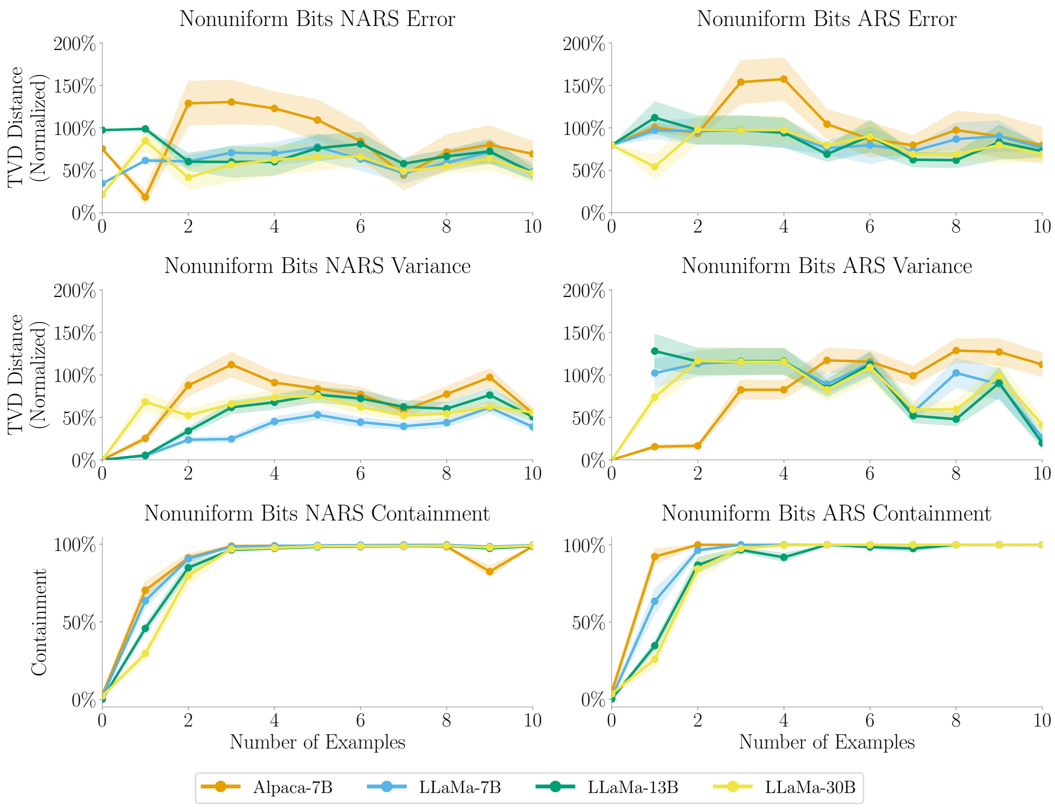


Figure 9: Results for the nonuniform bits domain.

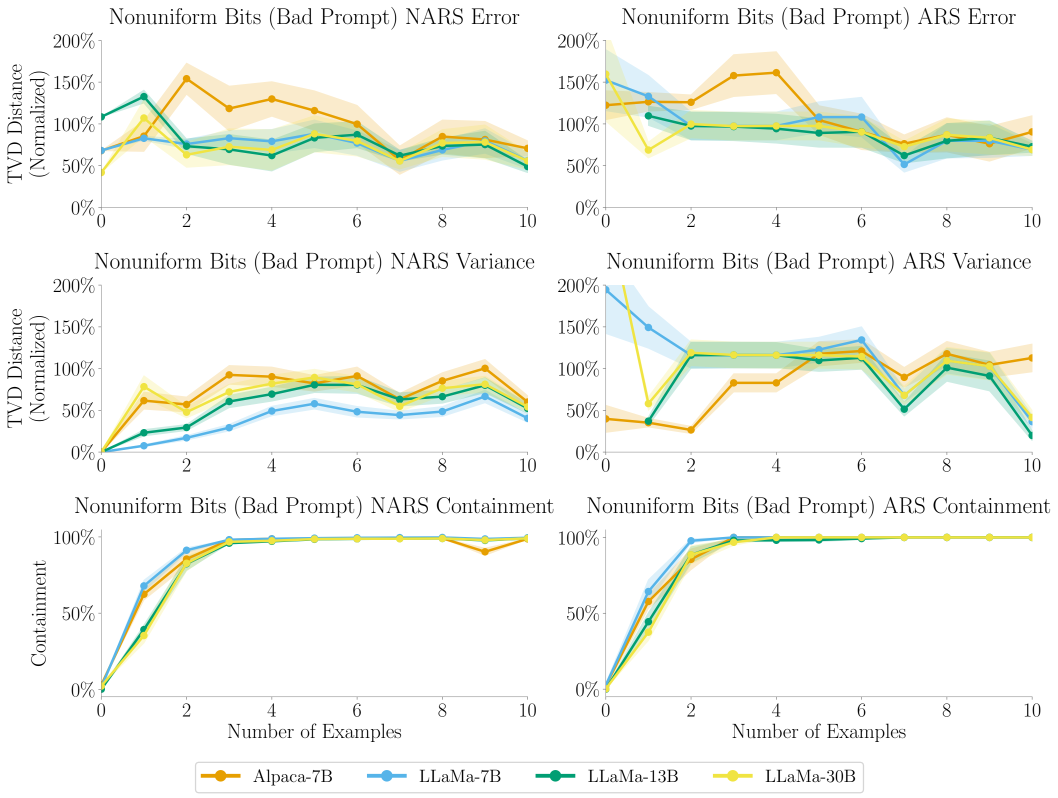


Figure 10: Results for the nonuniform bits with bad prompting domain.

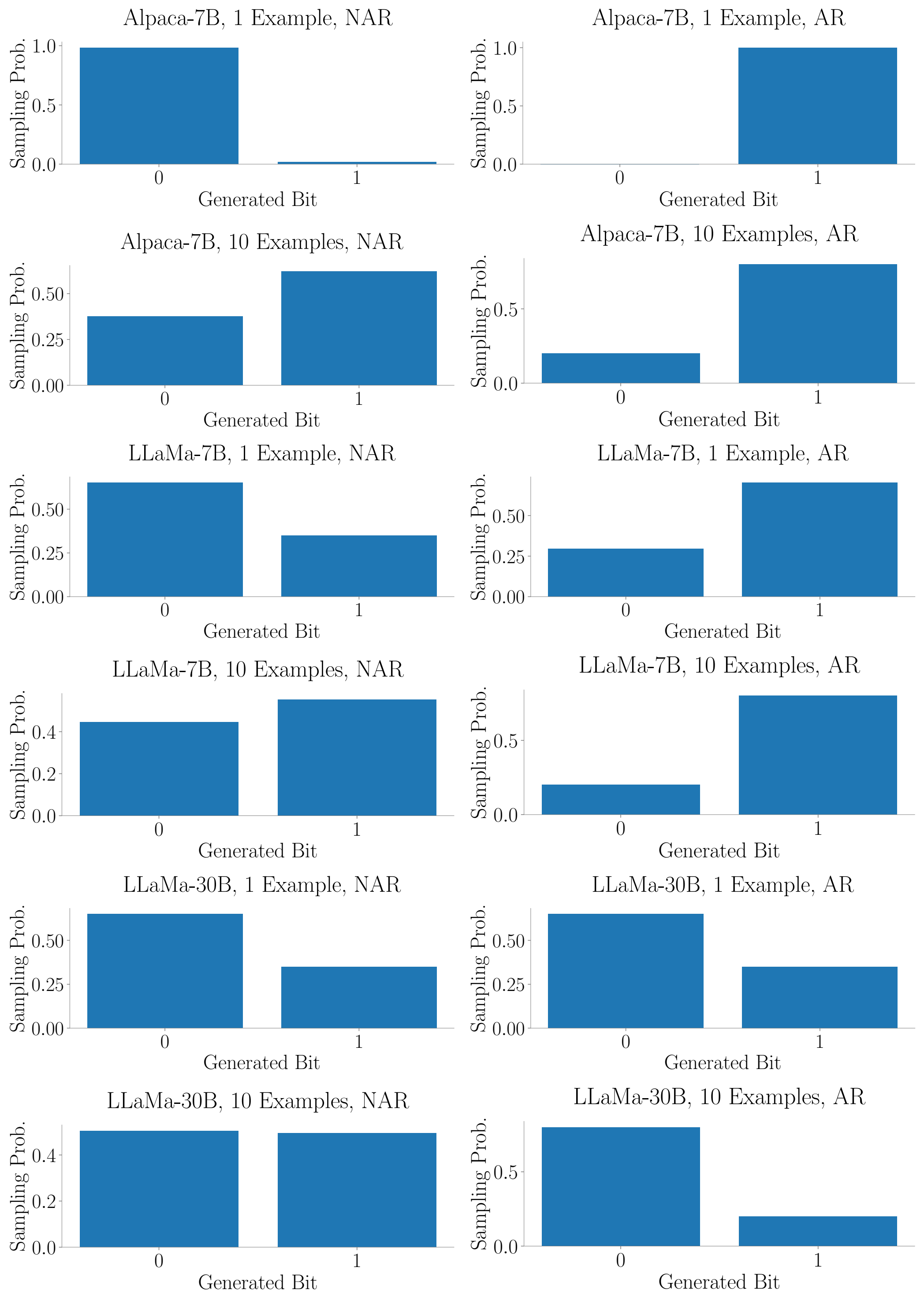


Figure 11: Case studies for the uniform bits domain, presenting empirical distributions of the median-error trial across a number of experimental settings. Each plot shows a histogram of generated bits from a single trial, which is chosen as the trial with the median TVD error in that configuration.

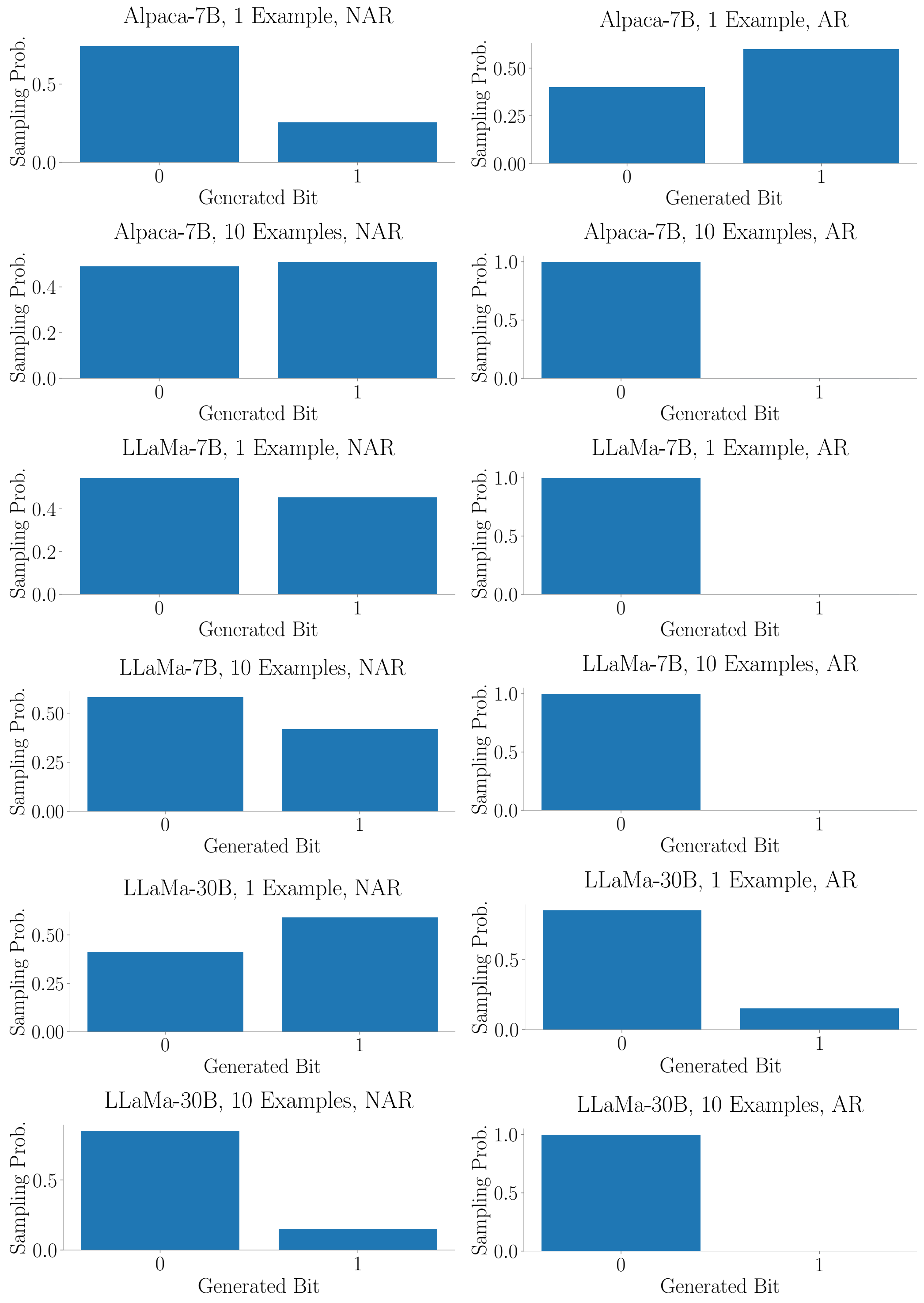


Figure 12: Case studies for the nonuniform bits domain, presenting empirical distributions of the median-error trial across a number of experimental settings. Each plot shows a histogram of generated bits from a single trial, which is chosen as the trial with the median TVD error in that configuration.

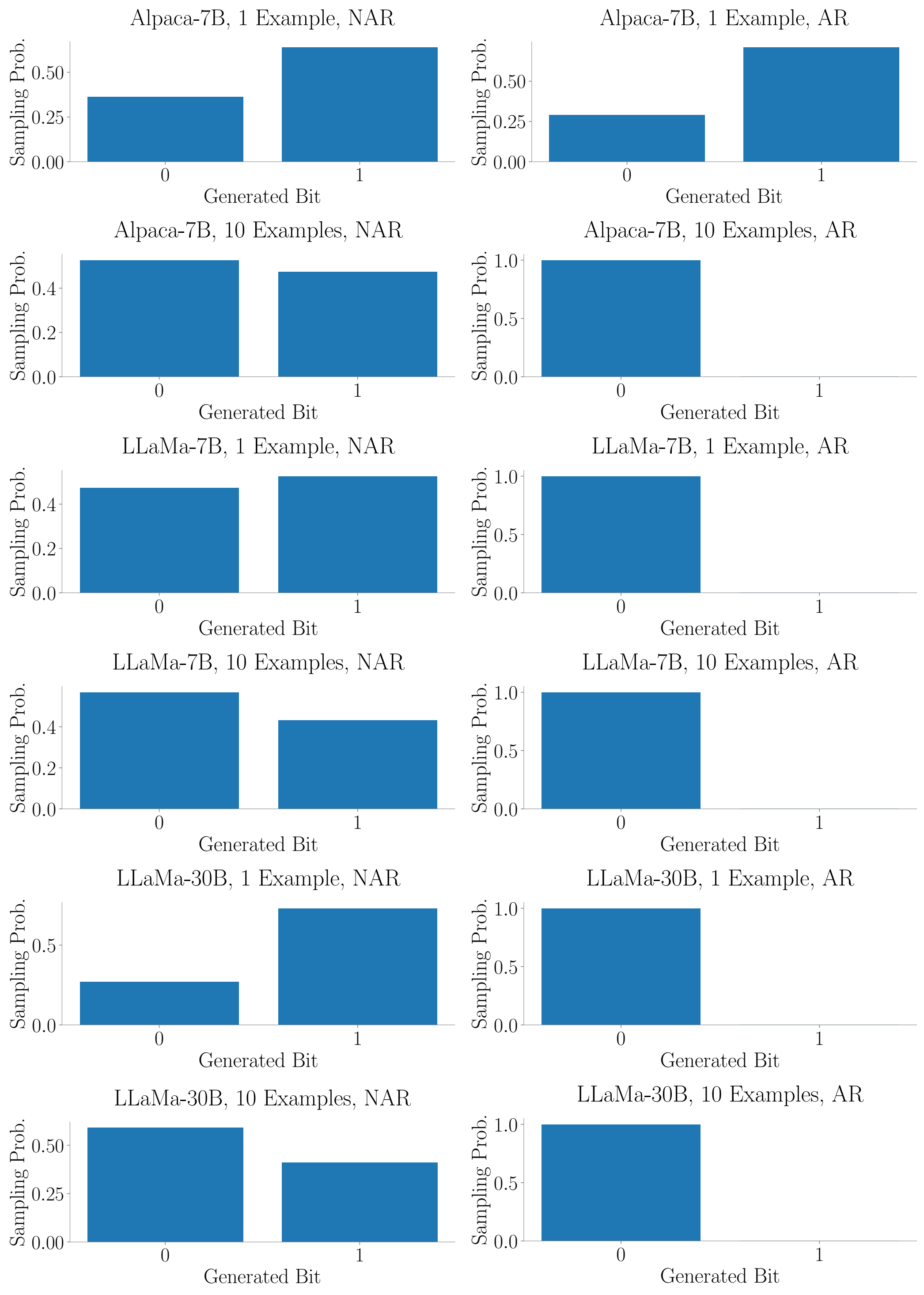


Figure 13: Case studies for the nonuniform bits with bad prompting domain, presenting empirical distributions of the median-error trial across a number of experimental settings. Each plot shows a histogram of generated bits from a single trial, which is chosen as the trial with the median TVD error in that configuration.

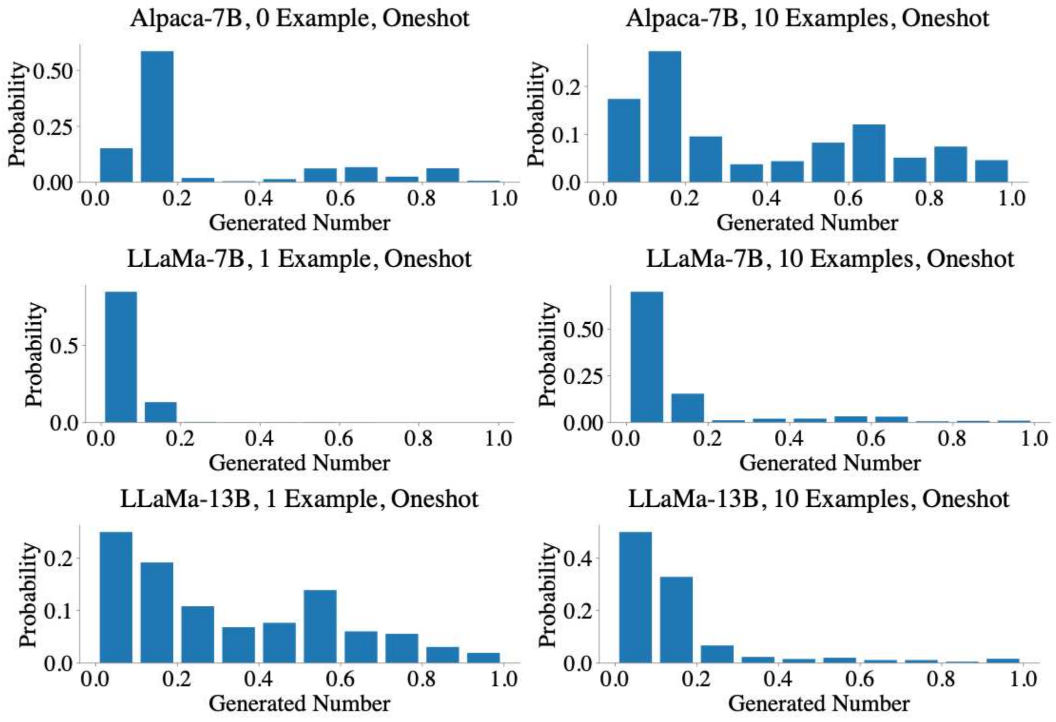


Figure 14: Normal distribution results