entro-p: An Entropy-Based Logit Processor for Improved Pass@k Reasoning

Anonymous ACL submission

Abstract

Interest in creating language models (LMs) capable of solving challenging math and coding questions has given rise to a host of new sampling strategies. One of the simplest approaches for questions that are difficult to solve but easy to verify is to sample repeatedly until the LM derives a correct solution. Due to the high cost of these sampling methods, even small gains in reasoning efficiency matter. Drawing inspiration from recent work that developed logit processors that improve creative outputs from LMs, we develop the first logit processor, entro-p, that specifically targets improved performance on pass@k reasoning tasks for LMs. Designing a logit processor for pass@k reasoning tasks is challenging because for small k, the optimal strategy is close to greedy sampling, but for large k, one might sample from the unprocessed logits to maximize the range of solutions in the search space. Our processor, entro-p, finds a happy medium between these two extremes that is able to improve both high and low-k performance relative to existing logit processors. Using entro-p, we are able to achieve performance gains on pass@100 performance of up to 2% on the MATH, AIME24, and AIME25 benchmarks, and up to 3.1% on the MBPP benchmark. These efficiency gains, which come at little extra cost during inference time, demonstrate that improvements in reasoning efficiency do not always require additional training resources. Moreover, they broaden understanding of how targeted logit processing can improve task performance beyond creative content generation.

1 Introduction

Currently, there are many reasoning, math, and coding questions that language models (LMs) cannot solve consistently in a single attempt (Brown et al., 2024). To remedy this problem, some researchers have explored increasing inference-time compute to improve models' capacity to excel at reasoning tasks. There are many reasoning tasks that are difficult to solve, but easy to verify (Kulal et al., 2019; Chen et al., 2021). For such tasks Brown et al. (2024) suggested a naive but effective solution: one can prompt LMs thousands of times in parallel to improve their probability of solving difficult problems at least once. Language models prompted using this method are known as "Large Language Monkeys."

In the Large Language Monkeys setting, practitioners set a fixed per-question compute budget that could allow for up to 10,000 independent samples. They then repeatedly ask a language model the same question. Performance on the ith question is measured by $\text{pass}_i@k$, given by

$$\text{pass}_i@k \stackrel{\text{def}}{=} \mathop{\mathbb{E}}_{k \text{ Attempts}} \Big[\mathbb{I}[\text{Any attempt on } i\text{-th problem succeeds}]\Big]. \tag{1}$$

This per-question success rate is averaged over the entire dataset, so that pass@k on the full dataset $\mathcal{D}$ containing P problems is defined as

$$\text{pass}_{\mathcal{D}}@k = \frac{1}{P}\sum_{i=1}^{P} \text{pass}_i@k.$$

To estimate $\text{pass}_i@k$, we used the unbiased and numerically stable estimator from Chen et al. (2021) and followed the procedure from Anonymous (2025): for the i-th problem, sample $n \gg k$ attempts per problem, count the number of successes c, and then compute an estimate of $\widehat{\text{pass}_\text{i}@\text{k}}$ for different k values using the expression:

$$\widehat{\text{pass}_\text{i}@\text{k}} = 1 - \frac{\binom{n-c}{k}}{\binom{n}{k}} \tag{2}$$

Independently sampling to solve hard problems is hugely expensive; any improvement in efficiency can save large amounts of compute. Inspired by

recent improvements in creativity garnered by improved logit processors (Nguyen et al., 2024), we ask: is there a logit processor that can improve pass@k reasoning performance?

In this paper, we demonstrate that the choice of logit processor has a significant impact on pass@k success rates: appropriate logit processors can improve pass@1 accuracy by over 10% relative to "base" sampling (i.e. sampling using un-truncated logits) on common math and coding benchmarks. We discover a surprising paradox: a logit processor that performs well using a pass@1 metric often performs poorly using a pass@k metric for large values of k. This is illustrated theoretically in Section 5.2. Ideally, one would like a logit processor that performs well on both pass@1 and pass@k metrics. Towards this goal, we propose the *entro-p* logit processor in Section 3 as a dynamic variant of min-p. We compare entro-p to other token sampling methods including min-p (Nguyen et al., 2024), top-p (Holtzman et al., 2020), and base sampling in Section 4. The comparison is made across the benchmark datasets MATH (Hendrycks et al., 2021), MBPP (Austin et al., 2021), and AlpacaEval Creative Writing (Li et al., 2023). These comparisons show that entro-p holds a slight edge over other token sampling methods using pass@k metrics for both large and small values of k, demonstrating that it improves factuality overall, while contributing little additional latency to generation.

To summarize our contributions:

1. We show that the choice of logit processor has a strong impact on model performance on reasoning tasks: a good processor can garner performance gains up to 10% over base sampling for both pass@1 and pass@k, $k \approx 500$.

2. We highlight an interesting paradox: strong performance under a pass@1 metric often translates into poor performance under a pass@k metric for large k.

3. We introduce entro-p, a logit processor that is able to perform well under pass@k for both large and small k on reasoning tasks. In addition to showing good performance on reasoning benchmarks, entro-p achieves SOTA performance on Alpaca Creative Eval (Li et al., 2023), showing that reasoning performance does not need to compromise creativity.

2 Background

To better contextualize this paper, we explain the notion of truncation sampling, which gave rise to several modern logit processors. Language models are often miscalibrated on low-probability tokens, assigning non-zero probabilities to tokens that do not make sense. Hewitt et al. (2022) showed that pure sampling is deficient because language models typically predict a non-zero probability for *all* possible tokens at each generation step, which is a byproduct of using cross-entropy loss during pretraining and not a feature of the empirical distribution of language (Meister and Cotterell, 2021). Indeed, Holtzman et al. (2020) showed that pure sampling (sampling directly from a language model's predicted probability distribution) can lead to incoherent text. To address this problem, practitioners introduced truncation sampling, which removes low-probability tokens and recalculates the sampling distribution over the tokens that are left.

Before describing past methods of truncation sampling, or *logit processing*, we establish some common notation. Let $\mathcal{V}$ denote the vocabulary of a language model, and let $\mathbb{P}_\theta(x_t|x_{1:t-1})$ represent the conditional probability distribution for the next token x_t. Here $x_{1:t-1}$ denotes the sequence of the first $t-1$ generated tokens. In practice, next-token probabilities are output as logits by the language model. Formally, if $|\mathcal{V}| = n$, then

$$x_t \in \{v_1, ..., v_n\}$$

where the language model assigns a *logit* score ℓ_i to v_i. One can recover the probability distribution over $(v_1, ..., v_n)$ via the softmax function:

$$\mathbb{P}_\theta(v_i|x_{1:t-1}) = \frac{e^{\ell_i}}{\sum_{j=1}^{n} e^{\ell_j}}.$$

During truncation sampling, one selects a subset $S \subset \mathcal{V}$, resetting the probability

$$\tilde{\mathbb{P}}_\theta(v_i|x_{1:t-1}) = \begin{cases} \frac{e^{\ell_i}}{\sum_{j:v_j \in S} e^{\ell_j}} & v_i \in S \\ 0 & \text{else.} \end{cases}$$

We denote the reverse order statistics of the logits by $\ell_{(1)} > \ell_{(2)} > \cdots > \ell_{(n)}$, and we use the same convention for the corresponding probabilities and the vocabulary: $v_{(1)}$ is the token with the highest predicted probability, which is $p_{(1)}$.

Having established this notation, we can now describe the leading existing logit processors.

2.1 Top-p

Holtzman et al. (2020) introduced the first truncation method: *top-p* sampling. Top-p sets

$$c = \min \left\{ i : \sum_{j=1}^{i} p_{(i)} \geq p \right\}.$$

Then, it selects the set $S \subset \mathcal{V}$ to be

$$S = \{v_{(1)}, ..., v_{(c)}\}.$$

In words, top-p selects the most probable tokens that contain at least p of the total probability mass in the sampling distribution for the next token.

2.2 Top-k

Top-k sampling selects $S = \{v_{(1)}, ..., v_{(k)}\}$. Typically, k is chosen betwen 15 and 100 tokens. Setting $k = 1$, one recovers greedy decoding. Although simple, this method helped Fan et al. (2018a) achieve significant gains for neural story generation in 2018.

2.3 Min-p

Recently, Nguyen et al. (2024) discovered that setting a hard cutoff, as top-p and top-k do, can inhibit creativity. They achieved increases in creativity through the min-p logit processor, which sets

$$S = \{v_i : p_i \geq p_{(1)} \cdot p\}.$$

In other words, min-p considers all tokens that have a probability at least p times the probability of the most-likely next token. Although simple, this method significantly improves creativity as measured by both human evaluations and Alpaca (Li et al., 2023).

With these past logit processing methods for context, we now proceed to describe our new logit processor for reasoning tasks: entro-p.

3 entro-p

3.1 Motivation

Nguyen et al. (2024) noted that using an adaptive threshold can improve creativity in LMs. However, the threshold set by Nguyen et al. (2024) depends only on the most probable token; thus, it does not account for other information in the predicted probability distribution. Intuitively, truncation sampling should filter logits more aggressively when the LM exhibits higher certainty about the "correct" next token, and less aggressively when there is more uncertainty. We measure the LM's uncertainty using entropy, allowing us to produce a more flexible cutoff that yields better results than min-p. Since the calculation of the cutoff in entro-p involves the whole probability distribution (via the entropy), entro-p has better sensitivity to the whole probability distribution in comparison to min-p.

A second motivation for entro-p comes from our observations of pass@k performance across different samplers. As we demonstrate in Section 5, superior performance in pass@1 sampling typically translates into degraded performance in pass@k sampling. Through entro-p, we aim to build a logit processor that can perform well on both pass@1 and pass@k, catering to multiple different compute budgets.

3.2 Method Description

In this section, we describe entro-p sampling, which uses entropy-based adaptive thresholds to improve upon min-p sampling. Before proceeding to the method, we briefly review the definition of entropy, which is central to our method. Recall that for a discrete probability distribution that takes values $x_1, ..., x_n$ with probabilities $p_1, ..., p_n$, the entropy is defined as

$$h = -\sum_{i=1}^{n} p_i \log(p_i).$$

Entropy is maximized by a uniform probability distribution, and minimized by a distribution that puts all probability mass on a single point. Entro-p truncates aggressively when measured entropy is low, and less aggressively when measured entropy is high. Recall that in our notation from Section 2, an LM selects $x_t \in \mathcal{V}$ from the distribution given by

$$x_t \sim \mathbb{P}_\theta(x_t | x_{1:t-1}).$$

The reverse order statistics of the logits are denoted by $\ell_{(1)} > ... > \ell_{(n)}$. In our notation, a logit processor selects a subset $S \subset \mathcal{V}$ of tokens to consider during sampling.

Let U and L be hyperparameters used to clamp the measured entropy, and d be an additional hyperparameter used in Step 5. When selecting the next token, the entro-p logit processor proceeds in the following steps:

1. Find the maximum logit $\ell_{(1)}$.

2. Reduce the tokens under consideration from $\mathcal{V}$ to $\tilde{S} = \{v_i : p_i \geq p_{(1)} \cdot e^{-7}\}$ and let $\tilde{P}_\theta(x_t|x_{1:t-1})$ denote the resulting (renormalized) distribution.

3. Recalculate $\tilde{p}_i = \frac{\exp(\ell_i)}{\sum_{v_i \in \tilde{S}} \exp(\ell_i)}$.

4. Compute the entropy $h = h_t$ of $\tilde{P}_\theta(x_t|x_{1:t-1})$:

$$h_t = -\sum_{v_i \in \tilde{S}} \tilde{p}_i \log \tilde{p}_i.$$

5. Refine the set $\tilde{S}$ further by setting $S = \{v_i \in \tilde{S} : \tilde{p}_i \geq (\tilde{p}_{(1)} - (d + h))\}$.

6. Replace h by the clamped value $h^* = \max(\min(h_i, U), L)$.

7. Scale the logits corresponding to tokens in the set S by $\ell_i^{\text{scale}} = \ell_i / h^*$ for $v_i \in S$. Finally, sample the next token using the logits ℓ_i^{scale}.

The number of tokens in the vocabulary is fixed. Since the only operations needed to implement entro-p are sorting and calculation of entropy, entro-p has a run-time of $O(|\mathcal{V}| \log(|\mathcal{V}|))$, which is insignificant compared to the forward method of a modern LM for $|\mathcal{V}| \approx 128{,}000$.

3.3 entro-p Case Study

We now provide a case study illustrating how entro-p can reduce the occurrence of errors in contexts requiring technical reasoning. Recall that in such a context, the logit distributions are very often highly concentrated on only a few tokens. It is thus generally beneficial to truncate more conservatively, such as by using a lower p value in top-p sampling, or by employing a lower temperature for scaling. These methods are relatively static, and in some cases, insufficient.

In Figure 1, we show an interaction in which the language model initially makes a mistake in the penultimate step of its evaluation of a mathematical expression. Towards the end of its second response, the model is tempted again to make the same mistake, as can be seen in the relatively high logit score for the token '3.' Entro-p utilizes the entropy of the distribution at this step of generation to determine the truncation level, indicated by the dotted blue line, discarding the incorrect token. On the other hand, we see that even under a restrictive choice of p, the min-p processor fails to filter out this token.

User: Evaluate $i^5 + i^{-25} + i^{45}$.
Assistant: To evaluate $i^5 + i^{-25} + i^{45}$, we use the fact that $i^4 = 1 \ldots, i^{-25} = \ldots = -i$.
$\ldots$ Therefore, $i^5 + i^{-25} + i^{45} = i - (-i) + i = 3i$,
User: Why is there a subtraction in the last step?
Assistant: The subtraction is not necessary. $\ldots$ The correct step is simply adding i and $-i$ to get 0, and then adding i to get ____

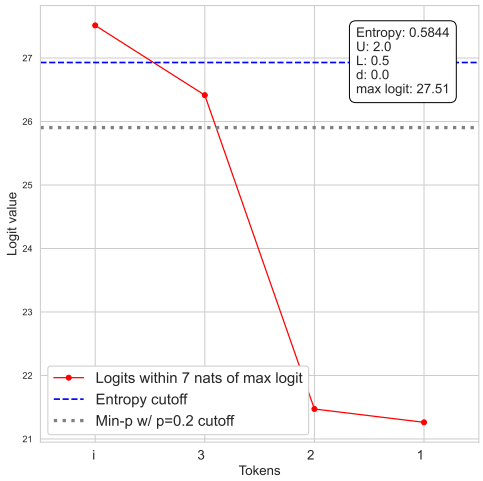


Figure 1: An instance in which entro-p filters out flawed tokens more selectively than min-p. This graph shows the logit scores of top tokens at a generation step in a conversation initiated by a math reasoning problem from the MATH dataset. The dotted grey line indicates the truncation level of min-p set to $p = 0.2$, and the dashed blue line indicates the truncation level of entro-p. The correct token is 'i' and is retained by both processors, but min-p also retains the incorrect token '3.'

4 Experimental Setup

4.1 Datasets

We evaluate the effectiveness of entro-p against min-p sampling and base sampling (using only temperature scaling) on three widely used benchmarks focused on reasoning: MATH (Hendrycks et al., 2021), MBPP (Austin et al., 2021), and AlpacaEval for creativity (Li et al., 2023). These benchmarks allow for easily verifiable solutions, either by comparing numerical outputs with provided answers or by executing simple Python scripts and checking for successful system exits.

Our experimental datasets for evaluations are as follows:

- The MATH dataset consists of 12,500 challenging math word problems with solutions written in LaTeXcontaining boxed final numerical answers (Hendrycks et al., 2021). We employ 4-shot prompting, following Gao et al. (2024)'s Minerva MATH task, without the use

of chat-templating.

- The MBPP (Mostly Basic Programming Problems) dataset consists of 974 problem statements requesting Python code to perform simple tasks, and test cases for each problem to check if a purported solution indeed solves the task at hand. We use 3-shot prompting without the use of chat-templating in order for the model to produce functioning Python code without the extraneous outputs typically produced by instruction-tuned models.

- The AIME24 dataset (Jia, 2024) consists of problem statements and solutions to the 2024 American Invitational Mathematics Examination (AIME). This is a small dataset containing 30 questions and answers. The final answers for these problems are numerical and reported separately from the problem solution for easy verification.

- The AlpacaEval Creative Writing uses LLM-as-a-judge to measure the efficacy of an LM on creative prompts (Li et al., 2023). The win rate is computed for all sampling methods against base sampling with temperature 1. We use GPT-4o as our LLM judge.

From MATH and MBPP, which are large benchmark datasets, we select 300 questions at random, filtering out problems and solutions from the MATH dataset that were beyond 600 tokens in combined length in order to avoid cases where the model's outputs are truncated before it reaches a final answer. We are unable to do pass@k evaluations on more than 300 questions, since we have to sample 500 questions per problem for a total of 150,000 runs per dataset. We carry out our evaluations on MATH and MBPP using EleutherAI's *language model evaluation harness* lm-eval (Gao et al., 2024) and using vLLM (Kwon et al., 2023) with roughly 2,400 Nvidia a4000 GPU hours. We evaluate pass@k on all problems in the AIME2024 dataset.

4.2 Evaluation Details

We test three logit processors: pure sampling with temperature scaling (base), min-p, and entro-p. On MBPP we tested these processors on the models Gemma-2-2b-it (Rivière et al., 2024), Mistral-7B-Instruct-v0.3 (Jiang et al., 2023), Llama-3.2-3B-Instruct, and Llama-3.1-8B-Instruct (Touvron et al., 2023). On the MATH and AIME24 datasets we tested the processors on Llama-3.2-3B-Instruct and Llama-3.1-8B-Instruct. We report the average pass@k score for each processor for a selection of values of k between 1 and 500. We compute pass@k as described in (2). In each case, we tested each logit processor with a grid of hyperparameters. The two hyperparameters which resulted in the best pass@100 scores were selected for each processor. The pass@k scores between 1 and 500 were then reported for each processor with its two possible best hyperparameter settings. See Figures 2 and 3.

We note that although we show smaller gains over past methods than Nguyen et al. (2024), we do far more extensive hyperparameter sweeps over competing methods to allow for a more fair and realistic comparison.

5 High pass@1 Performance Often Corresponds to Low pass@k Performance

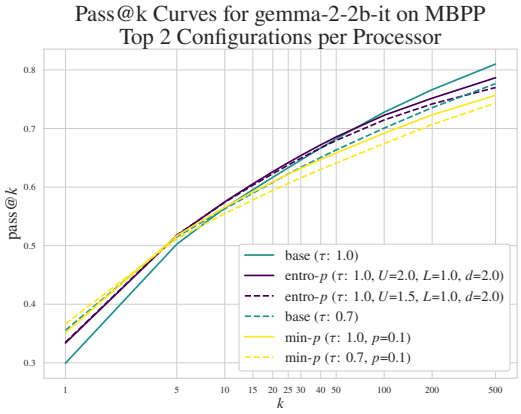
A natural goal in reasoning tasks where the answer is easy to verify is to maximize the chance that a single generated solution is correct. However, repeated sampling for sufficiently high k can expose a hidden trade-off: methods that aim to boost pass@1 performance often reduce the *coverage* of the model, significantly reducing pass@k performance.

In the following subsections, we provide empirical evidence and a toy model to illustrate how this phenomenon can arise.

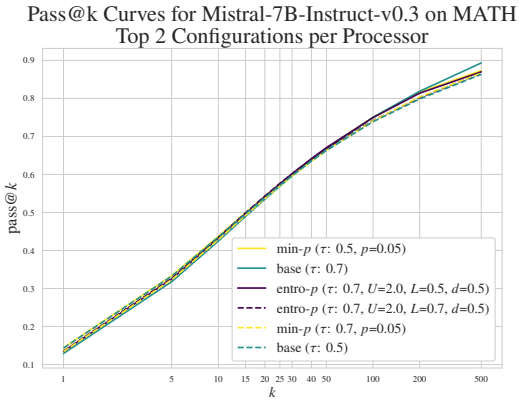
5.1 Empirical Evidence

We observe a general trend that persists throughout many of the evaluations, but is especially pronounced for MBPP: the logit processors that perform best for pass@1 perform among the worst for pass@100. For instance, in Figure 2a, we see that base sampling achieves an accuracy 7% lower than min-p evaluated by pass@1. However, when evaluated by pass@500, it outperforms min-p by 7%. In a complete reversal, the worst logit processor for pass@1 becomes the best for pass@500. The same trend emerges for base sampling in Figures 3a, b. This paradox is not restricted to the MBPP dataset: in Figures 3c, e we observe the same phenomemon on the MATH and AIME24 datasets.

The reason for this trend is likely that more conservative logit processors are more likely to succeed in pass@1 because they sample the most prob-



(a)



(b)

Figure 2: Pass@k scores for gemma-2-2b-it and Mistral-7B-Instruct-v0.3 on MBPP and MATH, respectively. Pure sampling, min-p, and entro-p are considered, with different values for their hyperparameters. Note that the logit processors that perform the best for pass@1 often underperform for pass@500 and vice versa.

able tokens at each step. However, over multiple attempts, they are unable to exhaustively explore the space of solutions, so they fail to find low-probability solutions that are necessary to solve hard problems. We offer a toy example to illustrate mathematically how this phenomenon can emerge in practice.

5.2 A Toy Example

In practice, it is analytically intractable to analyze token-by-token performance of language models at scale. Therefore, to motivate the existence of the paradox whereby high pass@1 performance can imply poor pass@k performance, we provide a toy example. This example provides one mechanism by which such a paradox can arise, though it is by no means the only mechanism.

Consider testing two language models A and B, which are trying to solve a problem with a single correct answer that is one token in length. Suppose that these two models employ different token selection methods.

Given an input sequence $x_{1:i}$ of tokens, model A considers a set of possible tokens S_A with size $|S_A| = m$. From the set S_A, model A will select a single token, chosen uniformly with probability $\frac{1}{m}$. The set S_A will be chosen deterministically based on $x_{1:i}$, i.e. S_A will always be the same set when A receives the same input sequence. Let the probability that S_A contains the correct token be p. This strategy represents a simplification of sampling methods such as top-p. On the other hand, assume that model B selects a token ranging across the entire vocabulary $\mathcal{V}$: this token has a probability q of being correct on each attempt. Suppose that $q \ll p/m$. We can now analyze the performance of each of these sampling strategies for different pass@k.

If S_A contains the correct answer, then Model A will solve the problem with probability $1/m$ on each attempt. The set S_A contains the correct answer with probability p, so model A succeeds on pass@k with probability

$$p(1 - (1 - 1/m)^k).$$

On the other hand, model B succeeds on pass@k with probability

$$1 - (1 - q)^k.$$

For pass@1, model A succeeds with probability p/m and model B succeeds with probability q. Thus, model A has better performance on pass@1. On the other hand, as $k \to \infty$, model A can never succeed with probability exceeding p, whereas the success probability for model B asymptotically approaches 1. Thus, it is possible for B to greatly outperform A for pass@k with k large while still vastly underperforming relative to A when k is small. See Figure 4.

This toy example illustrates that superior results on pass@1 may not translate to pass@k for higher k. Although this example may seem naive, it closely reflects the scaling behavior that we observe in practice. We may view model B as "more confident" in its outputs, despite often being wrong, whereas model A is less confident in its outputs.

6 Empirical performance of entro-p

6.1 Math and Coding Benchmarks

In Table 1, we report the pass@100 performances of entro-p and min-p processors used with various

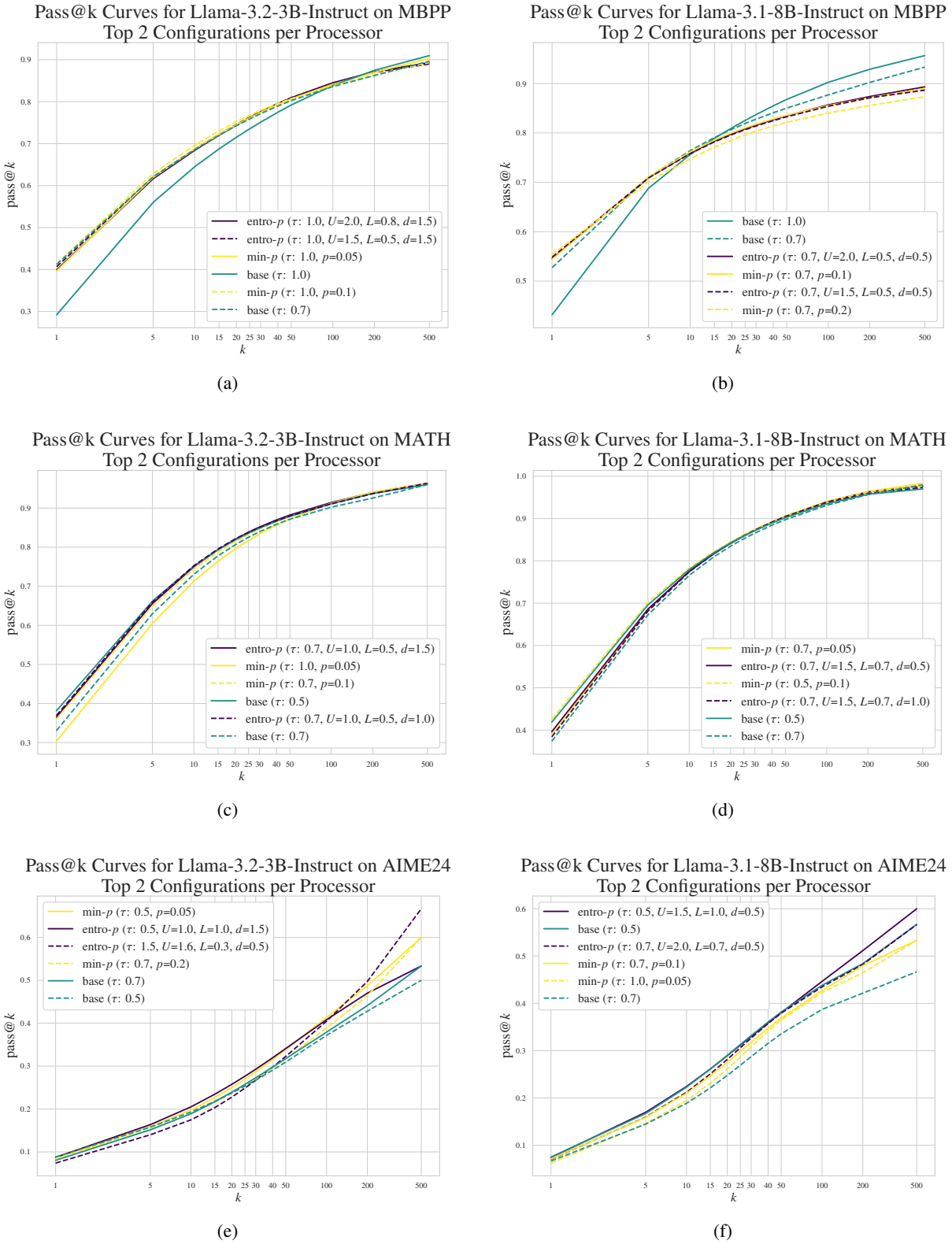


Figure 3: Pass@k scores for Llama-3.2-3B-Instruct and Llama-3.1-8B-Instruct on MBPP, MATH, and AIME24. Pure sampling, min-p, and entro-p are considered, with different values for their hyperparameters. Again we see entro-p overperforming on pass@k for large k and processors which overperform on pass@1 underperforming on pass@500 and vice versa.

models on the AIME24 and MBPP benchmarks as well as a dataset crafted from the AIME 2025 exam parts I and II (AOPS, 2025), formatted in the same manner as the AIME24 dataset. Entro-p

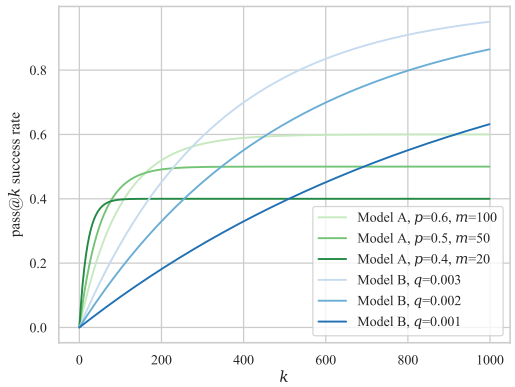


Figure 4: Toy models A and B scored on pass@k using different parameters. Notice that for small k, Model A is more effective than model B, but this trend reverses for large k.

Table 2: We compare top five configurations for each of **min-p** and **entro-p** on a **Alpaca-Eval** Creative Writing benchmark, reporting **Length-Controlled Win Rate**.

min-p Configurations	
Parameters	**LC Win Rate (%)**
$(\boldsymbol{\tau} = 1.5,\ \mathbf{p} = 0.1)$	58.12
$(\boldsymbol{\tau} = 1.5,\ \mathbf{p} = 0.15)$	56.73
$(\boldsymbol{\tau} = 1.5,\ \mathbf{p} = 0.2)$	55.45
$(\boldsymbol{\tau} = 1.0,\ \mathbf{p} = 0.05)$	55.07
$(\boldsymbol{\tau} = 2.0,\ \mathbf{p} = 0.2)$	54.82
entro-p Configurations	
Parameters	**LC Win Rate (%)**
$(\boldsymbol{\tau} = 1.2,\ \mathbf{d} = 1.0,\ \mathbf{l} = 1.0,\ \mathbf{u} = 2.0)$	**58.39**
$(\boldsymbol{\tau} = 1.0,\ \mathbf{d} = 0.75,\ \mathbf{l} = 1.0,\ \mathbf{u} = 2.0)$	56.96
$(\boldsymbol{\tau} = 1.0,\ \mathbf{d} = 0.5,\ \mathbf{l} = 1.2,\ \mathbf{u} = 2.0)$	56.31
$(\boldsymbol{\tau} = 1.5,\ \mathbf{d} = 1.5,\ \mathbf{l} = 1.0,\ \mathbf{u} = 1.5)$	56.30
$(\boldsymbol{\tau} = 1.0,\ \mathbf{d} = 0.5,\ \mathbf{l} = 1.0,\ \mathbf{u} = 2.5)$	56.17

outperforms min-p on 7 out of 11 model-dataset combinations. On 3 of the 4 other datasets entro-p underperformed min-p by just 0.1%.

Model	**Dataset**	**min-p**	**entro-p (ours)**
Llama-3.1-8B-Instruct	MATH	**94.0%**	93.9%
Llama-3.1-8B-Instruct	MBPP	85.5%	**85.7%**
Llama-3.1-8B-Instruct	AIME24	42.7%	**44.6%**
Llama-3.1-8B-Instruct	AIME25	21.3%	**22.8%**
Gemma-2-2b-it	MATH	78.4%	**78.5%**
Gemma-2-2b-it	MBPP	69.2%	**72.3%**
Llama-3.2-3B-Instruct	MATH	91.2%	**91.4%**
Llama-3.2-3B-Instruct	MBPP	84.1%	**84.5%**
Llama-3.2-3B-Instruct	AIME24	**41.3%**	40.8%
Mistral-7B-Instruct-v0.3	MATH	**75.1%**	75.0%
Mistral-7B-Instruct-v0.3	MBPP	**81.1%**	81.0%

Table 1: Highest **pass@100** score comparison for min-p and entro-p across multiple (Model, Dataset) pairs. Hyperparameters for all pairs except for the AIME25 benchmark can be found in the legends of the graphs in Figure 3 and Figure 2. The hyperparameters for entro-p on AIME25 are $U = 1.0, L = 0.8, d = 0.0$ with temperature 0.7, and for min-p are $p = 0.2$ with temperature 0.5.

6.2 AlpacaEval Creative Writing

We report the results of evaluating min-p and entro-p in Table 2. A hyperparameter sweep, as exemplified in the table, shows that entro-p beats min-p, the previous SOTA, using *length-controlled win rate*. Thus, entro-p is effective for both logical reasoning and creative tasks.

7 Conclusion

The results of this paper show clearly that the choice of logit processor can have a significant effect on pass@k performance for large k. Furthermore, superior pass@1 performance does not translate to superior pass@k performance for large k. In fact, the reverse is sometimes true, with a worst-performing logit processor on pass@1 becoming best-performing on pass@k for large k. Owing to its dynamic output, the entro-p processor is a promising new option that can outperform min-p. It performs well with respect to pass@k for large values of k on reasoning tasks. At the same time, this doesn't cause a sacrifice on creativity tasks, as shown using AlpacaEval. More work is warranted to investigate the underlying mechanism for an inverse relationship between pass@1 performance and pass@k performance.

Limitations

We outline some limitations of entro-p and the search for a "best" logit processor below:

1. The paradox of logit processors performing well on pass@1 and poorly on pass@k underlines the difficulty of choosing a logit processor that performs well on pass@k independently of the value of k. This limitation applies to entro-p too, despite its promising performance.

2. Moreover, our results on e.g. AIME24 indicate that the performance of different logit processors on the same task is model-dependent. This indicates further difficulties in choosing a single logit processor which will perform well on a given task using a given model.

3. So far, we lack a good statistical theory of the probability distributions predicted by lan-

guage models. Such a theory would be quite useful in designing a good logit processor. In particular, what statistics should be computed to choose an ideal cutoff for truncation sampling?

4. Consequently, logit processors such as min-p and entro-p rely on heuristics (e.g. on when to make a language model "more confident" or "less confident").

5. We compare models on creativity tasks using the AlpacaEval evaluator, which uses LLM-querying to decide on better or worse output. This has the advantage of being fast and cheap. However, LLM evaluations may not fully align with human evaluations of creativity. It would be useful to have human evaluators to compare entro-p and other logit processors on creativity.

References

David H. Ackley, Geoffrey E. Hinton, and Terrence J. Sejnowski. 1985. A learning algorithm for boltzmann machines. *Cogn. Sci.*, 9(1):147–169.

Anonymous. 2025. How do large language monkeys get their power (laws). ICML 2025 Conference Submission, unpublished manuscript.

AOPS. 2025. 2025 AIME. https://artofproblemsolving.com/wiki/index.php/AIME_Problems_and_Solutions. Accessed: 14 Feb 2025.

Jacob Austin, Augustus Odena, Maxwell I. Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie J. Cai, Michael Terry, Quoc V. Le, and Charles Sutton. 2021. Program synthesis with large language models. *CoRR*, abs/2108.07732.

Bradley C. A. Brown, Jordan Juravsky, Ryan Saul Ehrlich, Ronald Clark, Quoc V. Le, Christopher Ré, and Azalia Mirhoseini. 2024. Large language monkeys: Scaling inference compute with repeated sampling. *CoRR*, abs/2407.21787.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 39 others. 2021. Evaluating large language models trained on code. *CoRR*, abs/2107.03374.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *CoRR*, abs/2110.14168.

Angela Fan, Mike Lewis, and Yann Dauphin. 2018a. Hierarchical neural story generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 889–898, Melbourne, Australia. Association for Computational Linguistics.

Angela Fan, Mike Lewis, and Yann Dauphin. 2018b. Hierarchical neural story generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 889–898, Melbourne, Australia. Association for Computational Linguistics.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, and 5 others. 2024. A framework for few-shot language model evaluation.

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. 2017. On calibration of modern neural networks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1321–1330. PMLR.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the MATH dataset. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*.

John Hewitt, Christopher Manning, and Percy Liang. 2022. Truncation sampling as language model desmoothing. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 3414–3427, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

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

Maxwell Jia. 2024. Aime24. https://huggingface.co/datasets/Maxwell-Jia/AIME_2024.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de Las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7b. *CoRR*, abs/2310.06825.

Sumith Kulal, Panupong Pasupat, Kartik Chandra, Mina Lee, Oded Padon, Alex Aiken, and Percy S Liang. 2019. Spoc: Search-based pseudocode to code. *Advances in Neural Information Processing Systems*, 32.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.

Xuechen Li, Tianyi Zhang, Yann Dubois, Rohan Taori, Ishaan Gulrajani, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Alpacaeval: An automatic evaluator of instruction-following models. `https://github.com/tatsu-lab/alpaca_eval`.

Clara Meister and Ryan Cotterell. 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, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, pages 5328–5339. Association for Computational Linguistics.

Minh Nguyen, Andrew Baker, Clement Neo, Allen Roush, Andreas Kirsch, and Ravid Shwartz-Ziv. 2024. Turning up the heat: Min-p sampling for creative and coherent llm outputs. *Preprint*, arXiv:2407.01082.

Krishna Pillutla, Swabha Swayamdipta, Rowan Zellers, John Thickstun, Sean Welleck, Yejin Choi, and Zaïd Harchaoui. 2021. MAUVE: measuring the gap between neural text and human text using divergence frontiers. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 4816–4828.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. 2023. GPQA: A graduate-level google-proof q&a benchmark. *CoRR*, abs/2311.12022.

Morgane Rivière, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, Johan Ferret, Peter Liu, Pouya Tafti, Abe Friesen, Michelle Casbon, Sabela Ramos, Ravin Kumar, Charline Le Lan, Sammy Jerome, Anton Tsitsulin, and 80 others. 2024. Gemma 2: Improving open language models at a practical size. *CoRR*, abs/2408.00118.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurélien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. Llama: Open and efficient foundation language models. *CoRR*, abs/2302.13971.

Johnathan Xie, Annie S. Chen, Yoonho Lee, Eric Mitchell, and Chelsea Finn. 2024. Calibrating language models with adaptive temperature scaling. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, EMNLP 2024, Miami, FL, USA, November 12-16, 2024*, pages 18128–18138. Association for Computational Linguistics.

A Related work

A language model generates text one token at a time according to a probability distribution predicted by the language model for the next token. Several intuitive methods for choosing the next token include choosing the highest-probability token (greedy sampling), choosing the token that begins a highest probability sequence of tokens (beam search), or simply sampling from the predicted probability distribution (pure sampling). However, these methods lead to text that is either repetitive and predictable (greedy sampling and beam search) or incoherent (pure sampling) (Holtzman et al., 2020).

In order to improve on the factuality and creativity of these sampling methods, researchers have investigated the family of *truncation sampling* methods (Hewitt et al., 2022). Such methods include *top-k* sampling (Fan et al., 2018b), *top-p* sampling (Holtzman et al., 2020), *min-p* sampling (Nguyen et al., 2024), and *η-sampling* (Hewitt et al., 2022). Heuristics for choosing truncation sampling over other sampling methods are given in (Pillutla et al., 2021) and (Holtzman et al., 2020).

Another simple and commonly used technique to increase or decrease the creativity of a language model is *temperature scaling*, introduced in (Ackley et al., 1985) and (Guo et al., 2017). Temperature scaling is frequently used in combination with truncation sampling methods. Recently, *adaptive temperature scaling* has been introduced as a dynamic temperature scaling method improving calibration of models (Xie et al., 2024).

Specific truncation sampling methods sometimes lack rigorous statistical justification. On the other hand, (Hewitt et al., 2022) develops a justification for truncation sampling methods as "desmoothing" by modeling a language model's token predicted probability distribution as a convex combination of the true distribution and a noisy nearly uniform distribution. They use this to derive several principles that an ideal truncation sampling method should abide by. Finally, min-p scaling has shown improvements over top-p, top-k, and η-sampling across several benchmarks (Cobbe et al., 2021; Rein et al., 2023).