NORTHWESTERN UNIVERSITY

Diversity Seeking Chain-of-Thought Reasoning in Large Language Models

A THESIS

SUBMITTED TO THE GRADUATE SCHOOL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

for the degree

MASTER OF SCIENCE

Computer Science

By

Isaac Miller

EVANSTON, ILLINOIS

June 2024

Technical Report Number: NU-CS-2024-14

Abstract

Reinforcement learning in Large Language Models (LLMs) often suffers from mode collapse due to reward overfitting. Generative Flow Networks (GFlowNets) promote diversity and enhance LLMs' reasoning on complex tasks like GSM8K. This study integrates GFlowNets with LLMs, focusing on diverse rationale generation without losing correctness. Our results show that GFlowNets can improve LLMs' problem-solving abilities in data-limited contexts while maintaining diversity and preventing mode collapse.

Acknowledgements

I am profoundly grateful to Omar Khattab for his invaluable guidance, feedback, and support throughout this research project. His insights and thoughtful questions have been fundamental to its success.

My thanks go to the DSPy community for helping with technical assistance and for their helpful advice.

I am thankful to the Northwestern University MAGICS Lab for providing me with access to computing resources and to the Northwestern QUEST staff for their dedication to maintaining the cluster's health.

I want to give a special thank you to my family and friends for their unwavering support and encouragement.

Table of Contents

Contents

List of Figures and Tables

List of Figures

List of Tables

Introduction

1.1 LLMs

Large Language Models (LLMs) learn complex patterns in text by leveraging enormous sums of data. These models typically are trained on the next token prediction task autoregressively. Their applications span various domains, including text generation, translation, summarization, and question-answering. Despite their success, LLMs have limitations. They generate text based on a given prefix or prompt. The combination of the prefix and the model's autoregressive nature limits the model's ability to perform complex reasoning.

1.2 Reasoning as a Latent Variable Problem

Autoregressive LLMs, optimized to predict tokens based on preceding context, face limitations in performing tractable inference over their vast knowledge base, restricting them to sampling conditioned on a prefix.

Given a question-answer pair (X, Y), the objective is to find sequences of tokens Z, which in this context is a chain of thought that maximizes the conditional probability of Y given X and Z.

$$p_{\mathrm{LM}}(Y \mid X) = \sum_{Z} p_{\mathrm{LM}}(Y \mid XZ) p_{\mathrm{LM}}(Z \mid X) \tag{1.1}$$

where $p_{\mathrm{LM}}(ZY \mid X)$ is the likelihood of the output Y given the input X and the chain of thought Z, and $p_{\mathrm{LM}}(Z \mid X)$ is the prior probability of the chain of thought Z given the input X. We denote the concatenation of token sequences as a single string, e.g., XZY.

Previous work has used prompting and in-context learning to generate sequences Z which lead to the correct output Y. As was done in Hu, Jain, et al. 2024, we treat Z as a hidden variable in a latent variable problem. This problem setup now allows us to treat chain-of-thought reasoning as a Bayesian inference problem, where we wish to sample from the posterior distribution $p_{\text{LM}}(Z \mid X, Y) = \frac{p_{\text{LM}}(XZY)}{\sum_{Z'} p_{\text{LM}}(XZ'Y)}$.

Sampling from this posterior is a challenging problem. We can evaluate $p_{\text{LM}}(XZY)$ in a straightforward manner using the log-likelihood of the output Y given the input X and the chain of thought Z. However, this does not allow us to sample from the posterior distribution $p_{\text{LM}}(Z \mid X, Y)$.

1.3 GFlowNets in LLMs

Generative Flow Networks (GFlowNets), introduced by Bengio et al. 2021, present a novel approach to generating diverse samples from intractable distributions. GFlowNets are diversity-seeking reinforcement learning algorithms designed to train policies that sample objects, such as token sequences Z, with probability proportional to a given reward function. As was done in Hu, Jain, et al. 2024, we use the joint likelihood of the output Y and the chain of thought Z as the reward function. Previous work showed that using GFlowNets in this manner was possible, and the goal of this paper is to apply this to a standard LLM benchmark. To quote Hu, Jain, et al. 2024, "GFlowNet objectives provide a principled and flexible approach to fine-tuning LLMs to match a target distribution where reward-maximizing RL fails to." We discuss GFlowNets more in Section 2.2

1.4 Problem Statement

There remain significant challenges in handling longer chains of thought and generating diverse, coherent rationales for different multi-step reasoning problems. Traditional search methods such as Markov Chain Monte Carlo (MCMC) and reinforcement learning (RL) approaches fail in mode collapse settings, where policies will settle around a small number of potential modes (Gao, Schulman, and Hilton 2022).

It is important to note that sampling from autoregressive language models is tractable as a product of ordered conditionals. For any token sequence X, you can find the conditional probability by measuring each token's probability in relation to the token sequence before it from left to right. The problem becomes intractable when you cannot sample from left to right based on this conditional distribution.

This work can be considered a direct follow-up to Hu, Jain, et al. 2024, where the most complex planning and reasoning problem required a model to learn tool arithmetic tool use. We extend this to mathematical word problems requiring multi-step reasoning. We propose that fine-tuning an LLM as a GFlowNet policy (referred to as GFN-FT) will allow us to improve the model's ability to generate diverse and coherent rationales in extremely data-limited settings on contemporary benchmarks.

1.5 Contributions

Our contributions include:

- Methodology for scaling GFlowNets to longer chains of thought in LLMs
- Practical findings on sampling and dataset size for training LLMs as GFlowNets
- Results for using an LLM as a GFlowNet performance in an extremely data-limited environment
- Empirical results showing the effectiveness of GFlowNets for improving diversity and reasoning ability on a modern benchmark

Literature Review

2.1 Prompting Methods

2.1.1 In-context Learning

In-context learning (Brown et al. 2020; Zhou et al. 2022; Dong et al. 2023) is a well-studied method for enhancing the quality of model outputs. This approach involves providing the model with examples of input-output pairs, enabling it to infer patterns and generate more accurate responses based on the given context. By presenting the model with relevant examples within the prompt, in-context learning leverages the model's ability to generalize from specific instances to broader applications.

Recent advancements in large language models have demonstrated the efficacy of zero-shot prompting (Kojima et al. 2023). Zero-shot prompting refers to the model's capability to generate outputs without being explicitly shown examples of those outputs during training or within the prompt. This ability underscores the robustness and adaptability of modern LLMs, which can effectively apply their learned knowledge to new and unseen tasks.

A core subset of in-context learning is few-shot prompting. This technique involves presenting the model with a few examples of the task at hand within the prompt. We capture the effectiveness of few-shot prompting by the following inequality for a set of n examples of (x_i, y_i) question-answer pairs:

$$p(Y \mid X, (x_1, y_1), (x_2, y_2), ..., (x_n, y_n)) \geq p(Y \mid X) \tag{2.1}$$

This inequality suggests that the model's performance on the task (predicting Y given X) improves when the model sees relevant examples (x_i, y_i) within the prompt.

2.1.2 Chain-of-Thought Prompting

Chain-of-Thought (CoT) prompting (Wei, Wang, et al. 2023) has emerged as a powerful technique to enhance the reasoning capabilities of large language models. Initially, CoT prompting was designed to help models break down complex, multi-step problems into smaller, manageable tasks that the model can solve sequentially. This approach leverages the model's ability to handle each step individually, resulting in a comprehensive solution to the overall problem. CoT was originally used inside of few-shot learning scenarios, where a few examples of the reasoning process guide and encourage the model to explain its reasoning prior to answering.

In Kojima et al. 2023, they add "Let's think step by step" before each answer. While this does not replace the role that few-shot prompting plays in terms of enticing the model to give better responses, it is a way to test the inherent reasoning abilities existing in the model.

These methods encourage models to generate intermediate reasoning steps, denoted as Z, which, given a QA pair (X, Y), help the model arrive at the correct output Y from a given input prompt X. By explicitly generating these reasoning chains, the model can produce outputs that are not only more likely to be correct but also provide a transparent thought process.

There is a similar guiding inequality to that of few-shot prompting:

$$p(Y \mid XZ) \geq p(Y \mid X) \tag{2.2}$$

That inequality states that few-shot prompting is generally helpful rather than a proven inequality. While CoT prompting often leads to outputs broken down into smaller steps, these outputs are not always correct. An expanding body of research is focused on improving prompting methodologies to ensure the generated outputs are both accurate and diverse. Researchers are exploring various techniques to refine these prompts and to encourage models to generate multiple plausible reasoning chains, thereby increasing the likelihood of correct outcomes.

2.1.3 Prompt Optimization with DSPy

DSPy (Khattab, Singhvi, et al. 2023; Khattab, Santhanam, et al. 2022) is a framework for automatically optimizing Language Model pipelines. It abstracts pipelines as text transformation graphs, enabling the systematic optimization of prompts and few-shot examples. DSPy modules, similar to neural network layers, can be parameterized to adapt to various tasks and datasets, including some forms of constrained generation (Singhvi et al. 2024).

We use DSPy to identify the most effective prompts for fine-tuning and enhancing model performance. Details on our implementation are in Section 3.1.4, with prompts documented in Appendix A. DSPy also evaluates model performance post-training, ensuring optimal prompt configurations throughout the model's lifecycle.

Other frameworks like SAMMO (Schnabel and Neville 2024) and Promptbreeder (Fernando et al. 2023) exist, but DSPy's comprehensive abstractions and modular design make it superior for prompt and pipeline optimization. Its task-agnostic nature allows for efficient adaptation to new tasks with minimal manual changes.

2.2 Generative Flow Networks

Generative Flow Networks, initially introduced by Bengio et al. 2021, provide a novel approach to generating diverse samples from intractable distributions. Unlike traditional Monte Carlo Markov Chain (MCMC) methods (Hastings 1970; Andrieu et al. 2003), GFlowNets use a learned policy to construct objects sequentially, ensuring the probability of generating an object is proportional to a predefined reward function.

GFlowNets have been successfully applied to various probabilistic modeling tasks beyond language, such as molecule synthesis (Bengio et al. 2021) and sequence generation. Madan et al. 2023 and Malkin et al. 2023 demonstrate that the Subtrajectory Balance (SubTB) objective accelerates convergence and improves stability in environments with sparse rewards and extended action sequences.

2.2.1 GFlowNets in Language Models

In the context of language models, GFlowNets attempt to sample from complex posterior distributions efficiently as was initially done in Hu, Jain, et al. 2024. By amortizing the computation required for training the sampler, GFlowNets ensure that the language model can operate without additional computational overhead during inference. This property is essential for user-facing systems where quick and efficient text generation is critical. During inference, GFlowNets trained with an autoregressive language model can produce high-quality samples by effectively learning to navigate the latent spaces defined by the task-specific constraints.

Empirical evidence demonstrates that GFlowNet fine-tuning can significantly improve the performance of language models across various tasks. For instance, in a subjectivity classification task, GFlowNet fine-tuning outperformed traditional supervised fine-tuning (Devlin et al. 2019) and Proximal Policy Optimization (PPO) (Schulman et al. 2017) by a substantial margin, particularly in low-data regimes. Similarly, for tasks requiring multi-step reasoning and tool use, such as arithmetic problem-solving, GFlowNet fine-tuning achieved higher accuracy and better generalization to out-of-distribution examples compared to other fine-tuning methods.

2.3 Fine-Tuning

In the context of language models, Wei, Bosma, et al. 2022 initially showed that fine-tuning with instructions can be used to improve the performance of LLMs. Since then, different fine-tuning methods have been proposed.

2.3.1 Fine-Tuning Methods

Proximal Policy Optimization

Proximal Policy Optimization (PPO), introduced by Schulman et al. 2017, is a policy gradient method for reinforcement learning (RL) that allows for multiple minibatch updates over a number of epochs. In the context of language models, PPO treats the language model as the policy and updates the policy according to a dataset, aiming to maximize likelihood. The issue with maximizing likelihood directly is that it does not seek to learn the distribution of the data, which can lead to overfitting to the reward model and mode collapse (Gao, Schulman, and Hilton 2022).

Reinforcement Learning from Human Feedback

Reinforcement Learning from Human Feedback (RLHF) is a method for fine-tuning LLMs by aligning the model's output distribution to be closer to that of the preference given by human labelers. Christiano et al. 2017 pioneered the use of RLHF for deep learning, and Ouyang et al. 2022; Stiennon et al. 2022 showed that RLHF can be used to fine-tune language models. It goes through a two-step process: first, a reward model learns from preference data, then the policy model is fine-tuned to the reward model, typically using PPO. The model will take steps toward maximizing the reward but use the policy of the original model to ensure that it does not diverge significantly from the original model due to clipping.

2.4 Motivating Example

In our exploration of arithmetic problem-solving, we draw upon the foundational work of Hu, Jain, et al. 2024, which demonstrates the efficacy of solving arithmetic problems through step-by-step reasoning. This approach leverages GFlowNet Fine-Tuning (GFN-FT) to enhance the model's ability to plan and use tools effectively. Our findings corroborate the increase in planning abilities through GFN-FT.

Moreover, our results extend beyond planning. The GFN-FT approach enables models to learn planning and generalize to previously unseen problems through amortized inference. This capability is particularly significant because it suggests that language models can improve their reasoning skills while avoiding the mode collapse problems of PPO.

To empirically validate this hypothesis, we conducted experiments on a modern benchmark, GSM8K (Cobbe et al. 2021). The results demonstrate not only an increase in the diversity of generated solutions but also a significant improvement in the reasoning capabilities of the models. This empirical evidence supports our assertion that GFN-FT can effectively augment the reasoning strength and output diversity of LLMs, providing a solid foundation for future advancements in this domain.

Methodology

3.1 Dataset

3.1.1 GSM8K Description

The GSM8K dataset (Cobbe et al. 2021) consists of 8,500 grade school math word problems designed to test multi-step reasoning skills. For this study, we focused on evaluating the model's ability to perform arithmetic without external tools, diverging from previous studies like Hu, Jain, et al. 2024, where they used a tool-assisted approach for arithmetic operations.

We utilize those rationales inside of the fine-tuning setup to seed the Replay Buffer, as is discussed in Section 3.2.3.

3.1.2 Preprocessing

In the GSM8K dataset, calculations are explicitly marked within "<<...>>" tags. For example, the calculation "3 + 2 = 5" is denoted as "<<3 + 2 = 5>>". This markup facilitates tool usage by delineating the arithmetic operations that need to be performed. However, to assess the model's inherent arithmetic capabilities, we opted to remove the markup from the dataset and turn off tool usage.

3.1.3 Pre-seeding the Replay Buffer

To initialize the model's learning process, we pre-seeded the replay buffer with 16 rationales extracted from the GSM8K dataset, corresponding to the 16 data points in the data set. These rationales provide a foundation from which the model can begin generating reasoning chains. This initial step is crucial for setting a baseline understanding of how to approach problem-solving in the context of arithmetic without external assistance.

Hu, Jain, et al. 2024; Vemgal, Lau, and Precup 2023 both state, along with many other RL papers, that utilizing a Replay buffer is crucial for the model to learn to perform well. We come to the same conclusion. Without a replay buffer, the model sometimes will struggle to find the correct formatting for this specific setup.

When the model generates a new rationale, the buffer will keep the rationale with the higher score according to the reward model.

3.1.4 DSPy Implementation

Few-Shot Examples

To identify the optimal prompt for both fine-tuning and the reward model, we employed a prompt optimization process using a straightforward chain-of-thought pipeline. This pipeline is designed to provide a structured prompt and format for the model, including optional few-shot examples to guide the model's output generation. Specifically, the pipeline accepts a question as input and generates a corresponding rationale followed by an answer.

The optimization process used a DSPy prompt optimizer, which systematically evaluated various prompts to determine the most effective one for fine-tuning. Additionally, to ensure the model comprehends the required formatting, we manually created a simple in-context example, as detailed in Appendix A. For this study, the optimal prompt identified by DSPy was manually incorporated into the fine-tuning code.

Evaluation

For the evaluation phase, we utilized the built-in DSPy evaluator. This tool accepts a set of input examples and tests the pipeline's output against the expected answers using a predefined metric. Specifically, we employed a metric from the DSPy library that extracts the last number present in the model's answer to compare it with the correct answer.

Rationale Collection

To collect rationales, we processed the test set through the optimized pipeline. Each rationale generated was then categorized as correct or incorrect using the same metric applied during evaluation. VLLM (Kwon et al. 2023) facilitates this classification by simplifying the process of switching between different LoRA configurations during evaluation. This orchestration enabled efficient and flexible testing of the model's outputs across temperatures and LoRA configurations.

3.2 Fine tuning setup

3.2.1 Training Details

All of the fine-tuning was conducted on a single A100 SXM 80GB GPU. To optimize the model's memory usage, we quantized the model to 8 bits, reducing the size in VRAM and computational requirements on the GPU. Additionally, for a cost-effective fine-tuning process, we employed Quantized Low-Rank Adaptation (QLoRA) (Dettmers et al. 2023; Hu, Y. Shen, et al. 2021) to lower memory usage.

For our fine-tuning setup, we trained the model for 100 epochs with a batch size of 16 and a learning rate of 0.0005. To further manage computational resources, we utilized 16 gradient accumulation steps. During training, the temperature was randomly sampled between 0.5 and 2 to introduce variability and improve generalization. For each question, we sampled three rationales, differing from the approach in Hu, Jain, et al. 2024, where ten rationale samples were used.

The LoRA configuration targeted all available attention layers in Mistral-7B, with a LoRA rank of 32, a LoRA alpha of 16, and a LoRA dropout rate of 0.1.

The reward temperature started at 1 and decayed to 0.5 over a horizon of 100 steps, which helped in adjusting the exploration-exploitation balance during training. Additionally, we applied a penalty to the reward to constrain the vocabulary, guiding the model to generate more appropriate outputs. We incorporated a replay buffer with a 25% chance of reusing previous samples to stabilize training and improve convergence. See Section 3.2.3 for more details on the replay buffer.

Early stopping was set with a patience of 15 epochs based on the validation loss to prevent overfitting. We selected the top three checkpoints for evaluation, which occurred at epochs 10, 23, and 30. Beyond epoch 30, we observed a degradation in output quality, likely due to the instability common in many reinforcement learning-based methods.

An important parameter for this task is the maximum generation length for the model. Due to memory constraints, the longer the generation length is, the fewer samples you are able to take per training step. For this reason, we select training examples where the example rationale is less than 70 tokens and cap the model's generation length at 70.

3.2.2 Limited Data Regime

In our study, we deliberately constrained the model to train on only 16 examples to evaluate its ability to generalize from a small subset to the entire dataset. This approach allows us to demonstrate that even with limited training data, the model can effectively learn and apply its reasoning capabilities across a broader set of problems. We hypothesize that this generalization capability can scale to different datasets. For instance, a model trained on the GSM8K dataset with the GFlowNet fine-tuning regime should be able to extend its reasoning abilities to other datasets such as the more challenging MATH dataset (Hendrycks et al. 2021), maintaining the favorable properties observed in this study. Further discussion on the implications and potential scalability of this approach can be found in Section 5.

3.2.3 Task Model

We use *Mistral 7B v0.2 instruct* (Jiang et al. 2023) to represent the GFlowNet architecture. As shown in Hu, Jain, et al. 2024, we can use LLMs to represent the GFlowNet policy itself.

We view the autoregressive token outputs as the different trajectories in the generative flow network. Because GFlowNets aim to sample trajectories proportional to their rewards, we view this as the goal of the model in this paper. The language model needs to be able to generate a variety of plausible and coherent chain-of-thought outputs.

Learning Objective

In our exploration, we adhere to the formulations from prior work, specifically adopting the notation and constructs used in Hu, Jain, et al. 2024. Here, q_{GFN} denotes the Generative Flow Network from which we sample sequences Z. These sequences are structured as $Z = z_1, z_2, \ldots, z_n \top$, where $\top$ is a designated stop symbol. This stop symbol is generated either when the model outputs the termination string "####" or when it reaches the predefined maximum sequence length. This termination logic is showcased within the few-shot example to orient the model's generation process.

Continuing with the established notation, we aim to refine the policy $q_{\text{GFN}}(\cdot \mid \cdot; \theta)$. This policy should uphold the intrinsic properties of GFlowNets, ensuring that the generated sequences Z are such that $R(Z)$ is proportional to $q_{\text{GFN}}^{\top}(Z)$. Here, $q_{\text{GFN}}^{\top}(Z)$ represents the marginal likelihood of encountering the termination token subsequent to a sequence $Z_{1\ldots n}$.

To effectively train our GFlowNet, we employ the modified version of the sub-trajectory balance objective as discussed in previous studies (Hu, Jain, et al. 2024; Madan et al. 2023). This objective is expressed as follows:

$$\mathcal{L}(Z;\theta) = \sum_{0 \le i < j \le n} \left(\log \frac{R(z_{1:i}\top) \prod_{k=i+1}^{j} q_{\text{GFN}}(z_k \mid z_{1:k-1}) q_{\text{GFN}}(\top \mid z_{1:j})}{R(z_{1:j}\top) q_{\text{GFN}}(\top \mid z_{1:i})} \right)^2, \tag{3.1}$$

Replay Buffer

Part of the training policy is to sample from a replay buffer (Jain et al. 2023; Deleu et al. 2022; M. W. Shen et al. 2023). For each question, we store the highest reward trajectory found so far. We sample from the replay buffer with 25% probability for each data point in an epoch.

As was suggested in Vemgal, Lau, and Precup 2023; Hu, Jain, et al. 2024, we seed the replay buffer with example rationales from the dataset. Specifically, we seed the buffer with examples for all of our training examples, which will be evicted upon finding a higher reward rationale.

3.2.4 Reward Model

The reward model in our approach follows the structure outlined in previous works, specifically Hu, Jain, et al. 2024. The reward model is designed to evaluate the quality of a generated rationale Z by computing the probability of an answer Y given an input X and the rationale Z under the likelihood model $p_{\text{LM}}(Y \mid XZ)$.

Formally, the reward for a rationale Z is defined as:

$$R(Z) = p_{\text{LM}}(XZY) \propto p_{\text{LM}}(Z \mid X, Y) \tag{3.2}$$

In this equation, p_{LM} represents the likelihood assigned by the language model to the sequence XZY, effectively measuring how well the generated rationale Z supports the correct answer Y given the input X.

For this study, we did not employ a specifically trained reward model. Instead, we utilized the base language model to provide the reward. This approach simplifies the training process and reduces computational overhead, yet it maintains the overall integrity of the evaluation by leveraging the language model's inherent capabilities.

We hypothesize that incorporating a more capable teacher reward model while continuing to use a weaker student model for sampling could further enhance the model's generalization abilities. We discuss this as a future direction in Section 6.3.

3.3 Experimental Setup

3.3.1 Temperature Variation

We consider temperature to be a critical variable in training GFN-FT LMs. Temperature affects the randomness and diversity of the generated text in autoregressive LLM text generation. For example, when presented with the prompt “the color of the sky is,” a lower temperature makes the model more conservative, centering the distribution on the word ”blue,” while a higher temperature increases output diversity, potentially leading to other less likely choices, such as ”beautiful.”

The temperature affects token sampling probability as shown in Equation 4.1:

$$P(x_i) = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)}$$

We evaluate our models at temperatures of 0, 0.4, 0.8, and 1.2 to observe how temperature variations impact the diversity and correctness of generated rationales.

3.3.2 Evaluation Metrics

Metric Definitions

To assess the performance of the model, we employ two primary metrics: correctness and diversity of rationales.

Correctness refers to the raw percentage of correct outputs generated by the model in a pass@20 scenario. The model generates 20 rationales for each problem, and we measure how many of these rationales lead to a correct solution. A fundamental property of GFlowNets is their ability to sample diverse trajectories that can solve a problem. We believe that this approach provides a more comprehensive assessment of the model’s problem-solving capabilities than a single pass@1 scenario.

Diversity of Rationales is measured using both unigram diversity for evaluation. The diversity of rationales as a whole provides insight into the variety of solutions the model generates. Additionally, we specifically analyze the diversity of correct, incorrect, and combined rationales. We also examine the diversity of the rationales considered as trajectories using unigram diversity to understand the extent of variation within the generated sequences.

Rationale for Metrics

We measure the correctness of the model in a pass@20 scenario, reflecting the model's ability to generate multiple rationales that lead to the correct answer. We also emphasize the importance of semantic diversity in the pass@20 scenario because it reflects an essential attribute of GFlowNets: their capacity to sample diverse trajectories. Evaluating the diversity of correct, incorrect, and combined rationales is crucial because diversity alone is insufficient if the more varied solutions are predominantly incorrect. To do this, we measure unigram diversity within the trajectories, another critical property of GFlowNets, to ensure that the generated rationales exhibit substantial variation and richness.

3.4 Experiment 1: Evaluation Correctness

In this experiment, we evaluate the model's correctness using the three best checkpoints selected based on validation loss. These checkpoints are assessed on the test set at four different temperatures: 0, 0.4, 0.8, and 1.2. DSPy orchestrates the evaluation process, ensuring consistent prompts and settings across all models.

In this setup, the model generates 20 rationales for each problem, and we measure how many of these rationales lead to a correct solution. The results of this evaluation are detailed in Table 4.1.

3.5 Experiment 2: Diversity of Rationales

In this experiment, we evaluate the diversity of the model's correct rationales across the same four temperatures as above for the same set of checkpoints: 0, 0.4, 0.8, and 1.2.

For each temperature, we generate 20 rationales for each problem. We then measure the unigram diversity of the correct rationales for each problem. We only consider this across 80 questions in the test set due to computational limitations. The results of this evaluation are detailed in Figure 4.1.

Results

4.1 Correctness on Evaluation

4.1.1 Performance Analysis

	Temperature			
Model	**0**	**0.4**	**0.8**	**1.2**
Mistral-7B-Instruct-v0.2	26.81%	25.56%	23.13%	19.57%
Mistral-GFN-FT 10 Epochs	**26.88%**	**27.13%**	**26.63%**	**22.06%**
Mistral-GFN-FT 23 Epochs	16.50%	18.44%	17.50%	12.81%
Mistral-GFN-FT 30 Epochs	21.00%	18.94%	18.13%	14.69%

Table 4.1: Correctness results across different temperatures for various models. The table shows the percentage of correct answers for the Mistral-7B-Instruct-v0.2 model and the Mistral-GFN-FT models fine-tuned for 10, 23, and 30 epochs at four different temperatures: 0, 0.4, 0.8, and 1.2. We highlight the highest percentage for each temperature setting in bold.

We observed that the model fine-tuned for ten epochs performed the best on the test set across all temperature settings. In contrast, models fine-tuned for more extended periods exhibited significantly lower correctness percentages, which we attribute to the inherent instability of reinforcement learning (RL). As illustrated in Figure 4.1, models trained after ten epochs demonstrated greater diversity in their correct answers but had a lower percentage of correct rationales overall.

4.2 Diversity of Rationales

We see in Figure 4.1 that the model is able to generate more diverse correct rationales as the temperature increases and that the 30 epoch model has the most diverse correct rationales. Notably, the GFN-FT models most consistently improve over the base Mistral model in the diversity of the generated rationales. This highlights the strength of the GFN-FT process in

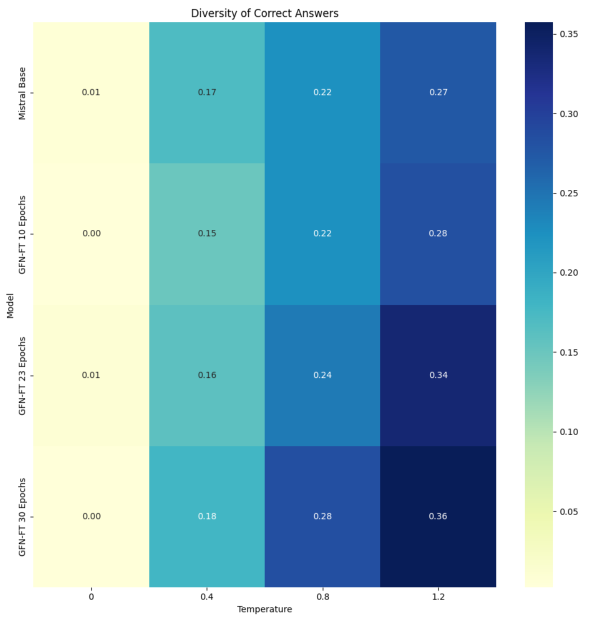

Figure 4.1: Diversity of Answers across Different Models and Temperatures. The heatmap shows the diversity of correct answers across the different checkpoints and temperature settings. The diversity is taken through unigram diversity. The top row on all charts is from the base Mistral model, the next row is from the GFN-FT model after ten epochs, the next row is from the GFN-FT model after 23 epochs, and the last row is from the GFN-FT model after 30 epochs. The x-axis is the temperature setting, and the color is the average diversity of the rationale.

enhancing the variety of outputs, even though it may sometimes come at the cost of accuracy.

Discussion

5.1 New Insights

We have successfully validated Generative Flow Networks as a practical methodology for enhancing performance on a contemporary LLM benchmark in a data-limited regime, prompting further exploration into their capabilities for complex reasoning tasks. Notably, as training progresses, we observe an increase in sequence diversity, indicative of the model's expanding problem-solving repertoire. However, significant degradation in output quality occurs with prolonged GFlowNet fine-tuning under these circumstances, a phenomenon not reported in LLMs in prior studies such as Hu, Jain, et al. 2024, which conducted longer training sessions than this study. The optimal strategies to mitigate this decline in quality remain unclear, marking an area for future investigation.

5.2 Research Decisions

Several pivotal decisions shaped the outcomes of our research. Initially, the choice of how many samples to take for each training point during an epoch was non-trivial. Through experimentation with two to eight samples, we identified three samples as the optimal balance for memory usage, training speed, and maximum token length. This issue could potentially be addressed by not treating all tokens as terminable states; however, such a modification would sacrifice some of the intrinsic properties of GFlowNets that benefit training dynamics. Another significant decision concerned the extent of the dataset utilized. Although the GSM8K dataset contains 7,500 training examples, we opted for a significantly smaller subset of 16 examples in this study, leaving the exploration of larger training sets as a promising direction for further research, discussed in Section 6.3.

5.3 Code Release

In our commitment to supporting the research community, we plan to release the code used in this paper publicly. Our implementation builds upon the foundation provided by Hu, Jain, et al. 2024's repository, which includes many of the necessary functions for applying GFN tuning to LLMs. We have designed the code so that researchers can quickly adapt it to new datasets involving chain-of-thought reasoning by making minimal adjustments to the data generator, facilitating further experimentation and development in this field.

Conclusion

6.1 Conclusion

In this study, we explored the application of GFlowNets to large language models, focusing on enhancing their problem-solving capabilities on a modern math benchmark. Our results demonstrate that GFN fine-tuning can indeed improve the performance of LLMs in harder and more data-limited settings than had been tried previously. The increased performance comes through increasing the diversity and correctness of the model outputs.

6.2 Limitations

While the results are promising, several limitations were noted. The process is inherently memory-intensive and computationally expensive. Moreover, treating all tokens as potentially terminating introduces inefficiencies that may not be sustainable at scale. These issues are common challenges in applying active learning techniques.

6.3 Future Work

Looking ahead, several avenues appear promising. Scaling GFN fine-tuning to larger models and datasets remains a significant challenge and an important next step.

Given that we only use 16 out of the 7,500 training examples, we can explore larger training sets to improve the model's performance and avoid some of the training instability that we observed.

Another intriguing direction is the use of GFNs in a self-play paradigm, where the model incrementally tackles more difficult problems, potentially improving its reasoning capabilities over time. Further exploration into a 'tree of thoughts' (Yao et al. 2023) framework could align GFNs more closely with natural reasoning patterns and avoid the cost of considering every token as terminal.

More exploration into better reward models could prove to be a computationally inexpensive way to improve the outputs of the GFlowNet policy. By using a smarter reward model, we would be able to draw samples using the smaller models and score those samples with a model that is better at reasoning. Using a smarter model would give us a better reward while still maintaining the computational efficiency of sampling from the smaller model.

6.3.1 Practical Applications

GFN-FT LLMs have significant practical applications, particularly in improving the performance-temperature tradeoff when generating rationales for complex problems.

The integration of DSPy (Khattab, Singhvi, et al. 2023) with GFN fine-tuning could potentially enhance model performance. DSPy currently increases the temperature to attempt to find more diverse rationales, and this follows as something that would be helpful in that scenario.

References

Andrieu, Christophe et al. (2003). "An Introduction to MCMC for Machine Learning". In: *Machine Learning* 50.1–2, pp. 5–43.

Bengio, Emmanuel et al. (2021). *Flow Network based Generative Models for Non-Iterative Diverse Candidate Generation.* arXiv: 2106.04399 [cs.LG].

Brown, Tom B. et al. (2020). *Language Models are Few-Shot Learners.* arXiv: 2005.14165 [cs.CL].

Christiano, Paul et al. (2017). *Deep reinforcement learning from human preferences.* arXiv: 1706.03741 [stat.ML].

Cobbe, Karl et al. (2021). *Training Verifiers to Solve Math Word Problems.* arXiv: 2110.14168 [cs.LG].

Deleu, Tristan et al. (2022). *Bayesian Structure Learning with Generative Flow Networks.* arXiv: 2202.13903 [cs.LG].

Dettmers, Tim et al. (2023). *QLoRA: Efficient Finetuning of Quantized LLMs.* arXiv: 2305.14314 [cs.LG].

Devlin, Jacob et al. (2019). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.* arXiv: 1810.04805 [cs.CL].

Dong, Qingxiu et al. (2023). *A Survey on In-context Learning.* arXiv: 2301.00234 [cs.CL].

Fernando, Chrisantha et al. (2023). *Promptbreeder: Self-Referential Self-Improvement Via Prompt Evolution.* arXiv: 2309.16797 [cs.CL].

Gao, Leo, John Schulman, and Jacob Hilton (2022). *Scaling Laws for Reward Model Overoptimization.* arXiv: 2210.10760 [cs.LG].

Hastings, W. K. (1970). "Monte Carlo sampling methods using Markov chains and their applications". In: *Biometrika* 57.1, pp. 97–109. DOI: 10.1093/biomet/57.1.97. eprint: http://biomet.oxfordjournals.org/cgi/reprint/57/1/97.pdf. URL: http://biomet.oxfordjournals.org/cgi/content/abstract/57/1/97.

Hendrycks, Dan et al. (2021). *Measuring Mathematical Problem Solving With the MATH Dataset.* arXiv: 2103.03874 [cs.LG].

Hu, Edward J., Moksh Jain, et al. (2024). *Amortizing intractable inference in large language models.* arXiv: 2310.04363 [cs.LG].

Hu, Edward J., Yelong Shen, et al. (2021). *LoRA: Low-Rank Adaptation of Large Language Models*. arXiv: `2106.09685 [cs.CL]`.

Jain, Moksh et al. (2023). *Biological Sequence Design with GFlowNets*. arXiv: `2203.04115 [q-bio.BM]`.

Jiang, Albert Q. et al. (2023). *Mistral 7B*. arXiv: `2310.06825 [cs.CL]`.

Khattab, Omar, Keshav Santhanam, et al. (2022). "Demonstrate-Search-Predict: Composing Retrieval and Language Models for Knowledge-Intensive NLP". In: *arXiv preprint arXiv:2212.14024*.

Khattab, Omar, Arnav Singhvi, et al. (2023). "DSPy: Compiling Declarative Language Model Calls into Self-Improving Pipelines". In: *arXiv preprint arXiv:2310.03714*.

Kojima, Takeshi et al. (2023). *Large Language Models are Zero-Shot Reasoners*. arXiv: `2205.11916 [cs.CL]`.

Kwon, Woosuk et al. (2023). "Efficient Memory Management for Large Language Model Serving with PagedAttention". In: *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.

Madan, Kanika et al. (2023). *Learning GFlowNets from partial episodes for improved convergence and stability*. arXiv: `2209.12782 [cs.LG]`.

Malkin, Nikolay et al. (2023). *Trajectory balance: Improved credit assignment in GFlowNets*. arXiv: `2201.13259 [cs.LG]`.

Ouyang, Long et al. (2022). *Training language models to follow instructions with human feedback*. arXiv: `2203.02155 [cs.CL]`.

Schnabel, Tobias and Jennifer Neville (2024). *Prompts As Programs: A Structure-Aware Approach to Efficient Compile-Time Prompt Optimization*. arXiv: `2404.02319 [cs.CL]`.

Schulman, John et al. (2017). *Proximal Policy Optimization Algorithms*. arXiv: `1707.06347 [cs.LG]`.

Shen, Max W. et al. (2023). *Towards Understanding and Improving GFlowNet Training*. arXiv: `2305.07170 [cs.LG]`.

Singhvi, Arnav et al. (2024). *DSPy Assertions: Computational Constraints for Self-Refining Language Model Pipelines*. arXiv: `2312.13382 [cs.CL]`.

Stiennon, Nisan et al. (2022). *Learning to summarize from human feedback*. arXiv: `2009.01325 [cs.CL]`.

Vemgal, Nikhil, Elaine Lau, and Doina Precup (2023). *An Empirical Study of the Effectiveness of Using a Replay Buffer on Mode Discovery in GFlowNets*. arXiv: `2307.07674 [cs.LG]`.

Wei, Jason, Maarten Bosma, et al. (2022). *Finetuned Language Models Are Zero-Shot Learners*. arXiv: `2109.01652 [cs.CL]`.

Wei, Jason, Xuezhi Wang, et al. (2023). *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models*. arXiv: `2201.11903 [cs.CL]`.

Yao, Shunyu et al. (2023). *Tree of Thoughts: Deliberate Problem Solving with Large Language Models*. arXiv: `2305.10601 [cs.CL]`.

Zhou, Hattie et al. (2022). *Teaching Algorithmic Reasoning via In-context Learning*. arXiv: `2211.09066 [cs.LG]`.

DSPy Prompts

For fine-tuning, we used the following prompt, optimized by DSPy and hand-tweaked for conciseness. We added the example inside the prompt to help the model understand the formatting required. The question, in this case about Nancy, is inserted as an example of a filled-in prompt.

```
<s> [INST]Carefully read the provided problem (‘question‘). Provide a
step-by-step explanation (‘rationale‘). Finally, concisely provide
the answer (‘answer‘) to the question after the string "####".
---

Follow the following format.

Question: ${{question}}
Rationale: ${{rationale}}
#### ${{answer}}

---

For example:

Question: John has 3 apples. He eats 1. How many apples does he have left?
Rationale: John had 3 apples. He ate 1. 3 - 1 = 2.
#### 2

---

Question: Nancy has a bag containing 22 tortilla chips. She gives
7 tortilla chips to her brother and 5 tortilla chips to her sister,
keeping the rest for herself. How many did Nancy keep for herself?[/INST]
Rationale:
```

Note that we leave the rationale outside of the instruction prompt, as we want the model to continue generating after the rationale. During evaluation, we used the following prompt to generate rationales for the test set:

```
Given the fields ‘question‘, produce the fields ‘answer‘.

---

Follow the following format.

Question: ${question}
Reasoning: Let’s think step by step in order to ${produce the answer}.
```

```
We ...
Answer: ${answer}

---

Question: {What is the capital of France?}
Reasoning: Let’s think step by step in order to
```

It is important to note that the evaluation example does not take advantage of the instruction tuning. Because we use the same prompt for both the GFN-FT model and the base model, we consider the evaluation fair, regardless of the inclusion of the instruction tag.