Entropic Distribution Matching for Supervised Fine-tuning of LLMs: Less Overfitting and Better Diversity

Ziniu Li[1,2], **Congliang Chen**[1,2], **Tian Xu**[3], **Zeyu Qin**[4], **Jiancong Xiao**[5], **Ruoyu Sun**[∗1,2], **and Zhi-Quan Luo**[1,2]

[1]The Chinese University of Hong Kong, Shenzhen
[2]Shenzhen Research Institute of Big Data
[3]Nanjing University
[4]Hong Kong University of Science and Technology
[5]University of Pennsylvania
{ziniuli,congliangchen}@link.cuhk.edu.cn, xut@lamda.nju.edu.cn,
zeyu.qin@connect.ust.hk, jcxiao@upenn.edu, {sunruoyu,luozq}@cuhk.edu.cn

Abstract

Large language models rely on Supervised Fine-Tuning (SFT) to specialize in downstream tasks. Cross Entropy (CE) loss is the de facto choice in SFT. However, CE often results in overfitting and limited output diversity due to its aggressive distribution matching strategy, which forces the model's generative distribution to closely mimic the empirical data distribution. This paper aim to address these issues by introducing the maximum entropy principle, encouraging models to resist overfitting while preserving output diversity. Specifically, we develop a new distribution matching method called GEM, which solves reverse Kullback-Leibler divergence minimization with an entropy regularizer.

For the SFT of Llama-3-8B models, GEM outperforms CE in several aspects. First, when applied to acquire general instruction-following abilities, GEM exhibits reduced overfitting, as evidenced by lower perplexity and better performance on the IFEval benchmark. Second, this advantage is also observed in domain-specific fine-tuning, where GEM continues to outperform CE in specialized math reasoning and code generation tasks. Last, we show that GEM-tuned models offer better output diversity, which helps scale up test-time compute: with the same sampling budget, they achieve performance gains of up to 10 points in math reasoning and code generation tasks, compared with CE-tuned models.[1]

1 Introduction

Large Language Models (LLMs) [39, 53, 52] are powerful generative models excelling in specialized tasks across various fields. Despite extensive pre-training, LLMs often struggle to follow instructions and answer users' queries effectively. To improve their performance in these tasks, instruction tuning [45, 60, 10], also known as Supervised Fine-Tuning (SFT) [41, 3], is employed. This process involves using high-quality labeled data (i.e., prompt-response pairs) and typically utilizes the Cross Entropy (CE) loss to maximize the likelihood of the labeled data.

SFT is the first stage of the post-training pipeline and plays a crucial role in future developments [7, 54, 34]. We expect models to generalize well by providing accurate answers and hope these

∗: Corresponding author.

[1]Code is available at https://github.com/liziniu/GEM.

38th Workshop on Fine-Tuning in Machine Learning (NeurIPS 2024).

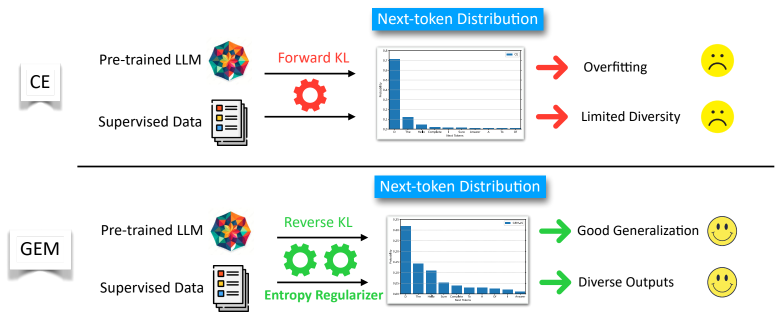


Figure 1: Illustration of the standard CE and the proposed method GEM for SFT of LLMs.

answers are diverse as well. While the importance of generalization is clear, generation diversity is also crucial, especially with the trend of scaling up test-time compute [50, 6, 64]. These emerging studies have shown that scaling up test-time compute, by selecting the optimal response from multiple generated options, can solve many complex reasoning tasks, with output diversity being a key factor in this process [59]. Additionally, many applications benefit from diverse responses. In creative writing, diverse outputs can inspire new ideas [12], and in chit-chat, users value options that suit their preferences [32]. AI interfaces like ChatGPT meet this need with features like regeneration buttons.

Unfortunately, using CE loss in SFT falls short of achieving the desired goals, because models fine-tuned with CE often suffer from overfitting [7, 23, 16] and lack of generation diversity [42, 38]. These limitations stem from the theoretical underpinnings of CE loss. In theory, optimizing CE loss amounts to minimizing the *forward* Kullback–Leibler (KL) divergence between the data distribution and the generative distribution of the LLM.[2] This process aggressively increases the likelihood of training data while overlooking other possibilities, which in turn leads to overfitting. For instance, CE-tuned models are often observed to over-memorize training data [15, 68], latch onto spurious features [7], and lose in-context learning abilities that already been acquired in the pre-training (a.k.a. alignment tax) [41, 3]. Furthermore, the aggressive update of the generative model's distribution to fit the training data leads to reduced entropy, which in turn limits output diversity. Previous research has shown that low-entropy distributions are associated with poor generalization performance [43, 14], suggesting that these issues are interrelated. To address these concerns, techniques like weight decay [53, 7] or noisy perturbations to embeddings [23] are commonly applied alongside CE loss. However, these challenges remain, highlighting the need for more principled solutions.

In this paper, we frame the fine-tuning of LLMs as a distribution matching problem, introducing the maximum entropy principle [24] to guide the process. This principle promotes the use of an entropy regularizer to avoid over-assigning high probabilities to the training data, thereby preserving output diversity, which is particularly important when working with limited data. We also propose generative distribution matching, encouraging the model to learn not only from supervision but also from its own generated errors, drawing inspiration from Generative Adversarial Networks (GANs) [17]. This approach contrasts with the passive imitation of supervised data typical in CE loss, aligning more closely with the entropy regularizer (discussed further in the main text). To implement these ideas, we develop the formulation of *reverse* KL divergence minimization with *entropy* regularization. However, this formulation is technically challenging and may require adversarial training techniques akin to those used in GANs. Our main technical contribution is the development of a new training algorithm, referred to as GEM, which addresses the above challenge and is as tractable as the CE loss. By adhering to the proposed principles, GEM favors distributions that captures key patterns in the data and enjoy high entropy; see Figure 1.

We demonstrate the effectiveness of our model by fine-tuning the Llama-3-8B pre-trained model with two types of tasks. First, using the `UltraFeedback` dataset [13], GEM achieves lower evaluation perplexity than CE and better performance on `IFEval` [70], indicating reduced overfitting and improved output diversity in creative writing. In math reasoning (`GSM8K` [11]) and code generation (`HumanEval` [8], `MBPP` [2]), GEM shows up to 7-point gains using sampling strategies like Majority Voting (MV) and Best-Of-N (BON). In the second experiment, fine-tuning on `MetaMathQA` [67] and

[2]The term *forward* KL arises from a technical distinction. We will later explore the concept of *reverse* KL. The key difference between the two lies in how the loss is defined: forward KL measures the loss over the fixed data distribution, while reverse KL defines the loss over the generative model's distribution.

`MagicCoder-OSS-Instruct` [62] for math reasoning and code generation, GEM outperforms CE by up to 10 points with MV and BON, confirming its effectiveness.

To summarize, our contributions are threefold:

- We introduce the framework of entropic distribution matching for SFT of LLMs to address the issues of overfitting and limited diversity.
- We develop a new training method GEM that can solve a particular distribution matching problem with reverse KL divergence minimization and maximum entropy regularization.
- We demonstrate that the improved generalization and diversity induced by our method can be beneficial to test-time compute.

2 Preliminary

Large Language Models (LLMs). LLMs have a large vocabulary, denoted as $[K] = \{1, 2, \ldots, K\}$ and process text by splitting it into a series of tokens $(x_1, \ldots, x_T)$, where each token $x_i \in [K]$ and T represents the sequence length. Let f be the generative distribution modeled by the language model. The notation $f(\cdot|x_1, \ldots, x_{t-1})$ specifies the categorical distribution conditioned on the context $(x_1, \ldots, x_{t-1})$. Typically, f is parameterized by a neural network, often a transformer [57], with the parameter θ. For the i-th token at time step t, its prediction probability is given by

$$f_\theta(i|x_1, \ldots x_{t-1}) = \texttt{softmax}(z_t) = \frac{\exp(z_t[i])}{\sum_{i'} \exp(z_t[i'])}.$$

where $z_t \in \mathbb{R}^K$ is the logit output from the neural network given the input $(x_1, \ldots, x_{t-1})$, and $z_t[i]$ is i-th element of z_t. This auto-regressive process specifies the joint probability of a sequence of tokens as $f_\theta(x_1, \ldots, x_T) = \prod_{t=1}^{T} f_\theta(x_t|x_1, \ldots, x_{t-1})$.

Supervised Fine-Tuning. To specialize in downstream tasks, LLM relies on Supervised Fine-Tuning (SFT) after pre-training. This process involves using a supervised dataset with high-quality prompt-response pairs $\{(x^i, y^i)\}_{i=1}^N$. The Cross Entropy (CE) loss is the de facto training objective for this purpose: $\min_\theta \sum_{i=1}^N -\log f_\theta(y^i|x^i)$. In theory, this corresponds to minimizing the *forward* KL divergence between the data distribution p and the generative distribution f_θ:

$$\min_\theta D_{\mathrm{KL}}(p, f_\theta) \iff \max_\theta \mathbb{E}_{x\sim\rho(\cdot)}\mathbb{E}_{y\sim p(\cdot|x)}[\log f_\theta(y|x)],$$

where ρ is the prompt distribution, which is usually not modeled during the SFT stage. Thus, the distribution ρ can be treated as a constant and we omit it when the context is clear. In practice, many questions can correspond to multiple valid answers (either in different forms or based on different reasoning), but it is nearly impossible to collect a comprehensive dataset that encompasses all possibilities. As a result, the empirical data tends to be limited in size and often exhibits a narrower distribution than desired. In such scenarios, the CE loss function aggressively maximizes the likelihood of the available empirical data and overlooks other possibilities. However, this approach can lead to poor generation diversity and overfitting, as previously noted.

3 Entropic Distribution Matching

In this paper, we explore principled approaches for SFT, presenting two principles. The first addresses overfitting and limited output diversity, inspired by neuroscience, particularly synaptic plasticity. Homeostatic plasticity highlights the need for balanced learning [56, 55], where overly strengthened neural connections lead to rigidity, similar to how assigning high probabilities to tokens causes over-memorization, limiting adaptability and generalization. Based on these insights, we propose:

- Principle 1: The model should assign higher probabilities to the observed data while preventing over-memorization.

The above principle can be implemented by adding an entropy regularizer. Our second principle advocates a *generative* approach to distribution matching, where models learn from their own generated data and mistakes, rather than just imitating supervised data. Unlike CE loss, which passively mimics labels, this approach mirrors how children learn effectively through exploration and adjusting based on mistakes [47, 19]. Generative models [17, 21] similarly refine their outputs through self-correction. In summary, we propose:

- Principle 2: The distribution matching approach should be "generative", meaning the model learns from both ground truth supervision and its own generated.

3.1 Proposed Formulation: Reserve KL with Entropy Regularization

To implement the two principles outlined above, we propose studying the formulation of *reverse* KL divergence minimization with maximum entropy regularization. The objective is defined as follows:

$$\max_f \mathbb{E}_x \Big\{ \underbrace{\mathbb{E}_{y\sim f(\cdot|x)}\left[\log p(y|x)\right] - \mathbb{E}_{y\sim f(\cdot|x)}[\log f(y|x)]}_{=-D_{\mathrm{KL}}(f,p)} + \gamma \cdot \underbrace{\mathbb{E}_{y\sim f(\cdot|x)}[-\log f(y|x)]}_{\mathcal{H}(f)} \Big\}. \quad (1)$$

The first term corresponds to the *reverse* KL divergence between the target distribution p and the model distribution f. This term supports Principle 2 by encouraging the model to learn from its generated data samples (as reflected in the expectation $\mathbb{E}_{y\sim f(\cdot|x)}$), similar to GANs [18].This contrasts with the passive learning in CE, where the expectation is taken over a static data distribution. The second term, entropy regularization, aligns with Principle 1 by preventing over-memorization. From a Bayesian perspective, this means placing a uniform distribution belief when learning from data, so it ensures that the probabilities for labeled data do not become excessively high. In addition, entropy regularization brings another benefit: the output diversity can be improved. This means that the model is aware of other possible options, which is very important for scaling-up test-time compute [50, 6]. We note that adding entropy regularization to the CE loss supports Principle 1 but not Principle 2; its limitations are discussed in Appendix D, and we will empirically show that it is inferior to the proposed approach in Section 4.

While the objective defined in Equation (1) appears promising, it presents significant challenges in practice. The main challenge is that we only have access to empirical data from the distribution p, not its full probability density function, making the reverse KL term impossible to compute directly. Additionally, calculating the expectation of the reverse KL across the model's generative distribution is not easy. This paper contributes a new algorithm to address these challenges.

3.2 Proposed Algorithm: GEM

In this section, we present a practical algorithm for solving the optimization problem of reverse KL with entropy regularization. Our approach is inspired by Relativistic GANs [25], where an auxiliary distribution q is introduced for distribution matching, and relative-pair comparisons are incorporated into the training objective. Specifically, our formulation is that:

$$\max_f \quad \mathcal{L}_q(f) \triangleq \mathbb{E}_x \mathbb{E}_{y^{\mathtt{real}}\sim p(\cdot|x)} \mathbb{E}_{y^{\mathtt{gene}}\sim q(\cdot|x)} \left[h\left(\log f(y^{\mathtt{real}}|x) - \log f(y^{\mathtt{gene}}|x) \right) \right] \quad (2)$$

$$\text{s.t.} \quad q = \operatorname*{argmax}_{\pi} \mathbb{E}_x \mathbb{E}_{y\sim\pi(\cdot|x)} \left[\log f(y|x)\right] + 1/\beta \cdot \mathcal{H}(\pi(\cdot|x)) = \mathtt{softmax}(1/\beta * \log f). \quad (3)$$

Here we use the subscript `real` to denote the supervised data and `gene` to denote the model-generated data for clarity. In addition, h is a monotonically increasing function (e.g., a linear function). Moreover, q is an artificially introduced distribution that will discarded after training. To interpret the formulation, we optimize f such that $\log f$ is higher for real data and lower for generated data. In this context, $\log f$ can be understood as the "energy" in an energy-based model [30] (or the reward in inverse reinforcement learning [37, 21]). Simultaneously, we update the distribution q to maximize the "energy" induced by $\log f$, thereby aligning it with the data distribution. During the optimization of q, an entropy regularizer is applied, which in turns guarantees the desired result.

Proposition 1. *Assume that h is a linear function, then $\mathcal{L}_q(f)$ has a unique stationary point, and this stationary point (with $\beta = 1/(\gamma+1) > 0$) corresponds to the optimal solution of Problem (1).*

Proposition 1 implies that solving the proposed problem in Equations (2) and (3) provides the optimal solution of reverse KL with entropy regularization in Equation (1). In practice, we can parameterize f using a transformer and optimize the parameters with gradient ascent. We outline such a training procedure in Algorithm 1, referring to this approach as GEM, which stands for <u>G</u>enerative and <u>E</u>ntropy-regularized <u>M</u>atching of distributions. We point out that $h(u) = \mathtt{logsigmoid}(u)$ as in [26] can also work in practice. We also note that Proposition 1 relies on $\beta > 0$, meaning that GEM cannot solve the pure reverse KL minimization problem.

We highlight two key computational advantages of GEM for generative distribution matching in LLMs. First, only a single model, f, is optimized using a closed-form solution for q, eliminating the need for adversarial training, reducing overhead, and simplifying hyperparameter tuning. Second, the loss function and gradients are computed with the *exact* expectation $\mathbb{E}_{y^{\mathtt{gene}}\sim q(\cdot|x)}[\cdot]$, ensuring stable training by lowering gradient variance. These advantages distinguish our approach from GAN-style methods, which require two models and rely on inexact stochastic gradients.

Algorithm 1 GEM

Input: Dataset $\mathcal{D} = \{(x_i, y_i^{\texttt{real}})\}$

1: **for** iteration $k = 1, \ldots,$ **do**

2: Set $q_k = \texttt{softmax}(1/\beta * \log f_{\theta_k})$

3: Compute loss $\mathcal{L}_q(f_\theta) = \sum_i \sum_{y^{\texttt{gene}}} q(y^{\texttt{gene}}|x_i) \cdot h\left([\log f_\theta(y_i^{\texttt{real}}|x_i) - \log f_\theta(y^{\texttt{gene}}|x_i)]\right)$

4: Update $\theta_{k+1} = \theta_k + \eta \cdot \nabla_\theta \mathcal{L}_q(f_\theta) \mid_{\theta=\theta_k}$

Output: Generative model f_θ

Extensions to Sequential Data. In the above part, we have derived the algorithm for the case y is non-sequential. We note that optimization in the sequential case could be highly difficult. With a little abuse of notations, let $y = (y_1, \ldots, y_T) \triangleq y_{1:T}$. Note that the prompt x should also be sequential in general, but this does not affect our discussion as it serves the input to the conditional distribution. Now, we can extend the formulation in Equations (2) and (3) to the following:

$$\begin{aligned} \max_f \quad & \mathbb{E}_x \mathbb{E}_{y_{1:T}^{\texttt{real}} \sim p(\cdot|x)} \mathbb{E}_{y_{1:T}^{\texttt{gene}} \sim q(\cdot|x)} \left[h\left(\log f(y_{1:T}^{\texttt{real}}|x) - \log f(y_{1:T}^{\texttt{gene}}|x) \right) \right] \\ \text{s.t.} \quad & q = \operatorname*{argmax}_\pi \mathbb{E}_x \mathbb{E}_{y_{1:T} \sim \pi(\cdot|x)} \left[\log f(y_{1:T}|x) \right] + 1/\beta \cdot \mathcal{H}(\pi(\cdot|x)) \end{aligned}$$

Here, we encounter a challenge: the joint distribution of $y_{1:T}$, as a cascaded categorical distribution, is quite complicated. This results in the expectation $\mathbb{E}_{y_{1:T}^{\texttt{gene}}}[\cdot]$ cannot be easily calculated as before. While Monte Carlo estimation—drawing samples to approximate the gradient—might seem like a viable solution, we found it does not work in experiments. We believe the main reason is that the sample space is huge, and the pre-trained distribution f is quite different from the data distribution p that we aim to learn.[3] As a result, when we use stochastic sampling to estimate the gradient, it does not provide effective feedback.

To deal with the above challenges, we propose decomposing the multi-step sequential optimization problem into multiple single-step optimization problems and solve each efficiently. This is inspired by the data distribution "reset" trick introduced by [46] in imitation learning, where the teacher first demonstrates a few actions, and the student completes the reset. For our problem, we restrict the distribution matching to the case that the prefix samples up to time step t are drawn from the data distribution p and solves the optimization problem at the t-th time step as before. Its mathematical formulation is given below:

$$\begin{aligned} \max_f \mathcal{L}_q^{\texttt{seq}}(f) &= \mathbb{E}_x \left\{ \sum_{t=1}^T \mathbb{E}_{y_{1:t-1}^{\texttt{real}} \sim p(\cdot|x)} \mathbb{E}_{y_t^{\texttt{real}} \sim p(\cdot|x, y_{1:t-1}^{\texttt{real}})} \mathbb{E}_{y_t^{\texttt{gene}} \sim q(\cdot|x, y_{1:t-1}^{\texttt{real}})} [\Delta] \right\} \\ \text{where} \quad \Delta &= \left[h\left(\log f(y_t^{\texttt{real}}|x, y_{1:t-1}^{\texttt{real}}) - \log f(y_t^{\texttt{gene}}|x, y_{1:t-1}^{\texttt{real}}) \right) \right], \end{aligned} \tag{4}$$

The main advantage of this formulation is that for each sub-problem, we still have access to the conditional distribution, allowing the previously discussed computational advantages to remain applicable. The same idea applies to the training of distribution q. We outline the proposed procedure in Algorithm 2 and provide its PyTorch implementation in Appendix B. We note that the implementation of our method requires almost the same GPU memory consumption and compute time as optimizing the CE loss. It is important to note that our proposed solution serves as an approximation to Equation (4). While the exact gap between the approximation and the true solution is unknown, our practical algorithm has already demonstrated promising results.

4 Experiments

In this section, we present our numerical results for fine-tuning the pre-trained Llama-3-8B model to demonstrate the effectiveness of the proposed method. The main results are reported, with additional results in Appendix F and experiment details in Appendix E.

4.1 General-Purpose Fine-tuning

Set-up. We first develop an LLM that is capable of following instructions for various prompts. To this end, we utilize the `UltraFeedback` dataset [13]. This dataset contains prompts from instruction datasets like Evol-Instruct and UltraChat, and responses generated by models such as GPT-4 and

[3] Specifically, pre-trained models cannot generate the `EOS` (end-of-sentence) token properly, resulting in repetitive sequences, even with infinite length. But the supervised data has an `EOS` token and finite length.

Llama-2-7B/13B/70B-Chat. Each data point comprises two responses: one selected as the preferred option and the other as the rejected option, with the selection made by GPT-4. In our study, we use the preferred response for SFT, a practice commonly adopted in previous research [41, 3]. Following [67, 34, 13], we set the learning rate to 2×10^{-5}, employing a cosine learning rate decay schedule, and use a macro batch size of 128. The maximum sequence length, encompassing both the prompt and response, is set to 2,048 tokens. Models are trained for three epochs.

We implement the proposed GEM method with $\beta = 0.7$. As discussed, GEM has two variations: GEM-LS (GEM with `log-sigmoid`), and GE-Linear, each depending on the choice of the function h. Our primary baseline is the standard CE loss. Additionally, we explore a variant incorporating a weight decay of 0.1, which has been commonly used in previous studies [41, 3]. We refer to this approach as CE + WD. We also implement a method called CE + Entropy, which adds an entropy regularization term of 0.1 to the CE loss. This method aligns with the proposed Principle 1 but not Principle 2 (see Appendix D for more discussion). The NEFT method [23], which perturbs the input embedding with random noise in fine-tuning to mitigate overfitting, has also been implemented.

Instruction-Following. We first examine the model's learned ability in terms of instruction-following on the `IFEval` benchmark [70]. The model's performance on this benchmark provides insight into potential overfitting. There are four evaluation criteria: prompt-level strict accuracy, instruction-level strict accuracy, prompt-level loose accuracy, and instruction-level loose accuracy. For all metrics, a higher value indicates better performance.

Table 1: Performance of instruction-following on the benchmark `IFEval` [70]. For all metrics, a higher value means a better instruction following ability. The best results are shown in bold, with the second-best underlined.

Method	Instruction-Following			
	Strict Accuracy (Prompt Level)	Strict Accuracy (Instruction Level)	Loose Accuracy (Prompt Level)	Loose Accuracy (Instruction Level)
CE	36.23	46.76	40.85	50.96
CE+WD	**37.89**	47.48	**42.88**	<u>52.52</u>
CE+Entropy	36.78	<u>47.60</u>	40.66	51.08
NEFT	36.23	46.40	40.11	50.48
GEM-Linear	37.34	**48.20**	41.96	**52.64**
GEM-LS	<u>37.52</u>	<u>47.60</u>	<u>42.14</u>	52.04

We evaluate the trained models using greedy decoding and present the results in Table 1. We observe that CE underperforms compared with regularization-based methods, suggesting that CE suffers from overfitting. It is important to note that this overfitting is not due to over-optimization, as performance continues to improve over three training epochs for CE (36.15 in epoch 1, 41.45 in epoch 2, and 43.70 in epoch 3). For NEFT, we do not observe clear advantages by injecting noise in training for this task. On average across the four criteria, GEM-Linear and GEM-LS improve by 1.4 points (3.2% relative) and 1.1 points (2.5% relative) compared with CE.

In addition to the above evaluation, we also observe that GEM mitigates overfitting in the evaluation perplexity: GEM-LS and GEM-Linear achieve lower perplexity (around 3.16) than CE (3.48). Furthermore, GEM incurs less alignment tax (i.e., the loss of in-context learning abilities). Please refer to Appendix F for these results.

Creative Writing. We continue to assess models' output diversity in creative writing tasks: poem writing and story writing. For poems, we use prompts from the `poetry`[4] dataset, which includes 573 poems on themes such as love, and mythology. For stories, we design 500 prompts based on the ROC story dataset [35]. In both cases, we prompt the models to write a poem or story titled "[X]" with no more than 200 words, where [X] is a title from the respective dataset. Following [28], we use three criteria to evaluate diversity: 1) N-gram diversity: the proportion of distinct n-grams in a single response (intra-diversity); 2) Self-BLEU diversity: calculated as 100 minus the Self-BLEU score (inter-diversity), where one response is treated as a reference among multiple generated responses; 3) Sentence-BERT diversity: the cosine dissimilarity between pairs of responses in the embedding space. All criteria range from 0 to 100 (with Sentence-BERT diversity scaled by multiplying by 100), and higher values indicate greater diversity.

[4] `https://huggingface.co/datasets/merve/poetry`

Table 2: Evaluation of generation diversity in creative tasks of poem writing and story writing. For all criterion, a higher value indicates greater diversity.

Method	Poem Writing			Story Writing		
	N-gram	Self-BLEU	Sentence-BERT	N-gram	Self-BLEU	Sentence-BERT
CE	48.50	72.50	21.79	48.74	72.77	21.94
CE+WD	48.58	71.29	21.80	48.85	71.73	21.79
CE+Entropy	53.74	75.82	23.80	53.86	76.11	23.94
NEFT	49.87	75.04	23.44	50.00	75.32	23.36
GEM-Linear	56.50	**76.73**	**24.73**	56.69	**76.83**	**24.82**
GEM-LS	**56.55**	76.31	24.63	**56.82**	76.61	24.68

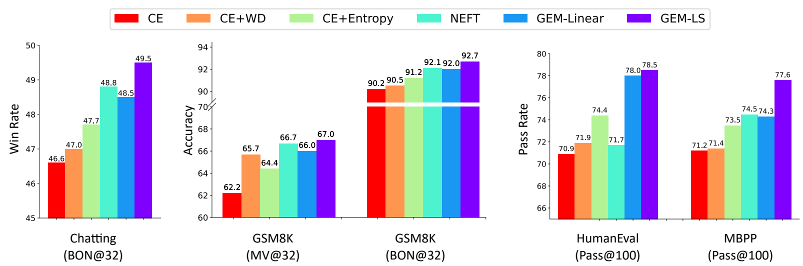


Figure 2: Performance of using advanced generation strategies such as best-of-n and majority voting in chatting (left), math reasoning (middle) and code generation (right) tasks.

To calculate these metrics, we ask the trained models to generate 16 samples using the decoding configuration `temperature=1`, `top_k=50`, and `top_p=0.9`. The evaluation results are presented in Table 2. In this task, we note that weight decay does not improve generation diversity, although it has shown effectiveness in mitigating overfitting in previous examples. On the other hand, entropy regularization, implemented to support Principle 2, brings the benefit of output diversity. NEFT also improves output diversity, consistent with [23]. Overall, GEM significantly improves output diversity compared with the baselines.

Chatting, Math Reasoning, and Code Generation. In this part, we show that improved generation diversity offers benefits beyond creative writing tasks. Specifically, diverse generation, when quality is ensured, is advantageous when using advanced generation methods such as Best-Of-N (BON) or Majority-Voting (MV) [59] to find better solutions. This is inline with recent advances in scaling up test-time compute [6, 50].

Specifically, we conduct three experiments in chatting, math reasoning and code generation below. Unlike before, we use a lower temperature of 0.6 (the default value for Llama models) to enhance response quality. The overall performance is displayed in Figure 2 with detailed results in Appendix F. Before analyzing the detailed results, we note that the good samples generated in this part could be further distilled into the model for improving zero-shot performance; see [48].

For chatting, we assess the model's ability to generate human-preferred responses. We prompt the trained models to answer 805 questions from the `AlpacaEval` dataset [33]. For each question, the model generates 32 responses and a reward model is then used to select the best responses. We employ the reward model `FsfairX-LLaMA3-RM-v0.1`[5], which has top performance on `RewardBench` [29]. Since the reward value itself does not mean anything, we choose the win rate as a metric. In particular, we estimate the win rate over GPT-4's generated response by the Bradley–Terry model. From Figure 2, we observe that GEM-LS can achieve about 3 points improvement in the win rate compared with CE. Among the baselines, NEFT demonstrates strong performance, partially due to its longer responses, as noted in [23].

For math reasoning, we evaluate performance on the `GSM8K` [11] benchmark, which contains 1,319 test questions. We prompt LLMs with chain-of-thought [61] to generate 32 responses for each question. We assess answer accuracy using both Majority-Voting (MV) [59] and Best-Of-N (BON) methods. Compared with CE, GEM-LS shows improvements of up to 4.8 points (7.7% relative) with MV and 2.5 points (2.8% relative) with BON.

[5] `https://huggingface.co/sfairXC/FsfairX-LLaMA3-RM-v0.1`

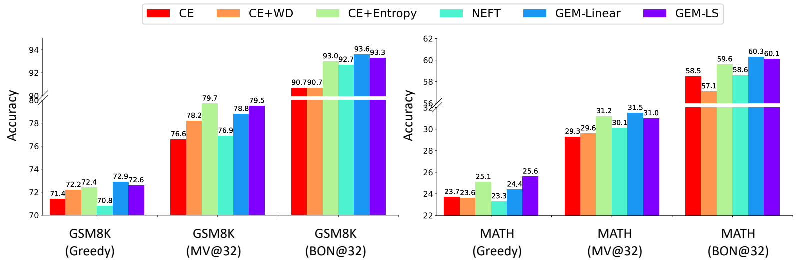


Figure 3: Performance on `GSM8K` (left) and `MATH` (right) when fine-tuning Llama-3-8B with the `MetaMathQA` dataset.

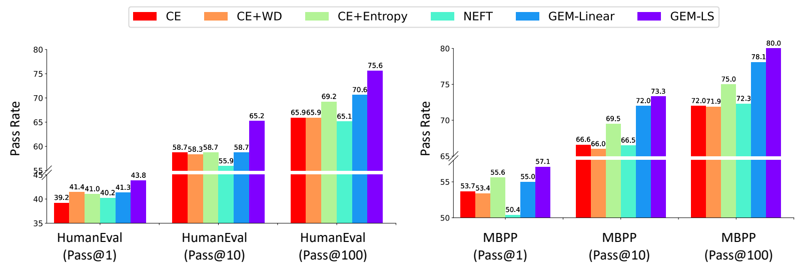


Figure 4: Performance on `HumanEval` (left) and `MBPP` (right) when fine-tuning Llama-3-8B with the `MagiCoder-OSS-Instruct` dataset.

For code generation, we consider two benchmarks: `HumanEval` [8] and `MBPP` [2]. In these scenarios, the trained models are asked to generate Python codes, and the executor judges their correctness. The common evaluation metric is the pass rate. We ask the trained models to generate 200 samples to estimate the pass@100. The generation configuration is the same as for the chatting task. We find that weight decay does not show significant improvement over CE, while GEM-LS can achieve up to a 7.6-point (10.7% relative) improvement over CE on `HumanEval` and a 6.4-point (9.0% relative) improvement on `MBPP` for pass@100.

4.2 Domain-specific Fine-tuning

In this section, we conduct experiments with domain-specific datasets. For math reasoning, we use the dataset `MetaMathQA` [67]. For code generation, we use the dataset `Magicoder-OSS-Instruct` [62]. The experiment setup, including training details and hyperparameters, is the same as before, and the specifics are provided in the Appendix.

Math Reasoning. We evaluate two benchmarks: `GSM8K` and `MATH` [20], using Majority Voting (MV@32), Best-Of-N (BON@32), and greedy decoding. Please see Figure 3. We find that weight decay works well on `GSM8K` but shows no clear gain on `MATH`, while NEFT offers no improvement. In contrast, GEM-LS outperforms CE on `GSM8K` by 1.2, 2.9, and 2.6 points for greedy decoding, MV@32, and BON@32, respectively. For `MATH`, GEM-LS improves by 1.9, 1.7, and 1.6 points. These results suggest entropy regularization reduces overfitting and enhances diversity.

Code Generation. We report the pass rate over 1, 10, 100 on `HumanEval` and `MBPP` benchmarks, in Figure 4. Pass@100 on `HumanEval` drops compared to previous results, while all methods improve on `MBPP`. Weight decay and NEFT show inconsistent gains, but entropy regularization consistently improves performance. GEM-LS significantly outperforms CE, with improvements on `HumanEval` by 4.6 (Pass@1), 6.5 (Pass@10), and 9.7 points (Pass@100). For `MBPP`, GEM-LS gains 3.4 (Pass@1), 6.8 (Pass@10), and 8.0 points (Pass@100).

5 Conclusion

In this paper, we propose an alternative method for the SFT of LLMs to tackle the challenges of overfitting and limited generation diversity, which are often caused by the aggressive updates of the CE loss and limited data. We demonstrate the effectiveness of combining generative distribution matching with entropy regularization. We note that the improved diversity also boosts performance in downstream tasks when advanced generation methods, such as the best-of-n sampling, are used. Overall, our results indicate that the proposed method is well-suited for generative models. Future work is noted in Appendix D.1.

References

[1] Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009.

[2] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

[3] Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, et al. Training a helpful and harmless assistant with reinforcement learning from human feedback. *arXiv preprint arXiv:2204.05862*, 2022.

[4] Emmanuel Bengio, Moksh Jain, Maksym Korablyov, Doina Precup, and Yoshua Bengio. Flow network based generative models for non-iterative diverse candidate generation. *Advances in Neural Information Processing Systems*, 34:27381–27394, 2021.

[5] Andrew Brock. Large scale gan training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*, 2018.

[6] Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*, 2024.

[7] Collin Burns, Pavel Izmailov, Jan Hendrik Kirchner, Bowen Baker, Leo Gao, Leopold Aschenbrenner, Yining Chen, Adrien Ecoffet, Manas Joglekar, Jan Leike, et al. Weak-to-strong generalization: Eliciting strong capabilities with weak supervision. *arXiv preprint arXiv:2312.09390*, 2023.

[8] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[9] Zixiang Chen, Yihe Deng, Huizhuo Yuan, Kaixuan Ji, and Quanquan Gu. Self-play fine-tuning converts weak language models to strong language models. *arXiv preprint arXiv:2401.01335*, 2024.

[10] Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. Scaling instruction-finetuned language models. *Journal of Machine Learning Research*, 25(70):1–53, 2024.

[11] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

[12] Simon Colton and Geraint A Wiggins. Computational creativity: The final frontier? In *ECAI 2012*, pp. 21–26. IOS Press, 2012.

[13] Ganqu Cui, Lifan Yuan, Ning Ding, Guanming Yao, Bingxiang He, Wei Zhu, Yuan Ni, Guotong Xie, Ruobing Xie, Yankai Lin, et al. Ultrafeedback: Boosting language models with scaled ai feedback. In *Forty-first International Conference on Machine Learning*, 2024.

[14] Abhimanyu Dubey, Otkrist Gupta, Ramesh Raskar, and Nikhil Naik. Maximum-entropy fine grained classification. *Advances in neural information processing systems*, 31, 2018.

[15] Yubin Ge, Devamanyu Hazarika, Yang Liu, and Mahdi Namazifar. Supervised fine-tuning of large language models on human demonstrations through the lens of memorization. In *NeurIPS 2023 Workshop on Instruction Tuning and Instruction Following*, 2023.

[16] Zorik Gekhman, Gal Yona, Roee Aharoni, Matan Eyal, Amir Feder, Roi Reichart, and Jonathan Herzig. Does fine-tuning llms on new knowledge encourage hallucinations? *arXiv preprint arXiv:2405.05904*, 2024.

[17] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[18] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[19] Hyowon Gweon, Hannah Pelton, Jaclyn A Konopka, and Laura E Schulz. Sins of omission: Children selectively explore when teachers are under-informative. *Cognition*, 132(3):335–341, 2014.

[20] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

[21] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems 29*, pp. 4565–4573, 2016.

[22] Edward J Hu, Moksh Jain, Eric Elmoznino, Younesse Kaddar, Guillaume Lajoie, Yoshua Bengio, and Nikolay Malkin. Amortizing intractable inference in large language models. *arXiv preprint arXiv:2310.04363*, 2023.

[23] Neel Jain, Ping-yeh Chiang, Yuxin Wen, John Kirchenbauer, Hong-Min Chu, Gowthami Somepalli, Brian R Bartoldson, Bhavya Kailkhura, Avi Schwarzschild, Aniruddha Saha, et al. Neftune: Noisy embeddings improve instruction finetuning. *arXiv preprint arXiv:2310.05914*, 2023.

[24] Edwin T Jaynes. On the rationale of maximum-entropy methods. *Proceedings of the IEEE*, 70 (9):939–952, 1982.

[25] Alexia Jolicoeur-Martineau. The relativistic discriminator: a key element missing from standard GAN. In *Proceedings of the 7th International Conference on Learning Representations*, 2019.

[26] Alexia Jolicoeur-Martineau. On relativistic f-divergences. In *International Conference on Machine Learning*, pp. 4931–4939. PMLR, 2020.

[27] Liyiming Ke, Matt Barnes, Wen Sun, Gilwoo Lee, Sanjiban Choudhury, and Siddhartha S. Srinivasa. Imitation learning as f-divergence minimization. *arXiv*, 1905.12888, 2019.

[28] Robert Kirk, Ishita Mediratta, Christoforos Nalmpantis, Jelena Luketina, Eric Hambro, Edward Grefenstette, and Roberta Raileanu. Understanding the effects of rlhf on llm generalisation and diversity. *arXiv preprint arXiv:2310.06452*, 2023.

[29] Nathan Lambert, Valentina Pyatkin, Jacob Morrison, LJ Miranda, Bill Yuchen Lin, Khyathi Chandu, Nouha Dziri, Sachin Kumar, Tom Zick, Yejin Choi, et al. Rewardbench: Evaluating reward models for language modeling. *arXiv preprint arXiv:2403.13787*, 2024.

[30] Yann LeCun, Sumit Chopra, Raia Hadsell, M Ranzato, Fujie Huang, et al. A tutorial on energy-based learning. *Predicting structured data*, 1(0), 2006.

[31] Jiaxiang Li, Siliang Zeng, Hoi-To Wai, Chenliang Li, Alfredo Garcia, and Mingyi Hong. Getting more juice out of the sft data: Reward learning from human demonstration improves sft for llm alignment. *arXiv preprint arXiv:2405.17888*, 2024.

[32] Jiwei Li, Michel Galley, Chris Brockett, Jianfeng Gao, and Bill Dolan. A diversity-promoting objective function for neural conversation models. *arXiv preprint arXiv:1510.03055*, 2015.

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

[34] Wei Liu, Weihao Zeng, Keqing He, Yong Jiang, and Junxian He. What makes good data for alignment? a comprehensive study of automatic data selection in instruction tuning. *arXiv preprint arXiv:2312.15685*, 2023.

[35] Nasrin Mostafazadeh, Nathanael Chambers, Xiaodong He, Devi Parikh, Dhruv Batra, Lucy Vanderwende, Pushmeet Kohli, and James Allen. A corpus and cloze evaluation for deeper understanding of commonsense stories. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 839–849, 2016.

[36] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[37] Andrew Y. Ng and Stuart J. Russell. Algorithms for inverse reinforcement learning. In *Proceedings of the 17th International Conference on Machine Learning*, pp. 663–670, 2000.

[38] Laura O'Mahony, Leo Grinsztajn, Hailey Schoelkopf, and Stella Biderman. Attributing mode collapse in the fine-tuning of large language models. In *ICLR 2024 Workshop on Mathematical and Empirical Understanding of Foundation Models*, 2024.

[39] OpenAI. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[40] Takayuki Osa, Joni Pajarinen, Gerhard Neumann, J Andrew Bagnell, Pieter Abbeel, Jan Peters, et al. An algorithmic perspective on imitation learning. *Foundations and Trends® in Robotics*, 7(1-2):1–179, 2018.

[41] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems 35*, pp. 27730–27744, 2022.

[42] Vishakh Padmakumar and He He. Does writing with language models reduce content diversity? *arXiv preprint arXiv:2309.05196*, 2023.

[43] Gabriel Pereyra, George Tucker, Jan Chorowski, Łukasz Kaiser, and Geoffrey Hinton. Regularizing neural networks by penalizing confident output distributions. *arXiv preprint arXiv:1701.06548*, 2017.

[44] Dean Pomerleau. Efficient training of artificial neural networks for autonomous navigation. *Neural Computation*, 3(1):88–97, 1991.

[45] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67, 2020.

[46] Stéphane Ross, Geoffrey J. Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics*, pp. 627–635, 2011.

[47] Laura E Schulz and Elizabeth Baraff Bonawitz. Serious fun: preschoolers engage in more exploratory play when evidence is confounded. *Developmental psychology*, 43(4):1045, 2007.

[48] Pier Giuseppe Sessa, Robert Dadashi, Léonard Hussenot, Johan Ferret, Nino Vieillard, Alexandre Ramé, Bobak Shariari, Sarah Perrin, Abe Friesen, Geoffrey Cideron, et al. Bond: Aligning llms with best-of-n distillation. *arXiv preprint arXiv:2407.14622*, 2024.

[49] Ilia Shumailov, Zakhar Shumaylov, Yiren Zhao, Yarin Gal, Nicolas Papernot, and Ross Anderson. The curse of recursion: Training on generated data makes models forget. *arXiv preprint arXiv:2305.17493*, 2023.

[50] Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

[51] Hao Sun. Supervised fine-tuning as inverse reinforcement learning. *arXiv preprint arXiv:2403.12017*, 2024.

[52] Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, et al. Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295*, 2024.

[53] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[54] Lewis Tunstall, Edward Beeching, Nathan Lambert, Nazneen Rajani, Kashif Rasul, Younes Belkada, Shengyi Huang, Leandro von Werra, Clémentine Fourrier, Nathan Habib, et al. Zephyr: Direct distillation of lm alignment. *arXiv preprint arXiv:2310.16944*, 2023.

[55] Gina Turrigiano. Homeostatic synaptic plasticity: local and global mechanisms for stabilizing neuronal function. *Cold Spring Harbor perspectives in biology*, 4(1):a005736, 2012.

[56] Gina G Turrigiano. The self-tuning neuron: synaptic scaling of excitatory synapses. *Cell*, 135 (3):422–435, 2008.

[57] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems 30*, pp. 5998–6008, 2017.

[58] Nino Vieillard, Tadashi Kozuno, Bruno Scherrer, Olivier Pietquin, Rémi Munos, and Matthieu Geist. Leverage the average: an analysis of kl regularization in reinforcement learning. In *Advances in Neural Information Processing Systems 33*, pp. 12163–12174, 2020.

[59] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *Proceedings of the 11st International Conference on Learning Representations*, 2023.

[60] Jason Wei, Maarten Bosma, Vincent Y Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. Finetuned language models are zero-shot learners. *arXiv preprint arXiv:2109.01652*, 2021.

[61] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

[62] Yuxiang Wei, Zhe Wang, Jiawei Liu, Yifeng Ding, and Lingming Zhang. Magicoder: Empowering code generation with oss-instruct. In *Forty-first International Conference on Machine Learning*, 2024.

[63] Ting Wu, Xuefeng Li, and Pengfei Liu. Progress or regress? self-improvement reversal in post-training. *arXiv preprint arXiv:2407.05013*, 2024.

[64] Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. An empirical analysis of compute-optimal inference for problem-solving with language models. *arXiv preprint arXiv:2408.00724*, 2024.

[65] Jiancong Xiao, Ziniu Li, Xingyu Xie, Emily Getzen, Cong Fang, Qi Long, and Weijie J Su. On the algorithmic bias of aligning large language models with rlhf: Preference collapse and matching regularization. *arXiv preprint arXiv:2405.16455*, 2024.

[66] Tian Xu, Ziniu Li, and Yang Yu. Error bounds of imitating policies and environments. In *Advances in Neural Information Processing Systems 33*, pp. 15737–15749, 2020.

[67] Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. Metamath: Bootstrap your own mathematical questions for large language models. *arXiv preprint arXiv:2309.12284*, 2023.

[68] Shenglai Zeng, Yaxin Li, Jie Ren, Yiding Liu, Han Xu, Pengfei He, Yue Xing, Shuaiqiang Wang, Jiliang Tang, and Dawei Yin. Exploring memorization in fine-tuned language models. *arXiv preprint arXiv:2310.06714*, 2023.

[69] Biao Zhang, Zhongtao Liu, Colin Cherry, and Orhan Firat. When scaling meets llm finetuning: The effect of data, model and finetuning method. *arXiv preprint arXiv:2402.17193*, 2024.

[70] Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*, 2023.

A Related Work

Supervised Fine-tuning. SFT is the first stage of the post-training pipeline and plays an important role in subsequent developments. As mentioned in the introduction, using CE loss during the SFT stage often leads to overfitting and reduced output diversity. To address this, there is a line of research in scaling up SFT data (see, e.g., [67, 62, 69]), which, while effective, increases computational burden. Our work aims to develop training methods that more effectively leverage supervised data to mitigate overfitting and to enhance output diversity.

Entropy Regularization. Dubey et al. [14] proposed that achieving zero CE loss is not essential for high accuracy. Instead, they suggested that a conditional probability distribution where the argmax corresponds to the correct class is sufficient. This concept motivates our use of entropy regularization, which allows for assigning probabilities to alternative options beyond the observed data. Prior to our work, Pereyra et al. [43] also explored entropy regularization in the context of neural network training. Their method closely resembles the CE with entropy regularization that we investigate in this paper, and they found that penalizing confident outputs improves generalization. It is important to note that Pereyra et al. [43] focused on image classification tasks, while our focus is on text generation where data is sequential in nature and is more challenging. In the context of LLMs, Hu et al. [22] explored the maximum entropy regularization by using GFlowNet [4], but their methods require a reward function rather than supervised data.

Distribution Matching. Distribution matching forms the foundation of statistical machine learning [36]. The seminal work GAN [17] introduced the concept of generative distribution matching in deep learning. To address sequential data, GAIL [21] made a significant advancement. A major challenge in this field is scalability [5], as these methods typically require optimizing both a generator and a discriminator through adversarial training, which is notoriously difficult and computationally expensive. In this paper, we contribute a stable training algorithm for SFT of LLMs.

Closely related to our work, recent studies such as [9, 31] explored improving CE-trained models using techniques like self-play. However, our approach differs in two key ways. First, we focus on addressing the limitations of CE loss by designing methods that directly improve pre-trained models, whereas their methods are applied post-SFT. Second, we introduce the maximum entropy principle into distribution matching, while their work examines the standard distribution matching framework.

Our work also relates to imitation learning (IL) [1, 40], where a learner makes decisions based on expert demonstrations. In fact, SFT can be reframed as IL with deterministic transitions [51, 31]. Specifically, the cross-entropy loss corresponds to behavior cloning [44] in IL. Our framework is closely aligned with the generative adversarial imitation learning approach in [21], which usually outperforms behavior cloning [27, 66]. A key aspect of this framework is correcting mistakes by rolling out trajectories. As discussed in Section 3, our proposed algorithm also supports this idea.

B Implementation of GEM

Algorithm 2 GEM for Sequential Data

Input: Dataset $\mathcal{D} = \{(x_i, y_1, \ldots, y_T)\}$

1: Initialize $\widetilde{\mathcal{D}} = \emptyset$

2: **for** sample index i **do** ▷ *"Reset" data distribution*

3: **for** timestep index $t = 1, \ldots, T$ **do**

$$\widetilde{x} = x_i \oplus (y_1^{\texttt{real}}, \ldots y_{t-1}^{\texttt{real}}), \quad \widetilde{y} = y_t^{\texttt{real}}$$

$$\widetilde{\mathcal{D}} \leftarrow \widetilde{\mathcal{D}} \cup \{(\widetilde{x}, \widetilde{y})\}$$

4: $f_\theta \leftarrow$ Call **Algorithm 1** on $\widetilde{\mathcal{D}}$

Output: Generative model f_θ

```python
def gem_loss(logits, labels, beta=0.7, ignore_index=-100, h="linear"):

    shift_logits = logits[..., :-1, :].contiguous()
    shift_labels = labels[..., 1:].contiguous()

    mask = shift_labels != ignore_index
    shift_logits = shift_logits[mask]
    shift_labels = shift_labels[mask]

    with torch.no_grad():
        logits_on_labels = torch.gather(
            shift_logits, dim=-1, index=shift_labels.unsqueeze(-1)
        ).squeeze(-1)

        logits_diff = shift_logits - logits_on_labels.unsqueeze(-1)
        if h == "linear":
            weights = torch.ones_like(logits_diff)
        elif h == "log_sigmoid":
            weights = F.sigmoid(0.01 * logits_diff)
        else:
            raise ValueError(h)

    gene_log_probs = F.log_softmax(shift_logits, dim=-1)
    q_probs = torch.exp(
        F.log_softmax(shift_logits / beta, dim=-1)
    ).detach()

    real_log_probs = torch.gather(
        gene_log_probs, dim=-1, index=shift_labels.unsqueeze(-1)
    ).squeeze(-1)

    loss = -torch.sum(
        q_probs * weights * (real_log_probs.unsqueeze(-1) - gene_log_probs), dim=-1
    ).mean()

    return loss
```

Listing 1: Pytorch Code of GEM

We have two remarks regarding the implementation above. First, we use a coefficient of 0.01 to scale the input in the `log-sigmoid` function. This ensures that the function behaves nearly linearly. Second, this implementation requires almost the same GPU memory and computation time as the CE loss.

C Proof

Proof of Proposition 1. When h is a linear function, we have that

$$\begin{aligned}
\mathcal{L}_q(f) &= \mathbb{E}_x \mathbb{E}_{y^{\text{real}} \sim p(\cdot|x)} \mathbb{E}_{y^{\text{gene}} \sim q(\cdot|x)} \left[\log f(y^{\texttt{real}}|x) - \log f(y^{\texttt{gene}}|x) \right] \\
&= \mathbb{E}_x \mathbb{E}_{y^{\text{real}} \sim p(\cdot|x)} \mathbb{E}_{y^{\text{gene}} \sim q(\cdot|x)} \left[\log f(y^{\texttt{real}}|x) \right] - \mathbb{E}_x \mathbb{E}_{y^{\text{real}} \sim p(\cdot|x)} \mathbb{E}_{y^{\text{gene}} \sim q(\cdot|x)} \left[\log f(y^{\texttt{gene}}|x) \right] \\
&= \mathbb{E}_x \mathbb{E}_{y^{\text{real}} \sim p(\cdot|x)} \left[\log f(y^{\texttt{real}}|x) \right] - \mathbb{E}_x \mathbb{E}_{y^{\text{gene}} \sim q(\cdot|x)} \left[\log f(y^{\texttt{gene}}|x) \right]
\end{aligned}$$

For any $x \in \mathcal{X}$, we have that

$$\frac{\partial \mathcal{L}}{\partial f} = \frac{p - q}{f} \tag{5}$$

To calculate the stationary point of $\mathcal{L}$, we require that $p = q$. Since $q = \texttt{softmax}(1/\beta \cdot \log f)$, the above equality requires that $f = \texttt{softmax}(\beta \cdot \log p)$. As analyzed in Proposition 2, for $\beta = 1/(\gamma + 1)$, this corresponds to the the optimal solution of minimizing reverse KL with entropy regularization.

$\square$

Proposition 2. *For the entropy-regularized KL minimization problem in Equation (1), in the function space, we have the optimal solution:*

$$f^\star(y|x) = \frac{1}{Z_x} p(y|x)^{1/(\gamma+1)}$$

where Z_x is a normalization constant $\sum_{y'} p(y'|x)^{1/(\gamma+1)}$.

The proof is based on the optimality condition of constrained optimization. Its proof can be found in the previous literature (see, e.g., [58, Appendix A]). We note that the above closed-form solution cannot be applied in practice because we do not have access to the density function of the data distribution p.

D Discussion

D.1 Future Work

We focus on the SFT stage in this paper and recognize that the models trained with our proposed methods can be further refined in subsequent stages. Notably, the enhanced diversity achieved by our approach can be beneficial in several contexts: it supports scaling up test-time computation [6, 50], helps mitigate preference collapse in preference alignment [65], facilitates self-improvement through distillation with best-of-n techniques [48], and helps mitigate model collapse in synthetic data generation [49, 63]. We see the potential of our method in these areas and plan to explore these topics in future work.

D.2 CE with Entropy Regularizer

We discuss the formulation of forward KL with entropy regularization in this section:

$$\max_f \mathbb{E}_x \Big\{ \underbrace{\mathbb{E}_{y\sim p(\cdot|x)}[\log f(y|x)]}_{=-D_{\mathrm{KL}}(p,f)+\text{constant}} + \gamma \cdot \underbrace{\mathbb{E}_{y\sim f(\cdot|x)}[-\log f(y|x)]}_{=\mathcal{H}(f)} \Big\} \tag{6}$$

This formulation supports the proposed Principle 2 but not Principle 1. We find that this formulation leads to an improper increase in tail probabilities when maximizing the entropy, as illustrated in Figure 5. In the context of LLMs, this increase often translates into nonsensical tokens in the vocabulary, leading to undesirable generation outputs (if additional strategies like top-k and top-p sampling are not used). A concrete example is provided in Table 3. The core issue arises because the gradient of the entropy regularizer can dominate for tokens with low probabilities. Specifically, the gradient of the forward KL is computed as $-p/f$, where the division is element-wise, and the gradient of the entropy is $-(1+\log f)$. Consequently, for tokens with low probabilities in both f and p, the gradient given by the forward KL is much smaller than that given by the entropy regularizer, thus disproportionately increasing the tail probabilities. In contrast, the proposed reverse KL formulation with entropy regularization does not have this issue. This is because the optimization is defined over the generative distribution f in our formulation, ensuring balanced gradients even for tokens with low probabilities (refer to Equation (5)).

D.3 Intuition and Example of GEM

We provide an intuitive understanding of GEM by explaining its training mechanism on a simple model: for a fixed $x \in \mathcal{X}$, we model $f_\theta(y|x) = \texttt{softmax}(\theta_x)$ with $\theta_x \in \mathbb{R}^K$. Consider h as the linear function described in Proposition 1. For a paired sample $(y^{\texttt{real}}, y^{\texttt{gene}}) = (i,j)$, we have the gradient for this sample:

$$\nabla_\theta \mathcal{L}_q(f_\theta)[i,j] = \begin{cases} w_{ij} e_{ij} & \text{if} \quad i \neq j \\ \mathbf{0} & \text{otherwise} \end{cases}$$

Here $w_{ij} = p(y^{\texttt{real}}|x) q(y^{\texttt{gene}}|x)$ lies in $[0,1]$, and e_{ij} is the vector with i-th element being 1 and the j-th element being -1 and 0 otherwise. Thus, the gradient of this paired data gives a direction for moving the logit θ_x from j-th position to i-th position, with the weight w_{ij}.

Consider a numerical example where $\theta_x = [2,1]$ with $K=2$, so $f = [0.73, 0.27]$. For $\beta = 0.7$, we have $q = [0.81, 0.19]$, which is more peaked compared with f. Given the data distribution

Figure 5: Distributions of next-token probabilities for trained models with the `UltraFeedback` dataset, presented from top to bottom: CE, CE+Entropy, GEM-LS. The prompt is "Give me a single-digit number". The top 300 probabilities are shown with a subsampling rate of 20 for clear visualization. A red dotted line indicates the probability threshold of 10^{-4}. The figure demonstrates that the CE+Entropy model has a longer tail with higher probabilities assigned to some nonsensical tokens, marked with crosses.

Table 3: Examples of generated sentences for models trained with the `UltraFeedback` dataset. The symbol "[...]" indicates good sentences that have been omitted to save space. The vanilla generation configuration `temperature=1`, `top_k=None`, and `top_p=1.0` is used. This example shows that CE with entropy regularization can generate nonsensical tokens (shown in red) due to the increase in tail probabilities.

Prompt	Whats the largest star in our galaxy?
CE	Hello! Atlantis is a legendary city that was said to have existed in ancient Greece. According to the story, it was a highly advanced and prosperous city that was located on an island in the ocean. [...]
CE+Entropy	Hello! Atlantis Documentary is a 2019 American documentaryéhoFLICT film directed by Já oblík and produced by Werner Herzog. The film explores the history and legacy of Atlantis, □ an ancient Greek city-state that was said to have_calendar knowledge and advanced technology, through interviews with scholars and historians.ython
GEM-LS	Hello there! As a helpful, respectful, and honest assistant, I'd be happy to help you explore the fascinating topic of Atlantis! Atlantis is an ancient Greek myth that tells the story of a legendary realm said to have existed in the Atlantic Ocean, west of the Pillars of Hercules. [...]

$p = [0.9, 0.1]$, the gradient of GEM is $0.9 \cdot 0.19 \cdot [1, -1] + 0.1 \cdot 0.81 \cdot [-1, 1] = [0.09, -0.09]$, leading to a relative logit change of 0.18. In comparison, the CE's gradient in this case is $[0.17, -0.11]$, resulting in a relative logit change of 0.28, which is 1.6 times larger then GEM. When converged, GEM would give a flatter distribution $[0.82, 0.18]$ due to the induced entropy regularization.

We have two remarks for the above analysis. First, we see that the distribution q determines the weights of probability transportation. Generally, for $0 < \beta < 1$, a *narrowed* distribution q, shifted from f, prioritizes the high-probability regions in f for probability transportation, while low-probabilities regions in f contributes less. This contrasts with CE, which would push probabilities of non-labeled tokens towards the labeled ones, potentially causing overfitting. Second, we note that h also determines how much probability is shifted. Specifically, we have $w_{ij} = p(y^{\texttt{real}}|x) q(y^{\texttt{gene}}|x) h'$ for a general function h. For the linear function studied, h' is always equal to 1. Another possible choice for h is the log-sigmoid function $h(u) = \log \texttt{sigmoid}(u) = u - \log(1 + \exp(u))$, which is studied in previous research [26]. This function provides a weighting effect. Since $h' = \texttt{sigmoid}(\log f(y^{\texttt{gene}}|x) - \log f(y^{\texttt{real}}|x)) \in (0, 1)$, it results in a large weight when $y^{\texttt{real}}$ is not yet dominant in the probability distribution, and a small weight when $y^{\texttt{real}}$ has already become dominant. Later on, we will study this function in experiments.

E Experiment Details

All experiments are conducted using A800-80GB GPUs with the DeepSpeed distributed training framework, utilizing ZeRO-2 and gradient checkpointing without offloading. We use flash-attention-2 with deterministic backward for reproducibility. The experiments are based on the pretrained Llama-3-8B model, using Adam as the optimizer with a global batch size of 128. Following [67, 34, 13], the learning rate is set to 2e-5, with a warm-up ratio of 0.03 and cosine learning rate decay. Training is performed over 3 epochs. All supervised datasets are formatted into the chat format using the Llama-3-8B-Instruct's tokenizer. When generation of responses is required for evaluation, we use the vLLM to accelerate inference.

E.1 UltraFeedback

We use the dataset filtered by HuggingfaceH4 team, which is available at https://huggingface.co/datasets/HuggingFaceH4/ultrafeedback_binarized. The dataset contains 61,135 training samples and 1,000 test samples. For training, we set the maximum sequence length to 2,048, dropping longer sequences and padding shorter ones. To achieve a global batch size of 128, we use a per-device batch size of 4, a gradient accumulation step of 4, and 4 GPUs. The training times takes about 24 GPU hours. For the CE method, we have tuned hyperparameters for weight decay and entropy regularization, selecting values from $\{0.1, 0.01, 0.001\}$. In both cases, a value of 0.1 provided the best overall results. For NEFT, we use a noise scale hyperparameter of 5, as recommended by [23].

Evaluation metrics, including perplexity, and entropy, are based on these 1,000 test samples. For entropy calculation, we compute the conditional entropy, whose expectation can be calculated exactly, and average over the sequence. For the instruction-following evaluation, we use the IFEval benchmark from [70]. We apply greedy decoding with a maximum generation length of 1,024 tokens.

For the diversity evaluation in poem writing, we use prompts derived from the `poetry` dataset on the Huggingface website, which includes 573 poems on themes like love, nature, and mythology by poets such as William Shakespeare. We prompt the trained models with questions like, "Write a poem titled '[X]' with no more than 200 words," where [X] is a title from the dataset. For story writing, we create 500 prompts based on the `ROC Story` dataset (2017 winter) [35], asking models to "Write a story titled '[X]' with no more than 200 words," where [X] is a title from the dataset. The maximum number of generation tokens is set to 512. The evaluation script follows the methodology from previous work by [28], using the script available at https://github.com/facebookresearch/rlfh-gen-div. For each question, 16 samples with the generation configuration `temperature=1.0`, `top_k=50`, `top_p=0.9` is used.

For the chat evaluation, we use the 805 test questions from the `AlpacaEval` dataset and employ the reward model `FsfairX-LLaMA3-RM-v0.1`. The maximum generation sequence length is set to 2048. For each question, 32 samples are generated with the configuration `temperature=0.6`, `top_k=50`, `top_p=0.9`. To calculate the win rate, we use the Bradley-Terry model:

$$\mathbb{P}(y \succ y' \mid x) = \frac{\exp(r(x,y))}{\exp(r(x,y)) + \exp(r(x,y'))}.$$

We use GPT-4 generated responses as a baseline for calculating the win rate, specifically the `gpt4_1106_preview`[6] version.

For the math reasoning task on `GSM8K`, we use the following prompt:

> Your task is to answer the question below. Give step-by-step reasoning before you answer, and when you're ready to answer, please use the format "The answer is: ...".
> Question: {question}

Answer extraction from the generated responses follows the approach from previous work [67], using the script available at https://github.com/meta-math/MetaMath/blob/main/eval_gsm8k.py. For each question, 32 responses are generated with the configuration `temperature=0.6`, `top_k=50`, `top_p=0.9`. The reported accuracy is based on 1,319 test questions.

For the code generation tasks on `HumanEval` and `MBPP`, there are 164 test questions for `HumanEval` and 378 test questions for `MBPP`. We use the prompt from [62]:

> You are an exceptionally intelligent coding assistant that consistently delivers accurate and reliable responses to user instructions.
> @@ Instruction
> {instruction}

For each question, 200 responses are generated with the configuration `temperature=0.6`, `top_k=50`, `top_p=0.9` to estimate the pass rate. The evaluation scripts are from https://github.com/ise-uiuc/magicoder/blob/main/experiments/text2code.py.

E.2 MagiCoder

We use the `MagiCoder-OSS-Instruct` dataset [62], which contains 74,197 training samples and 1,000 test samples (randomly selected from the original training set). The maximum sequence length

[6] https://github.com/tatsu-lab/alpaca_eval/blob/main/results/gpt4_1106_preview/model_outputs.json

for training is 1,024. To achieve a global batch size of 128, we use a per-device batch size of 8, gradient accumulation steps of 2, and 8 GPUs. The training takes approximately 24 GPU hours. The evaluation method is the same as previously described.

E.3 MetaMathQA

We use the `MetaMathQA` dataset [67]. To make the code generation task manageable, we select a subset of 79,000 samples for training and 1,000 samples for evaluation. The maximum sequence length for training is set to 1,024. To achieve a global batch size of 128, we use a per-device batch size of 8, gradient accumulation steps of 2, and 8 GPUs. Training takes approximately 24 GPU hours. The evaluation method is as previously described. For the `MATH` task, the prompt is the same as for the `GSM8K` task.

F Additional Results

F.1 General Purpose Fine-tuning

Perplexity and Entropy. For trained models, we also examine two statistics: perplexity, and entropy of the output distribution. We evaluate these two statistics on 1,000 test samples from the `Ultrafeedback` dataset. Results are reported in Table 4. Using CE as a baseline, we make several observations. First, weight decay does not significantly change the statistics. Second, directly incorporating entropy regularization increases both perplexity and entropy considerably. Notably, this increase is mainly due to relatively large tail probabilities. Third, GEM generally reduces perplexity while increasing entropy.

Table 4: Evaluation perplexity and entropy. Models are trained with the `UltraFeedback` dataset.

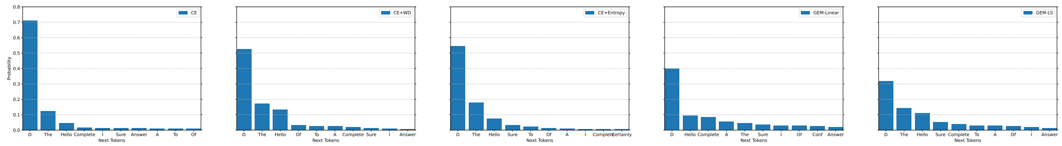
Method	UltraFeedback	
	Evaluation Perplexity	Evaluation Entropy
CE	3.48	0.68
CE+WD	3.46	0.68
CE+Entropy	3.78	2.65
NEFT	3.22	0.78
GEM-LS	3.18	1.19
GEM-Linear	3.16	1.16

Next-Token Prediction Distributions. We demonstrate the distribution collapse issue associated with the CE method using three simple prompts for the trained LLMs: 1) "Complete this sequence with a single letter: A, B, C, ___"; 2) "Give me a single-digit number"; and 3) "Tell me a type of fruit". All prompts are designed to have answers with 1 token for visualization.[7] The distributions are visualized in Figure 6. We see GEM-trained models produce flatter distributions, indicating support for multiple possible answers.

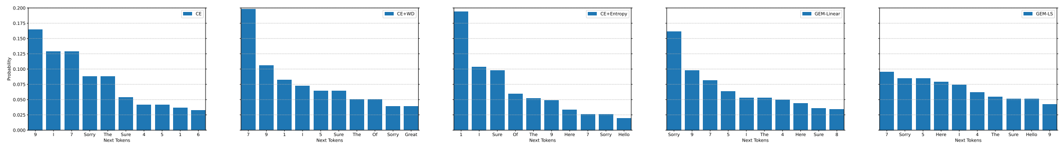
Alignment Tax. We assess the alignment tax by examining the performance drop in in-context learning abilities across six tasks: ARC, GSM8K, HellaSwag, MMLU, TruthfulQA, and WinoGrande, as listed in the OpenLLM leaderboard. For ARC, we use the arc-challenge metric with 25 shots. HellaSwag is evaluated with 10 shots, while TruthfulQA is tested with zero shots. MMLU, GSM8K, and WinoGrande are assessed using five shots each. Results are reported in Table 5. We observe that all fine-tuned models suffer from forgetting acquired in-context learning abilities. However, GEM-tuned models have the smallest alignment tax among these baselines.

Chatting, Math Reasoning, and Code Generation. We provide the detailed results in Tables 6 to 8. We observe that even with less generation samples, GEM also shows better performance.

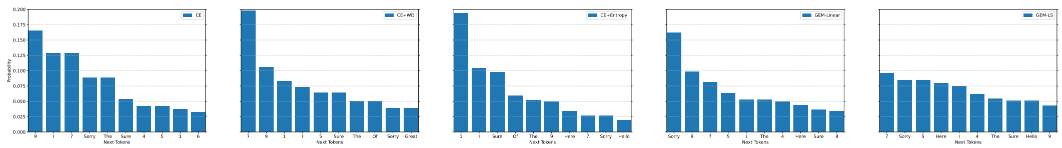
[7] For the first prompt, while "D" is the most likely answer, "A" could also be a valid response due to the pattern A, B, C, A, B, C, ...



(a) Prompt: Complete this sequence with a single letter: A, B, C, ___



(b) Prompt: Give me a single-digit number.



(c) Prompt: Tell me a type of fruit.

Figure 6: Distributions of next-token probabilities for trained models with the `UltraFeedback` dataset, presented from left to right: CE, CE+WD, CE+Entropy, and GEM-Linear, and GEM-LS. Only top-10 probabilities are visualized for clarity. These examples highlight the issue of limited generation diversity in CE.

Table 5: Performance of in-context learning on the benchmark `OpenLLMLeaderBoard`. Models are trained with the `UltraFeedback` dataset.

Method	**Open LLM LeaderBoard**						
	ARC	**GSM8K**	**HellaSwag**	**MMLU**	**TruthfulQA**	**WinoGrande**	**Average**
Pre-trained	58.36	50.64	82.14	65.18	43.86	77.58	62.96
CE	56.23	41.70	79.70	58.29	48.72	70.64	59.21
CE+WD	55.12	**41.77**	79.53	**59.66**	48.12	71.59	59.30
CE+Entropy	57.51	41.02	80.10	59.47	48.83	71.19	59.69
NEFT	55.29	38.21	77.90	56.17	**49.46**	72.38	58.24
GEM-Linear	57.68	41.02	81.60	59.08	47.59	73.32	60.05
GEM-LS	**58.28**	40.56	**81.81**	59.39	47.96	**73.64**	**60.27**

Table 6: Evaluation of reward and win rate on `AlpacaEval` dataset. Models are trained with the `UltraFeedback` dataset.

Method	**Reward**				**Win Rate**			
	BON@4	**BON@8**	**BON@16**	**BON@32**	**BON@4**	**BON@8**	**BON@16**	**BON@32**
CE	1.06	1.43	1.86	2.39	26.59	31.35	37.43	46.61
CE+WD	1.09	1.47	1.85	2.41	27.17	32.00	37.59	46.98
CE+Entropy	1.11	1.48	1.89	2.46	26.86	31.83	37.84	47.69
NEFT	**1.14**	**1.55**	1.94	2.52	**27.51**	**32.78**	38.73	48.80
GEM-Linear	1.12	1.52	1.94	2.51	**27.27**	32.36	38.76	48.50
GEM-LS	1.11	1.52	**1.96**	**2.56**	26.98	32.53	**39.18**	**49.46**

Table 7: Evaluation of accuracy on the math reasoning task GSM8K. Models are trained with the UltraFeedback dataset.

Method	GSM8K							
	MV@4	**MV@8**	**MV@16**	**MV@32**	**BON@4**	**BON@8**	**BON@16**	**BON@32**
CE	51.63	55.57	58.61	62.17	65.28	74.68	82.11	90.22
CE+WD	54.51	58.76	62.47	65.66	69.90	77.48	84.46	90.45
CE+Entropy	53.75	56.63	60.58	64.44	67.32	76.57	83.93	91.21
NEFT	55.12	**61.18**	**65.13**	66.72	**70.36**	**80.67**	**86.96**	92.12
GEM-Linear	53.68	58.07	62.77	65.58	69.83	79.30	86.50	91.96
GEM-LS	**55.95**	60.42	64.82	**67.02**	70.05	79.68	**86.96**	**92.72**

Table 8: Performance of pass rate on the code generation tasks HumanEval and MBPP. Models are trained with the UltraFeedback dataset.

Method	HumanEval				MBPP			
	Pass@10	**Pass@20**	**Pass@50**	**Pass@100**	**Pass@10**	**Pass@20**	**Pass@50**	**Pass@100**
CE	58.06	62.51	67.50	70.88	62.71	65.73	69.13	71.18
CE+WD	56.18	61.53	67.85	71.91	63.13	66.35	69.40	71.35
CE+Entropy	58.85	64.02	70.29	74.44	65.50	68.75	71.77	73.48
NEFT	52.62	59.47	67.08	71.65	64.58	67.88	71.82	74.51
GEM-Linear	60.34	66.12	73.12	77.97	64.54	68.57	72.30	74.33
GEM-LS	**60.94**	**66.95**	**73.83**	**78.47**	**67.28**	**71.50**	**75.50**	**77.64**

F.2 Domain-specific Fine-tuning

We provide the detailed results in Tables 9 to 11. The results indicate that GEM outperforms CE even with fewer generated samples.

Table 9: Evaluation of accuracy on the math reasoning task GSM8K. Models are trained with the MetaMathQA dataset.

Method	GSM8K							
	MV@4	**MV@8**	**MV@16**	**MV@32**	**BON@4**	**BON@8**	**BON@16**	**BON@32**
CE	73.46	73.77	75.13	76.57	76.50	80.74	85.14	90.67
CE+WD	73.84	75.06	76.50	78.24	77.94	81.05	86.05	90.67
CE+Entropy	75.06	76.04	77.71	**79.68**	79.61	83.70	88.70	92.95
NEFT	72.71	74.53	75.82	76.88	78.77	83.40	87.19	92.65
GEM-Linear	74.83	75.82	**78.09**	78.77	**81.43**	**85.60**	**89.69**	**93.56**
GEM-LS	**75.21**	**76.35**	77.33	79.53	80.82	85.06	89.31	93.33

Table 10: Evaluation of accuracy on the math reasoning task MATH. Models are trained with the MetaMathQA dataset.

Method	MATH							
	MV@4	**MV@8**	**MV@16**	**MV@32**	**BON@4**	**BON@8**	**BON@16**	**BON@32**
CE	26.40	27.04	28.30	29.34	33.20	39.98	48.20	58.46
CE+WD	26.20	27.02	28.38	29.56	33.22	39.32	47.54	57.10
CE+Entropy	**28.06**	29.26	30.34	31.20	35.58	41.84	50.66	59.64
NEFT	26.18	24.46	28.74	30.12	34.46	41.54	48.98	58.64
GEM-Linear	27.62	**29.30**	**30.64**	**31.48**	**36.82**	**43.74**	**52.04**	**60.30**
GEM-LS	27.46	28.88	29.92	31.00	36.00	42.98	50.96	60.12

Table 11: Performance of pass rate on the code generation tasks `HumanEval` and `MBPP`. Models are trained with the `MagiCoder-OSS-Instruct` dataset.

Method	**HumanEval**				**MBPP**			
	Pass@10	**Pass@20**	**Pass@50**	**Pass@100**	**Pass@10**	**Pass@20**	**Pass@50**	**Pass@100**
CE	58.71	61.50	64.18	65.86	66.54	68.68	70.76	71.95
CE+WD	58.33	61.06	63.77	65.89	65.96	68.38	70.67	71.89
CE+Entropy	58.66	62.66	66.79	69.17	69.47	71.76	73.79	75.02
NEFT	55.86	59.45	63.01	65.12	66.53	69.06	71.29	72.32
GEM-Linear	58.69	62.39	67.16	70.64	72.00	74.54	76.74	78.08
GEM-LS	**65.15**	**68.73**	**72.64**	**75.58**	**73.30**	**75.90**	**78.42**	**79.97**