ENTROPIC DISTRIBUTION MATCHING FOR SUPERVISED FINE-TUNING OF LLMS: LESS OVERFITTING AND BETTER DIVERSITY

Anonymous authors
Paper under double-blind review

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 aims 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.

We demonstrate the effectiveness of GEM by fine-tuning pre-trained models ranging from 3B to 70B in size. GEM consistently outperforms CE, reducing overfitting as indicated by lower evaluation perplexity and improved instruction-following performance. Moreover, GEM enhances output diversity, generating more varied and creative responses in tasks such as poem and story writing. This increase in diversity also translates into test-time performance gains. For instance, when fine-tuning Llama-3-8B, GEM achieves a 5-point improvement in math reasoning and a 8-point improvement in code generation tasks, leveraging majority voting and best-of-n sampling strategies.

1 INTRODUCTION

Large Language Models (LLMs) (OpenAI, 2023; Touvron et al., 2023; Team et al., 2024) 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 (Raffel et al., 2020; Wei et al., 2021; Chung et al., 2024), also known as Supervised Fine-Tuning (SFT) (Ouyang et al., 2022; Bai et al., 2022), 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 (Burns et al., 2023; Tunstall et al., 2023; Liu et al., 2023). We expect models to generalize well by providing accurate answers and hope these answers are diverse as well. While the importance of generalization is clear, generation diversity is also important, especially with the trend of scaling up test-time compute (Snell et al., 2024; Brown et al., 2024; Wu et al., 2024b). These emerging studies have shown that scaling up test-time compute, by selecting the optimal response from multiple generated options, can solve many complex mathematical reasoning tasks, with output diversity being a key factor in this process (Wang et al., 2023). Additionally, many applications benefit from diverse responses. For example, in creative writing, diverse outputs from generative models can inspire new ideas (Colton & Wiggins, 2012). In chit-chat dialogues, users also appreciate having multiple options to suit their preferences (Li et al., 2015). AI interfaces like ChatGPT and Claude AI address this need by offering features such as 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 (Burns et al., 2023; Jain et al., 2023; Gekhman et al., 2024) and lack of generation diversity (Padmakumar & He, 2023; O'Mahony et al., 2024).

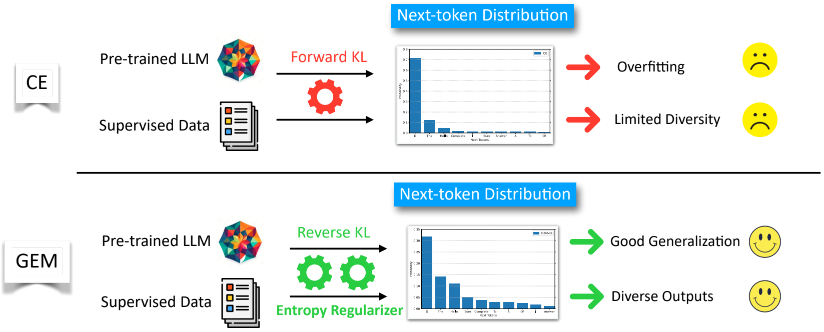


Figure 1: Illustration of the standard CE and the proposed method GEM for SFT of LLMs.

These limitations stem from the theoretical underpinnings of CE loss. In theory, optimizing CE loss corresponds to minimizing the *forward* Kullback–Leibler (KL) divergence between the data distribution and the generative distribution of the LLM.[1] 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 (Ge et al., 2023; Zeng et al., 2023), latch onto spurious features (Burns et al., 2023), and lose in-context learning abilities that already been acquired in the pre-training (a.k.a. alignment tax) (Ouyang et al., 2022; Bai et al., 2022). 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 (Pereyra et al., 2017; Dubey et al., 2018), suggesting that these issues are interrelated. To address these concerns, techniques like weight decay (Touvron et al., 2023; Burns et al., 2023) or noisy perturbations to embeddings (Jain et al., 2023) are commonly applied alongside CE loss. However, they have their own limitations (see discussion in Appendix D), highlighting the need for more principled solutions.

In this paper, we frame the SFT of LLMs as a distribution matching problem, introducing the maximum entropy principle (Jaynes, 1982) to guide the process. This principle promptes 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) (Goodfellow et al., 2014). 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 validate the effectiveness of GEM by fine-tuning pre-trained models ranging from 3B to 70B in size, including Qwen2.5-3B (Team, 2024), Qwen2.5-7B (Team, 2024), Llama-3-8B (Dubey et al., 2024), Gemma-2-9B (Team et al., 2024), and Llama-3.1-70B (Dubey et al., 2024). We find that GEM consistently outperforms CE, reducing overfitting with lower evaluation perplexity and improved instruction-following performance. GEM also mitigates the alignment tax issue by demonstrating better in-context learning performance. Additionally, GEM enhances output diversity, generating more varied and creative responses in tasks such as poem and story writing. This boost in diversity translates into test-time performance gains. For example, when fine-tuning Llama-3-8B, GEM achieves a 5-point improvement in math reasoning and a 8-point improvement in code generation tasks, using majority voting and best-of-n strategies. Importantly, to match the performance of baselines, GEM often requires only 0.5x the sampling budget.

To summarize, our contributions are threefold:

---

[1] 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.

- 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 RELATED WORK

We review relevant work in the main text, with additional related work discussed in Appendix A.

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 the synthetic data (see, e.g., (Yu et al., 2023; Wei et al., 2024; Zhang et al., 2024a)), 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. (2018) 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. (2017) 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. (2017) 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. (2023) explored the maximum entropy regularization by using GFlowNet (Bengio et al., 2021), but their methods require a reward function rather than supervised data.

3 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 Transformer (Vaswani et al., 2017), 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.

4 ENTROPIC DISTRIBUTION MATCHING

In this paper, we explore principled approaches for SFT, presenting two core principles. The first principle tackles the issues of overfitting and limited output diversity. We draw inspiration from neuroscience, specifically the concept of avoiding over-memorization and achieving balanced learning. In neuroscience, synaptic plasticity, particularly homeostatic plasticity, underscores the importance of maintaining balance in learning processes (Turrigiano, 2008; 2012). Overly strengthening certain neural connections can lead to rigid, maladaptive behaviors, analogous to how assigning excessively high probabilities to observed tokens can result in over-memorization in models, thereby limiting their ability to adapt and generalize. 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 realized by incorporating an entropy regularizer into the learning process. Expanding on this, our second principle advocates for a *generative* approach to distribution matching. This approach encourages the model to learn from its own generated data and mistakes, rather than merely imitating supervised demonstrations. Unlike the traditional CE loss, which leads the model to imitate training data labels passively, a generative approach involves learning through self-generated mistakes. This principle is grounded in cognitive science (Schulz & Bonawitz, 2007; Gweon et al., 2014), which demonstrates that children learn more effectively through exploration and experimentation, adjusting their understanding based on discrepancies between expectations and reality. Similarly, research on generative models (Goodfellow et al., 2014; Ho & Ermon, 2016) supports this notion by showing how models can learn to produce realistic data through iterative refinement. To summarize, we propose:

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

4.1 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:

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

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 over $y \sim f(\cdot|x)$), similar to GANs (Goodfellow et al., 2016).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 (Snell et al., 2024; Brown et al., 2024).

We note that the two terms in Equation (1) are well-aligned in the sense that both are defined over the generative distribution f. In contrast, adding entropy regularization to the CE loss does not achieve this. It has limitations in improper increasing tail probabilities of the distribution. For a more detailed discussion, please refer to Appendix D.2.

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.

4.2 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 (Jolicoeur-Martineau, 2018), 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:

$$\begin{aligned} \max_{f} \quad & \mathcal{L}_q(f) \triangleq \mathbb{E}_x \mathbb{E}_{y^{\texttt{real}} \sim p(\cdot|x)} \mathbb{E}_{y^{\texttt{gene}} \sim q(\cdot|x)} \left[h\left(\log f(y^{\texttt{real}}|x) - \log f(y^{\texttt{gene}}|x) \right) \right] \\ \text{s.t.} \quad & q = \operatorname*{argmax}_{\pi} \mathbb{E}_x \mathbb{E}_{y^{\texttt{gene}} \sim \pi(\cdot|x)} \left[\log f(y^{\texttt{gene}}|x) \right] + 1/\beta \cdot \mathcal{H}(\pi(\cdot|x)) = \texttt{softmax}(1/\beta * \log f) \end{aligned} \tag{2}$$

Here we use $y^{\texttt{real}}$ to denote the supervised label in the dataset and $y^{\texttt{gene}}$ 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, and π is an arbitrary distribution introduced for mathematical clarity. 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 (LeCun et al., 2006) (or the reward in inverse reinforcement learning (Ho & Ermon, 2016)). Simultaneously, we update the distribution q that maximizes 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 Equation (2) provides the optimal solution of reverse KL with entropy regularization in Equation (1). In practice, we can implement f using a Transformer (Vaswani et al., 2017) 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 Generative and Entropy-regularized Matching of distributions. We also note that Proposition 1 relies on $\beta > 0$, meaning that GEM cannot solve the pure reverse KL minimization problem.

Algorithm 1 GEM

```
Input: Dataset $\mathcal{D} = \{(x_i, y_i^{\texttt{real}})\}$
for iteration $k = 1, \dots,$ do
    Set $q_k = \texttt{softmax}(1/\beta * \log f_{\theta_k})$
    Compute loss $\mathcal{L}_q(f_\theta) = \sum_i \sum_{y^{\texttt{gene}}} q(y^{\texttt{gene}}|x_i) \cdot h\left(\left[\log f_\theta(y_i^{\texttt{real}}|x_i) - \log f_\theta(y^{\texttt{gene}}|x_i)\right]\right)$
    Update $\theta_{k+1} = \theta_k + \eta \cdot \nabla_\theta \mathcal{L}_q(f_\theta) |_{\theta=\theta_k}$
Output: Generative model $f_\theta$
```

Scalability. We highlight two key computational advantages of GEM that enable efficient generative distribution matching and its scalability to billion-parameter models:

- Single Model Optimization: GEM trains only a *single* model f by leveraging a closed-form solution for q. This contrasts with GAN-style distribution matching methods (Goodfellow et al., 2014; Jolicoeur-Martineau, 2018), which require the simultaneous optimization of two models (a generator and a discriminator), complicating the tuning process and increasing computational difficulty.
- Accurate Gradient Estimation: GEM computes the loss function and gradients using the *exact* expectation $\mathbb{E}_{y^{\texttt{gene}} \sim q(\cdot|x)}[\cdot]$, ensuring stable training by minimizing gradient estimation variance. This is feasible because q is a categorical distribution with finite elements in LLMs. In contrast, GAN-style methods typically rely on inexact stochastic gradients obtained by sampling from the generative distribution f, making training notoriously difficult.

Training Dynamics and Intuition. 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 compared with f. Given the data distribution $p = [0.9, 0.1]$, 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, squeezed 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 (Jolicoeur-Martineau, 2020). 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.

Extension 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 Equation (2) to the following:

$$
\begin{aligned}
\max_{f} \quad & \mathbb{E}_x \mathbb{E}_{y_{1:T}^{\texttt{real}} \sim p(\cdot|x)} \mathbb{E}_{y_{1:T}^{\texttt{gene}} \sim q(\cdot|x)} \left[h \left(\log f(y_{1:T}^{\texttt{real}}|x) - \log f(y_{1:T}^{\texttt{gene}}|x) \right) \right] \\
\text{s.t.} \quad & q = \operatorname*{argmax}_{\pi} \mathbb{E}_x \mathbb{E}_{y_{1:T}^{\texttt{gene}} \sim \pi(\cdot|x)} [\log f(y_{1:T}^{\texttt{gene}}|x)] + 1/\beta \cdot \mathcal{H}(\pi(\cdot|x))
\end{aligned}
\tag{3}
$$

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, as used in (Chen et al., 2024; Li et al., 2024), might seem like a potential solution—drawing samples to approximate the gradient—we found it does not work in our setting. 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.[2] As a result, when we use stochastic sampling to estimate the gradient, it does not provide effective feedback. Please refer to Appendix D.3 for more detailed discussion.

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 (Ross et al., 2011) 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, so we still have the closed-form solution that $q(\cdot|x, y_{1:t-1}^{\texttt{real}}) = \texttt{softmax}(1/\beta \cdot \log f(\cdot|x, y_{1:t-1}^{\texttt{real}}))$. We outline the proposed procedure for dealing with sequential data in Algorithm 2 and provide its PyTorch implementation in Appendix B. Notably, thanks to the reset trick, GEM's training requires nearly the same GPU memory consumption and compute time as optimizing the CE loss; see Appendix B.

We acknowledge that our proposed solution approximates Equation (3) due to the use of the "reset" trick and greedy optimization. While the exact gap introduced by this approximation is difficult to

---

[2] 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.

quantify, we expect it to be minimal when f closely matches p through distribution matching. In such cases, sampling from the surrogate p effectively resembles sampling from the target distribution f. In the next section, we demonstrate the practical effectiveness of GEM.

5 EXPERIMENTS

In this section, we present our numerical results from fine-tuning the pre-trained Llama-3-8B model, a strong LLM, to demonstrate the effectiveness of the proposed method. Additionally, we have explored other models ranging in size from 3B to 70B, with the corresponding results provided in Appendix F.6. Detailed experimental settings are described in Appendix E.

5.1 INSTRUCTION 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 (Cui et al., 2024). This dataset contains prompts from instruction datasets like Evol-Instruct and UltraChat, and responses generated by models such as GPT-4 and Llama-2-7B/13B/70B-Chat. Following (Yu et al., 2023; Liu et al., 2023; Cui et al., 2024), 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.

As discussed, GEM has two variations: GEM-LS (h is the `log-sigmoid` function), and GE-Linear (h is the linear function), each depending on the choice of the function h. We implement the proposed GEM method with $\beta = 0.7$. 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 (Ouyang et al., 2022; Bai et al., 2022). 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 (Jain et al., 2023), 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 (Zhou et al., 2023), which includes 500 prompts from 25 types of verifiable instructions. 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` (Zhou et al., 2023). 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 evaluation above, we observe that GEM further mitigates overfitting in other aspects. For one thing, GEM achieves a **lower evaluation perplexity** (3.16) compared with CE (3.48). For

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

another thing, GEM demonstrates **reduced alignment tax**, reflected in its superior in-context learning ability, achieving 60.3 compared with CE's 59.21. For more details, please refer to Appendix F.

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`[3] dataset, which includes 573 poems on themes such as love, and mythology. For stories, we design 500 prompts based on the ROC story dataset (Mostafazadeh et al., 2016). 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 (Kirk et al., 2023), 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.

To calculate these metrics, we ask the trained models to generate 16 samples per question. 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 1, brings the benefit of output diversity. NEFT also improves output diversity, consistent with (Jain et al., 2023). Overall, GEM significantly improves output diversity compared with the baselines. Furthermore, GEM produces higher-quality writing, as detailed in the evaluation provided in Table 7 in the Appendix.

Next, we demonstrate the improved generation diversity and generalization of GEM in **test-time compute** (Brown et al., 2024; Snell et al., 2024; Wu et al., 2024b). During the test stage, advanced generation techniques such as Best-Of-N (BON) and Majority-Voting (MV) (Wang et al., 2023) are utilized to identify superior solutions. To validate GEM's effectiveness, we conduct three experiments focusing on chatting, mathematical reasoning, and code generation. The overall performance is summarized in Figure 2, with a detailed analysis provided below.

Chatting. We assess the model's ability to generate human-preferred responses in chatting. We prompt the trained models to answer 805 questions from the `AlpacaEval` dataset (Li et al., 2023). 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`[4], which has top performance on `RewardBench` (Lambert et al., 2024), to select the best response among 32 samples. We report the win rate over GPT-4's generated response in Figure 2 (left column). The evaluation shows 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 (Jain et al., 2023). We also conduct LLM-as-a-judge (Zheng et al., 2023) to evaluate the response quality; please refer to the Appendix.

Math Reasoning. We evaluate performance on the `GSM8K` (Cobbe et al., 2021) benchmark, which contains 1,319 test questions. We prompt LLMs with chain-of-thought (Wei et al., 2022) to generate 32 responses for each question. We assess answer accuracy using both Majority-Voting (MV) (Wang et al., 2023) 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. The strong

---

[3] https://huggingface.co/datasets/merve/poetry

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

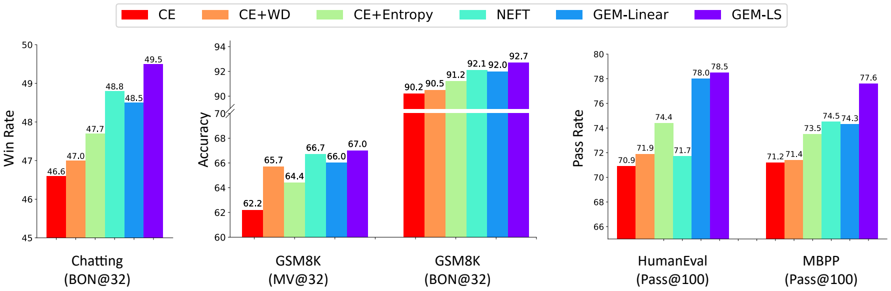


Figure 2: Performance of using advanced generation strategies such as best-of-n and majority voting in chatting (left), math reasoning (middle) and code generation (right) tasks.

performance of BON@32 indicates that while the model might know how to solve these questions, it is uncertain about these solutions in generation.

Code Generation. We consider two benchmarks: `HumanEval` (Chen et al., 2021) and `MBPP` (Austin et al., 2021). 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.

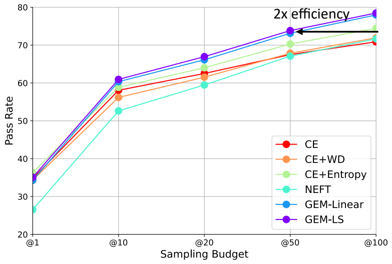


Figure 3: Pass rate on `HumanEval`. GEM demonstrates a 2x improvement in test-time computational efficiency.

We have shown that GEM outperforms CE when using the same sampling budget. Notably, GEM is highly efficient in test-time scaling, requiring only about half the sampling budget to achieve similar performance (see Figure 3). This efficiency is consistent across other tasks as well (refer to Appendix F). Furthermore, such self-generated good samples generated by GEM can be distilled back into the model through self-distillation, improving zero-shot performance. For recent advances in this area, see (Sessa et al., 2024).

5.2 DOMAIN-SPECIFIC FINE-TUNING

In this section, we conduct experiments with domain-specific datasets. For math reasoning, we use the dataset `MetaMathQA` (Yu et al., 2023). For code generation, we use the dataset `Magicoder-OSS-Instruct` (Wei et al., 2024). The experiment setup, including training details and hyperparameters, is the same as before, and the specifics are provided in the Appendix.

Math Reasoning. We consider two benchmarks: `GSM8K` and `MATH` (Hendrycks et al., 2021), which is competition-level and more challenging; see Figure 4. Following the previous set-up, we evaluate performance using Majority Voting over 32 samples (MV@32), and Best-Of-N over 32 samples (BON@32). The greedy decoding performance is also reported. We observe that the weight decay regularization performs well on `GSM8K` but shows no clear improvement on `MATH`. Furthermore, NEFT does not show improvement, even though it previously performed well in instruction-following. In contrast, GEM-LS outperforms CE on `GSM8K` by 1.2 points (1.7% relative), 2.9 points (3.8% relative), and 2.6 points (2.9% relative) for greedy decoding, MV@32, and BON@32, respectively. On the `MATH` benchmark, GEM-LS shows improvements of 1.9 points (8.0% relative), 1.7 points (5.8% relative), and 1.6 points (2.7% relative) for the same methods. These improvements in greedy decoding indicate that entropy regularization methods effectively mitigate overfitting, while the enhancements in MV and BON suggest increased generation diversity.

Code Generation. Following the previous set-up, we report the pass rate over $\{1, 10, 100\}$ on two key benchmarks, `HumanEval` and `MBPP`, in Figure 5. We observe that weight decay and NEFT do not achieve consistent improvement while entropy regularization does. Notably, GEM-LS significantly enhances performance over CE: on `HumanEval`, it improves by 4.6 points (11.7%

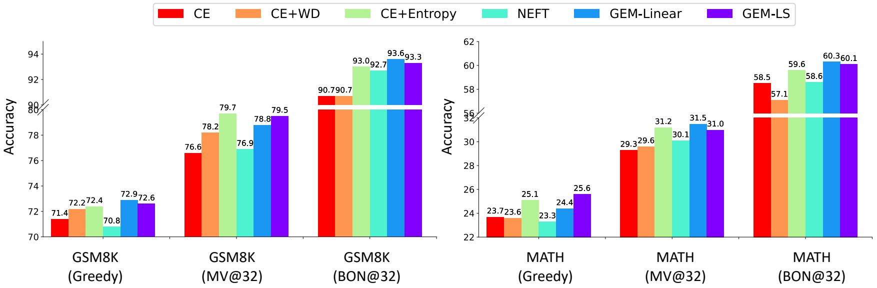


Figure 4: Performance on `GSM8K` (left) and `MATH` (right) when fine-tuning Llama-3-8B with the `MetaMathQA` dataset.

relative) for Pass@1, 6.5 points (11.1% relative) for Pass@10, and 9.7 points (14.7% relative) for Pass@100. On `MBPP`, GEM-LS achieves gains of 3.4 points (6.3% relative) for Pass@1, 6.8 points (10.2% relative) for Pass@10, and 8.0 points (11.1% relative) for Pass@100. These results suggest similar conclusions regarding overfitting and generation diversity as before.

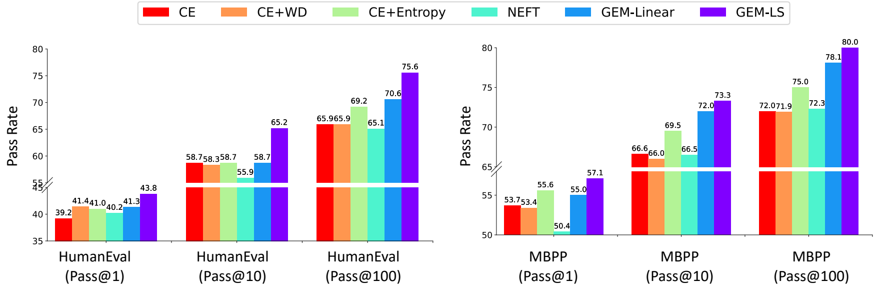


Figure 5: Performance on `HumanEval` (left) and `MBPP` (right) when fine-tuning Llama-3-8B with the `MagiCoder-OSS-Instruct` dataset.

Discussion. Overall, our results show that GEM improves both accuracy and diversity. Readers may wonder how this is possible. To address this, it is crucial to differentiate GEM from the ad-hoc method of increasing temperature to enhance diversity. Temperature adjustment, applied at the *inference* stage, reshapes the distribution by amplifying *tail probabilities*, often at the cost of reduced generalization performance (see Appendix F.4). In contrast, GEM addresses these challenges during *training*, leveraging supervision from the training data.

Readers may notice that CE with an entropy regularizer, as a training method, can also promote diversity. However, it often improperly inflates tail probabilities, as analyzed in Appendix D.2. In contrast, GEM adopts a *generative* learning approach, focusing on learning diverse responses within the mode probability regions (see analysis in Section 4.2). By targeting these regions, GEM can capture multiple valid responses that embody genuine diversity and generalization, rather than generating a mix of correct and incorrect outputs to artificially enhance diversity. Visualizations of the learned distributions are provided in Figure 7 in the Appendix for reference.

6 CONCLUSION

In this paper, we develop a method called GEM, as an alternative to the widely used CE loss, for the SFT of LLMs to address the challenges of overfitting and limited generation diversity. This method is designed within the framework of distribution matching with maximum entropy regularization. Notably, GEM achieves good generalization performance, and the improved diversity also benefits test-time computation in downstream tasks.

Our method has broader applicability to other research problems. Notably, the enhanced diversity achieved by our approach can be beneficial in several contexts: it helps mitigate preference collapse in preference alignment (Xiao et al., 2024), facilitates self-improvement through distillation with best-of-n techniques (Sessa et al., 2024), and helps mitigate model collapse in synthetic data generation (Shumailov et al., 2023; Wu et al., 2024a). Please see Appendix G for the discussion. We see the potential of our method in these areas and plan to explore these topics in future work.

ETHICS STATEMENT

Our work focuses on designing better algorithms for fine-tuning large language models, aiming to enhance their effectiveness and broaden their applications. In particular, the entropy regularizer we introduce for distribution matching encourages more diverse outputs from language models. We do not foresee any direct negative impacts from this approach.

REPRODUCIBILITY STATEMENT

The proof of Proposition 1 is provided in Appendix C. The PyTorch's implementation of Algorithm 2 is given in Appendix B. Experiment details to reproduce our numerical results can be found in Appendix E. We intend to release our code and model checkpoints upon acceptance.

REFERENCES

Bo Adler, Niket Agarwal, Ashwath Aithal, Dong H Anh, Pallab Bhattacharya, Annika Brundyn, Jared Casper, Bryan Catanzaro, Sharon Clay, Jonathan Cohen, et al. Nemotron-4 340b technical report. *arXiv preprint arXiv:2406.11704*, 2024.

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.

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.

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.

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.

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

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.

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.

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.

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.

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.

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.

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

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.

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

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

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.

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.

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.

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

Yanzhu Guo, Guokan Shang, Michalis Vazirgiannis, and Chloé Clavel. The curious decline of linguistic diversity: Training language models on synthetic text. *arXiv preprint arXiv:2311.09807*, 2023.

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.

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.

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

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.

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.

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

Alexia Jolicoeur-Martineau. The relativistic discriminator: a key element missing from standard gan. *arXiv preprint arXiv:1807.00734*, 2018.

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

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

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.

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.

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

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.

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.

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.

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.

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.

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

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.

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

Weronika Ormaniec, Felix Dangel, and Sidak Pal Singh. What does it mean to be a transformer? insights from a theoretical hessian analysis. *arXiv preprint arXiv:2410.10986*, 2024.

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.

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.

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

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.

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

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.

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.

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.

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.

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.

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.

Hao Sun and Mihaela van der Schaar. Inverse-rlignment: Inverse reinforcement learning from demonstrations for llm alignment. *arXiv preprint arXiv:2405.15624*, 2024.

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.

Qwen Team. Qwen2.5: A party of foundation models, September 2024. URL https://qwenlm.github.io/blog/qwen2.5/.

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.

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.

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

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

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.

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.

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.

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.

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.

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.

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

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*, 2024b.

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.

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.

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.

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.

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*, 2024a.

Yushun Zhang, Congliang Chen, Tian Ding, Ziniu Li, Ruoyu Sun, and Zhi-Quan Luo. Why transformers need adam: A hessian perspective. *arXiv preprint arXiv:2402.16788*, 2024b.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *arXiv preprint arXiv:2306.05685*, 2023.

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 MORE RELATED WORK

Distribution matching forms the foundation of statistical machine learning (Murphy, 2012). The seminal work GAN (Goodfellow et al., 2014) introduced the concept of generative distribution matching in deep learning. A major challenge in this field is scalability (Brock, 2018), 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 (Chen et al., 2024; Li et al., 2024) 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) (Argall et al., 2009; Osa et al., 2018), where a learner makes decisions based on expert demonstrations. In fact, SFT can be reframed as IL with deterministic transitions (Sun & van der Schaar, 2024; Li et al., 2024). Specifically, the cross-entropy loss corresponds to behavior cloning (Pomerleau, 1991) in IL. Our framework is closely aligned with the generative adversarial imitation learning approach in (Ho & Ermon, 2016), which usually outperforms behavior cloning (Ke et al., 2019; Xu et al., 2020). A key aspect of this framework is correcting mistakes by rolling out trajectories. As discussed in Section 4.2, 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 $f_\theta$
```

```
 1 def gem_loss(logits, labels, beta=0.7, ignore_index=-100, h="linear"):
 2 
 3     shift_logits = logits[..., :-1, :].contiguous()
 4     shift_labels = labels[..., 1:].contiguous()
 5 
 6     mask = shift_labels != ignore_index
 7     shift_logits = shift_logits[mask]
 8     shift_labels = shift_labels[mask]
 9 
10     with torch.no_grad():
11         logits_on_labels = torch.gather(
12             shift_logits, dim=-1, index=shift_labels.unsqueeze(-1)
13         ).squeeze(-1)
14 
15         logits_diff = shift_logits - logits_on_labels.unsqueeze(-1)
16         if h == "linear":
17             weights = torch.ones_like(logits_diff)
18         elif h == "log_sigmoid":
19             weights = F.sigmoid(0.01 * logits_diff)
20         else:
21             raise ValueError(h)
22 
23     gene_log_probs = F.log_softmax(shift_logits, dim=-1)
24     q_probs = torch.exp(
25         F.log_softmax(shift_logits / beta, dim=-1)
26     ).detach()
27 
28     real_log_probs = torch.gather(
29         gene_log_probs, dim=-1, index=shift_labels.unsqueeze(-1)
30     ).squeeze(-1)
31 
32     loss = -torch.sum(
33         q_probs * weights * (real_log_probs.unsqueeze(-1) -
    gene_log_probs), dim=-1
34     ).mean()
35 
36     return loss
```

Listing 1: Pytorch Code of GEM

To understand the above implementation, we note that we leverage the gradient analysis in Section 4.2: first, we calculate the re-weighting term in Lines 9–20. Then, we calculate the difference in log-probabilities in Lines 22–33. Note that we use a coefficient of 0.01 to scale the input in the `log-sigmoid` function. This ensures that the function behaves nearly linearly.

Computational Complexity Analysis: We observe that the computational complexity of GEM is nearly equivalent to that of optimizing CE loss. To clarify, the computational cost of CE involves two primary steps: a forward pass through the Transformer to compute the distribution f_θ, followed by calculating the likelihood $\log f_\theta(y|x)$. The forward pass, which entails multiple layers of matrix multiplications, is the primary computational bottleneck.

Similarly, GEM requires one forward pass through the Transformer to compute the distributions q and f, and then calculate the relative difference $\log f_\theta(y^{\text{real}}|x) - \log f_\theta(y^{\text{gene}}|x)$. As with CE, the forward pass is the main bottleneck in GEM. Backpropagation is performed in a comparable manner for both methods, resulting in GEM achieving nearly the same training speed as CE.

In terms of memory consumption, GEM requires storing an additional distribution q, which occupies the same amount of memory as f. For example, in our setup with a batch size of 4, a sequence length of 2048, and a vocabulary size of 128k, q requires only about 2 GB of memory. This is negligible compared with the memory consumed by other training components such as gradients, optimizer states, and activation caches, which can collectively exceed 100 GB.

C PROOF

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., (Vieillard et al., 2020, 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.

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

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

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.

$\square$

D DISCUSSION

We discuss baseline strategies for mitigating overfitting in Appendix D.1 and Appendix D.2. Additionally, we emphasize the optimization challenges inherent in sequential data and the significance of the reset trick introduced in GEM, as detailed in Appendix D.3.

D.1 CE WITH WEIGHT DECAY

Weight decay is a widely used technique for mitigating overfitting, particularly effective in training convolutional neural networks (CNNs). However, we observed that it is not always effective when applied to training generative models with Transformers. We hypothesize two primary reasons for this discrepancy:

- **Architecture.** The structural characteristics of CNNs make them more homogeneous in design, resulting in a relatively uniform loss landscape across different parameter blocks within the network. In contrast, Transformers exhibit heterogeneous properties, as noted in recent works (Zhang et al., 2024b; Ormaniec et al., 2024), leading to significantly varied loss landscapes across parameter blocks. As a result, a uniform weight decay applied to all parameters in Transformers may be suboptimal, since different parameter blocks might require specialized weight decay strategies.
- **Task**. CNNs in classification tasks are designed to learn a predictor that outputs a unique prediction, whereas generative models aim to learn a distribution. For generative models, directly regularizing the distribution is preferable. Weight decay, which regularizes parameters indirectly, may not effectively serve this purpose.

Based on these observations, we argue that entropy regularization is better suited for training generative models with Transformers. Unlike weight decay, entropy regularization directly focuses on the target distribution, and its influence can effectively backpropagate to specific parameters through adaptive optimizers like Adam, accommodating the heterogeneity of Transformer architectures. A more deeper exploration of this topic is left for 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 1 but not Principle 2. We find that this formulation leads to an improper increase in tail probabilities when maximizing the entropy, as illustrated in Figure 6. In the context of LLMs, this increase often translates into nonsensical tokens in the vocabulary, leading to undesirable generation outputs. A concrete example is provided in Table 3, in which we directly sample sentences.

The core issue of CE with an entropy regularizer arises because **the gradient of the entropy regularizer can dominate for tokens with low probabilities**. Specifically, we have that

$$\frac{\partial - D_{\mathrm{KL}}(p,f)}{\partial f} = -\frac{p}{f}, \quad \frac{\partial \mathcal{H}}{\partial f} = -(1 + \log f).$$

where the division is element-wise. Consequently, for tokens with low probabilities in both f and p, i.e., $f(x) \approx 0$ and $p(x) \approx 0$, the gradient from the forward KL divergence, $-p(x)/f(x)$, could be of constant order. In contrast, the gradient from the entropy regularizer, $(-1 + \log f(x)) \rightarrow -\infty$ as $f(x) \rightarrow 0$. This imbalance disproportionately increases the tail probabilities, leading to undesirable effects.

In contrast, the proposed method, GEM, does not have this issue. This is because both distribution matching and entropy regularization in GEM are defined jointly over the generative distribution f, rather than being applied separately to the data distribution p and generative distribution f. As shown in the proof of Proposition 1, the gradient of GEM is given by:

$$\frac{\partial - \mathcal{L}}{\partial f} = \frac{-p + q}{f} = -\frac{p}{f} + \frac{q}{f},$$

where the first term, $-p/f$, is identical to that of CE with entropy regularization, but the second term q/f, is unique to GEM. Since $q = \texttt{softmax}(1/\beta * \log f)$, q is a more squeezed distribution than f. Consequently, for $f(x) \approx 0$, we have $q(x)/f(x) < 1$. This ensures that the gradient is not dominated in the low-probability (tail) region, preventing improper increases in tail probabilities. Thus, GEM achieves a more balanced optimization. This aligns with the analysis in Section 4.2, which shows that GEM's distribution matching activities are primarily focused on high-probability (mode) regions.

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. 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. [...]

D.3 GEM WITH SEQUENTIAL DATA

In this section, we extend the technical challenge in Section 4 regarding how to solve the problem in Equation (3). To recap, the problem is that

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

A key challenge is that the expectation $\mathbb{E}_{y_{1:T}}^{\texttt{gene}}[\cdot]$ cannot be calculated as easily as before. Worse still, Monte Carlo estimation as used in (Chen et al., 2024; Li et al., 2024), by drawing samples from the distribution, does not provide an accurate gradient estimate. A fundamental difficulty in this stochastic approximation arises from the distribution shift between the SFT data and the pre-trained distribution. To better understand this, refer to examples provided in Table 4. We observe that SFT data typically has finite-length sequences, while the pre-trained distribution produces samples that are repetitive and can even be infinite in length.

This causes issues: assuming the probability of any token is lower-bounded by a small number $c \in (0, 1)$, this means that $\log f(y_{1:T}^{\texttt{gene}}|x)$ approaches to $-\infty$ when T goes to infinite for pre-training distribution data. While this might seem acceptable as the gradient would reduce the probability of such samples, the challenge is that the sample size is vast: for the Llama-3-8B model, the vocabulary size is 128k, and with a typical sequence length of 2048, the sample space size is 128000^{2048}. This makes it difficult for the model to find effective directions for improvement. To validate this claim, we directly implemented the idea of stochastic approximation and found that training failed after 80 optimization steps (with 10k samples)[5], and the model could not generate good responses; see Table 4. In fact, techniques in (Chen et al., 2024; Li et al., 2024) are usually applied to models after SFT, where the distribution shift between the model and data is smaller. This technical remark is also discussed in the online forum `https://github.com/uclaml/SPIN/issues/26#issuecomment-2062926716`.

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 (Yu et al.,

---

[5] This method is computationally slow due to sampling responses. In fact, these computational resources of 80 steps are actually more than those required for GEM for 1 epoch tuning of 60k samples.

Table 4: Examples of generated sentences from pre-trained models. The symbol "[...]" indicates sentences have been omitted to save space for the ground truth label, and the repetitive sentences for the Llama-3-8B pretrained model. Examples illustrate the distribution shift between SFT data and the pre-trained distribution. In SFT data, ground truth labels have a finite length (marked by an explicit `<EOS>` in the example), whereas the pre-trained distribution often generates repetitive sentences without a clear endpoint.

Prompt	I would like to know more about Clyde Hill Washington...what can you tell me?
Ground Truth Label	Clyde Hill is a small, affluent city located in King County, Washington, United States. It is situated on the eastern shore of Lake Washington, north of Bellevue, and east of Kirkland. It is known for its stunning views of the lake and the Cascade Mountains. Clyde Hill was incorporated in the year 1953 and is home to approximately 2,800 residents. The city has a mix of residential and commercial areas, with large estates and luxurious mansions, making it one of the wealthiest ZIP codes in the United States. The median household income is around $200,000, and the median home value is roughly $1.6 million. [...] Overall, Clyde Hill, Washington, offers a unique and luxurious lifestyle to its residents. The combination of picturesque scenery, luxury living, and excellent amenities makes it a truly special place. `<EOS>`
Llama-3-8B-Pretrained	I would like to know more about Clyde Hill Washington...what can you tell me? I would like to know more about Clyde Hill Washington...what can you tell me? I would like to know more about Clyde Hill Washington...what can you tell me? I would like to know more about Clyde Hill Washington...what can you tell me? [...]
GEM without reset trick	Thank you for reaching out with your queries on Clyde Hill Washington! As of 2021, the estimated median home value in Clyde Hill, Washington, is $3,750,000. [...] Thank you for trusting me with your queries and I hope I was able to help. Stay in touch and talk to you soon! Hexatrigesimalerusform((___))stdexcepts každ □ [%://́);?¿¡/stdexcept
GEM	Clyde Hill is a small, affluent neighborhood located in the eastern part of Washington, D.C., bordering the Potomac River. It is known for its stunning views of the Washington Monument, the National Mall, and the Potomac River. The area is home to a mix of embassies, diplomatic missions, and luxury residential buildings. Clyde Hill is also home to the historic Clyde Hill House, which was built in 1929 and served as the official residence of the U.S. Ambassador to France from 1933 to 1946. The house is now a private residence and is not open to the public. [...] Overall, Clyde Hill is a unique and beautiful neighborhood that offers a glimpse into the history and elegance of Washington, D.C. `<EOS>`

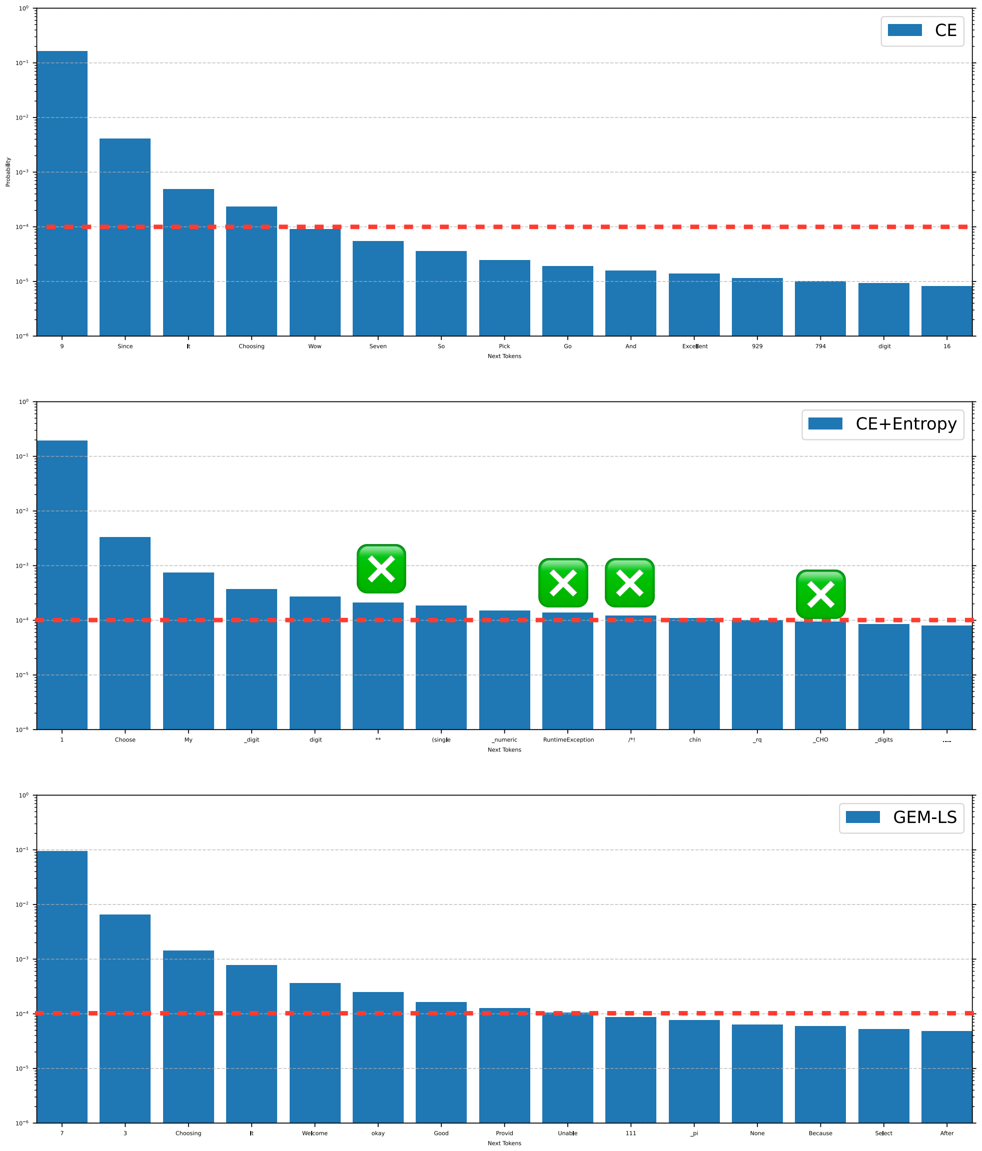


Figure 6: Distributions of next-token probabilities for trained models with the `UltraFeedback` dataset, presented from top to bottom: CE, CE+Entropy, GEM-LS. The prompt is “Give me a single-digit number”. The top 300 probabilities are shown with a subsampling rate of 20 for clear visualization. A red 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.

2023; Liu et al., 2023; Cui et al., 2024), the learning rate is set to 2×10^{-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 8, and 4 GPUs. The training times takes about 24 GPU hours for all methods. 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 (Jain et al., 2023).

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 (Zhou et al., 2023). 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) (Mostafazadeh et al., 2016), 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 (Kirk et al., 2023), 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. We highlight the top-k and top-p sampling strategies are important to address the tail probability issue of CE + Entropy.

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 (Yu et al., 2023), 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 (Wei et al., 2024):

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

---

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

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 (Wei et al., 2024), which contains 74,197 training samples and 1,000 test samples (randomly selected from the original training set). The maximum sequence length 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 (Yu et al., 2023). 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 5. 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 5: Evaluation perplexity and entropy. Models are trained with the `UltraFeedback` dataset.

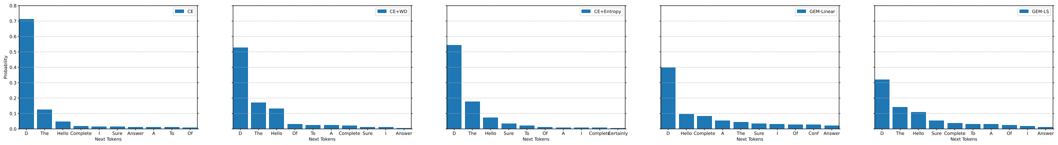
Method	**UltraFeedback**	
	Evaluation Perplexity	**Evaluation Entropy**
CE	3.48	0.68
CE+WD	3.46	0.68
CE+Entropy	3.78	**2.65**
NEFT	3.22	0.78
GEM-LS	3.18	1.19
GEM-Linear	**3.16**	1.16

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 7. We see GEM-trained models produce flatter distributions, indicating support for multiple possible answers.

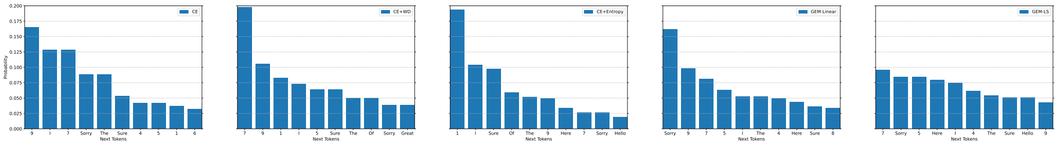
In-Context Learning and Alignment Tax. We assess the alignment tax by examining the performance drop in in-context learning abilities across six tasks: ARC, GSM8K, HellaSwag, MMLU,

---

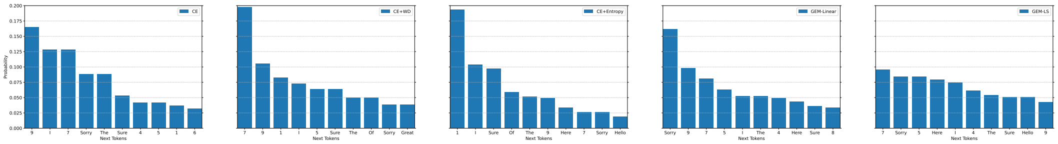
[7] For the first prompt, while "D" is the most likely answer, "A" could also be a valid response due to the pattern A, B, C, A, B, C, ...



(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 7: 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.

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 6. 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.

Table 6: 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**

Creative Writing. In addition to the diversity evaluation in Section 5.1, we also evaluated the writing quality of poems and stories using the LLM-as-a-judge framework (Zheng et al., 2023). The LLM judge assessed responses based on five criteria: helpfulness, relevance, accuracy, depth, and creativity, employing evaluation prompts from FastChat's judge prompts (`https://github.com/lm-sys/FastChat/blob/main/fastchat/llm_judge/data/judge_prompts.jsonl`). Specifically, we use the "single-v1" version. Each response was rated on a scale from 1 to 10. The evaluation encompassed 500 questions for both poem and story writing, with 16 responses per question, resulting in 96,000 responses across six methods. Scores were computed by calculating both the average and maximum scores across the 16 responses for each question, followed by averaging these scores across all 1,000 questions. To optimize evaluation costs, we utilized the open-source Llama-3.1-70B-Instruct model, a strong LLM suitable for assessing writing quality.

Table 7: Evaluation of writing quality of poems and stories using the LLM-as-a-judge framework (Zheng et al., 2023). Models are trained with the `UltraFeedback` dataset.

Method	Poem Writing		Story Writing	
	Average Score	Best Score	Average Score	Best Score
CE	6.87	8.12	6.90	8.17
CE+WD	6.83	8.11	6.86	8.16
CE+Entropy	7.06	8.31	7.08	8.29
NEFT	7.06	8.31	7.08	8.29
GEM-Linear	7.17	8.42	7.18	**8.40**
GEM-LS	**7.21**	**8.43**	**7.20**	8.39

Results, shown in Table 7, indicate that GEM outperformed baseline methods in both poem and story writing. This demonstrates that GEM not only enhances output diversity but also achieves strong generalization in creative writing tasks.

Chatting. We provide the reward score and associated win rate across different sampling budget in Table 8. We observe that even with less generation samples, GEM also shows better performance.

Table 8: 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**

In addition to utilizing the reward model as a judge, we also employ the LLM-as-a-judge approach (Zheng et al., 2023). The evaluation method is the same with the approach used for assessing poetry and story writing, as discussed earlier. The results, reported in Table 9, demonstrate that GEM achieves a higher quality score. While the LLM-as-a-judge method provides valuable insights, we acknowledge its limitations in distinguishing subtle differences between responses compared to the reward model evaluation used in this study. Importantly, the reward model is specifically trained for judgment tasks, making it a more robust and appropriate evaluation tool for our experiments.

Table 9: Evaluation of chatting response using the LLM-as-a-judge framework. Models are trained with the `UltraFeedback` dataset.

	CE	CE+WD	CE+Entropy	NEFT	GEM-Linear	GEM-LS
Average Score	7.09	7.12	7.05	7.10	7.14	**7.15**
Best Score	8.36	8.43	8.41	8.38	**8.47**	8.46

Math Reasoning. We evaluate the performance of majority voting and best-of-n methods across various sampling budgets, as reported in Table 10. The results demonstrate that GEM achieves comparable performance with significantly lower sampling budgets when employing majority voting. Notably, GEM shows strong consistency across votes: on average, the majority vote ratio across 32 responses is 52.6%, 53.4%, 53.3%, 53.3%, 53.0%, and 52.8% for CE, CE+WD, CE+Entropy, NEFT, GEM-Linear, and GEM-LS, respectively. This highlights GEM's ability to maintain consistency across votes.

Table 10: 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**

Code Generation. We present the pass rate performance across different sampling budgets in Table 11. The results indicate that GEM achieves comparable performance with significantly lower sampling budgets, often demonstrating a 2x improvement in efficiency.

Table 11: 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 12 to 14. The results indicate that GEM outperforms CE even with fewer generated samples.

Table 12: 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 13: 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

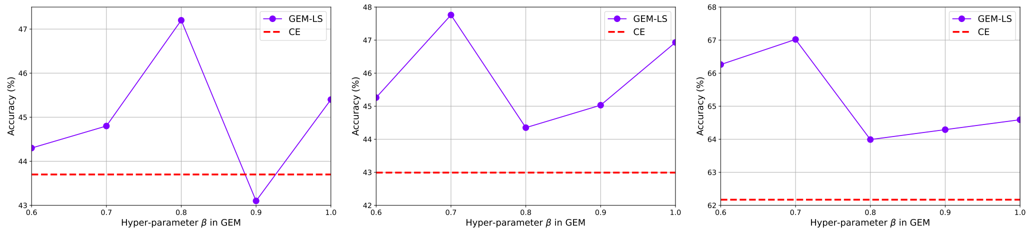
Table 14: Performance of pass rate on the code generation tasks `HumanEval` and `MBPP`. Models are trained with the `MagiCoder-OSS-Instruct` dataset.

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**

F.3 SENSITIVITY ANALYSIS OF β IN GEM

In this section, we conduct sensitivity analyses of the hyperparameter β in GEM. The typical range for β is between 0 and 1. As discussed in Section 4.2, GEM approaches the scenario without regularization as $\beta \to 1$, while $\beta \to 0$ represents a regime with strong entropy regularization. To assess its impact, we evaluate $\beta \in \{0.6, 0.7, 0.8, 0.9, 1.0\}$.

Our analysis focuses on three key metrics for models trained on the `UltraFeedback` dataset: (1) instruction-following performance on `IFEval` using greedy decoding, (2) reasoning performance on `GSM8K` using greedy decoding, and (3) reasoning performance on `GSM8K` using majority voting over 32 samples. The first two metrics reflect generalization performance, while the third highlights the benefits of improved diversity. The results, summarized in Figure 8, demonstrate that GEM is robust to variations in hyperparameter selection.

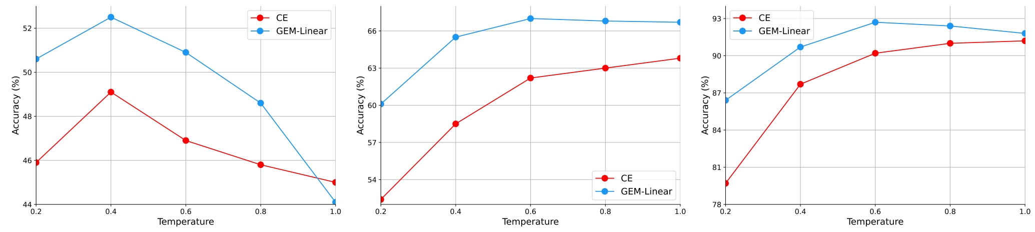


(a) `IFEval` (greedy decoding) (b) `GSM8K` (greedy decoding). (c) `GSM8K` (majority voting).

Figure 8: Sensitivity analysis of the hyperparameter β in GEM.

F.4 COMPARISON WITH TEMPERATURE ADJUSTMENT FOR DIVERSITY

Increasing the temperature is often used as an ad-hoc trick to enhance diversity at the inference stage. However, this approach can significantly increase tail probabilities, thereby raising the risk of errors. To address this, we avoid post-training adjustments to the temperature. Instead, we focus on carefully considering diversity during training, where training samples provide valuable guidance for achieving the desired level of diversity. An empirical analysis of temperature adjustment is presented in Figure 9.



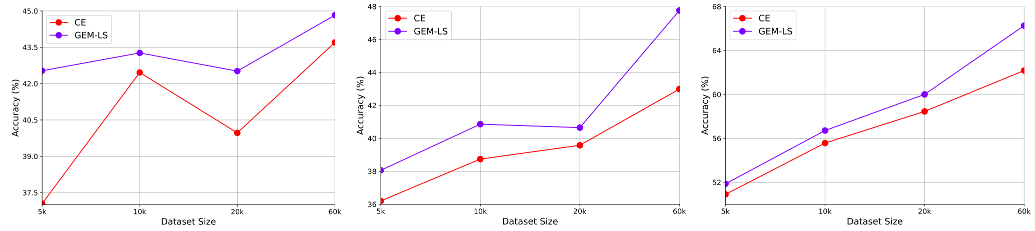
(a) `GSM8K` (random sampling) (b) `GSM8K` (majority voting) (c) `GSM8K` (best-of-n).

Figure 9: Performance by adjusting the temperature during inference. The models are trained using the `UltraFeedback` dataset.

From Figure 9, higher temperatures tend to degrade the performance of random sampling, while improving the performance of majority voting (across 32 responses) and best-of-n (across 32 responses), which eventually plateau. Notably, GEM consistently outperforms CE across random sampling, majority voting, and best-of-n, even when both methods are carefully tuned by adjusting the temperature.

F.5 PERFORMANCE IN LOW-DATA REGIME

In this section, we present results from training models with varying data sizes, focusing on the `UltraFeedback` dataset. Specifically, the dataset size is scaled from 5k, 10k to 20k, and 60k (total). For clarity, we report results using three metrics: instruction-following performance on `IFeval` with greedy decoding, and reasoning performance on `GSM8K` with both greedy decoding and majority voting (across 32 responses). The first two metrics evaluate generalization, while the third also assesses diversity. The results are summarized in Figure 10.



(a) `IFEval` (greedy decoding) (b) `GSM8K` (greedy decoding) (c) `GSM8K` (majority voting).

Figure 10: Performance by adjusting the training data size. The models are trained using the `UltraFeedback` dataset.

From Figure 10, we observe that GEM consistently outperforms CE across all data size configurations. This suggests that GEM is more effective at leveraging additional data to enhance both generalization and diversity.

F.6 RESULTS ON OTHER MODELS

In the main text, we presented numerical results on Llama-3-8B. Here, we extend our experiments to other architectures, selecting four representative models: Qwen2.5-3B (Team, 2024), Qwen2.5-7B (Team, 2024), Gemma-2-9B (Team et al., 2024), and Llama-3.1-70B (Dubey et al., 2024), all of which are recognized as strong performers. We evaluated six methods, including baseline approaches and our proposed techniques, by fine-tuning these pre-trained models on the `UltraFeedback` dataset. For all models, we applied the same configuration described in Appendix E, with one exception:

for Llama-3.1-70B, we used LoRA with a rank of 16 and a dataset size of 10,000 due to limited computation resources.

For evaluation, we selected five criteria: (1) greedy decoding performance on `IFEval` for instruction following, (2) greedy decoding performance on `GSM8K` for math reasoning, (3) best-of-n win rate in chatting, (4) majority voting in `GSM8K`, and (5) pass rate in `HumanEval` for code generation. The first two metrics focus on generalization, while the last three evaluate test-time scaling and diversity. Except for Llama-3.1-70B, we use a smaller generation budget of 8 due to limited computational resources, while elsewhere we use a generation budget of 32. Results are reported in Figure 11.

From Figure 11, we observe that GEM consistently outperforms CE across various models, while baseline methods often fail to provide consistent improvements over CE. In particular, when fine-tuning Llama-3.1-70B using NEFT (Jain et al., 2023), we observed extremely poor performance, likely caused by noise introduced in the embeddings. We hypothesize that this failure occurs because the embedding layer is not trainable under the LoRA configuration. Consequently, we have excluded its performance results from Figure 11. In many cases, GEM also exceeds the performance of the baseline methods. Notably, GEM achieves significant gains on specific tasks; for instance, on Qwen2.5-3B, the MV@32 performance on `GSM8K` improves by an impressive 17.7 points.

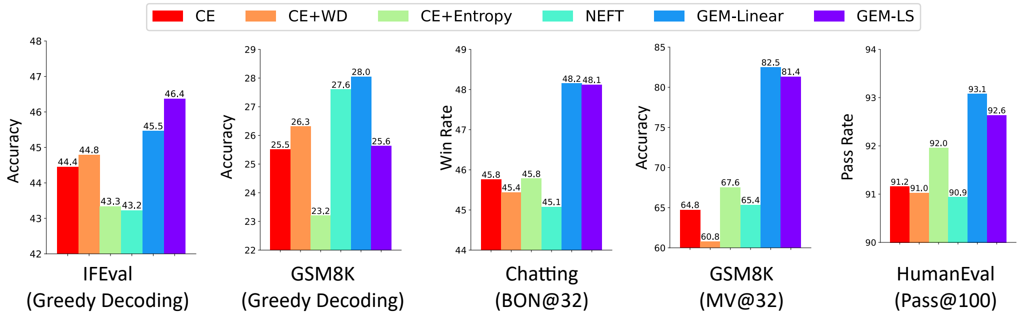
G FUTURE WORK

Our formulation of entropic distribution matching, along with the practical GEM algorithm, extends its applicability beyond SFT. Below, we explore its broader potential and provide detailed discussions on its poential applications.

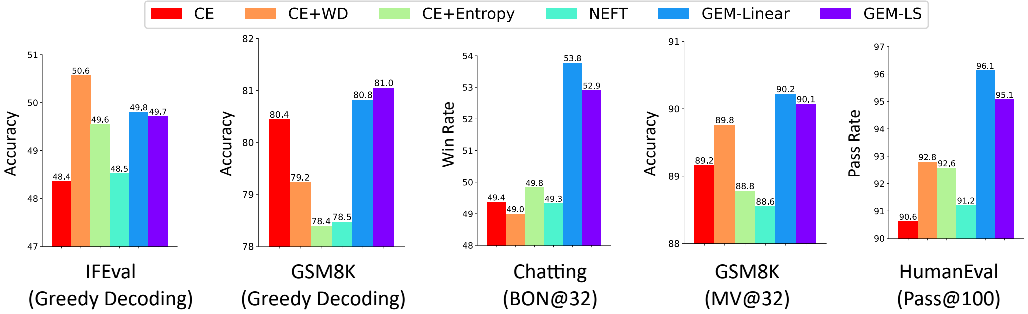
Preference Collapse in RLHF. SFT-trained models can be refined through Reinforcement Learning from Human Feedback (RLHF) to better align with human values (Ouyang et al., 2022; Bai et al., 2022). In this context, Xiao et al. (2024) studied the impact of SFT models on preference learning in RLHF, demonstrating that if an SFT model collapses (i.e., becomes biased toward certain outputs with near-certain probability), it can further lead to preference collapse in alignment. Their findings underscore the importance of addressing collapse during the SFT stage.

Synthetic Data Generation. SFT-trained models are often used as synthetic data generators for self-improvement (see, e.g., (Adler et al., 2024; Dubey et al., 2024)). In this context, maintaining output diversity is essential. By generating a wide range of diverse outputs, models can explore various potential solutions, reducing the risk of overfitting and uncovering better-performing strategies. Our experiments about best-of-n against a reward model Section 5 is inline of this topic.

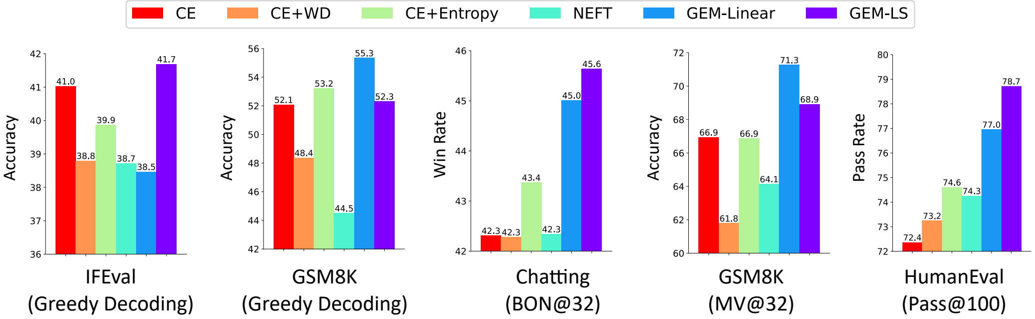
Mode Collapse. When models are repeatedly fine-tuned on text generated by their predecessors, linguistic diversity gradually erodes. This recursive process amplifies existing errors and biases, with each successive generation inheriting the limitations of its predecessors. This phenomenon, known as mode collapse, has been extensively documented in prior studies (Guo et al., 2023; Shumailov et al., 2023). By introducing entropy regularizer, we expect to help mitigate mode collapse in the self-improvement.



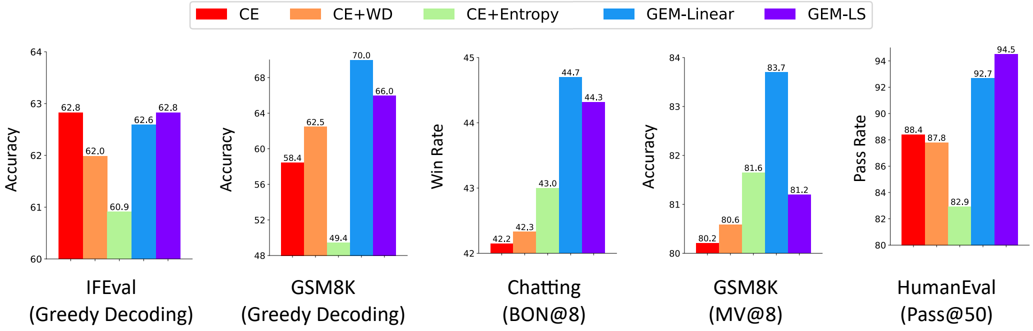
(a) Qwen-2.5-3B



(b) Qwen-2.5-7B



(c) Gemma-2-9B.



(d) Llama-3.1-70B.

Figure 11: Performance across different architectures. The models are trained using the `UltraFeedback` dataset.