Few-shot, Interpolation-based Style-conditioned Text Generation using LLMs

by

Moemen Gaafar

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

Abstract

This thesis addresses the task of few-shot style-conditioned text generation using large language models (LLMs). We propose a novel, model-agnostic approach for adapting LLMs to arbitrary styles using a few text samples from a certain author. Instead of using pre-defined features, our method represents style directly in terms of model weights and employs a Variational Autoencoder (VAE) to construct a latent space of these weights, allowing for a generic style representation.

We investigate whether style features can be generically extracted from LLM weights, if a salient latent space can effectively encode authorial styles, and whether an interpolation strategy can extract novel finetuned models for low-resource authors. Our approach is evaluated on three datasets, comparing it to naive finetuning and prompting techniques.

Results show that our method outperforms and is more reliable than naive finetuning in low-resource settings based on automatic metrics. While our method outperforms prompting for some LLMs using a low number of text samples, its performance still does not consistently exceed that of prompting, especially as the number of available text samples increases. This work contributes to controllable text generation by introducing a weight space interpolation technique for few-shot style adaptation and demonstrating that model weights can directly represent text style, providing insights for future research in this area.

Preface

This thesis presents an original work by Moemen Gaafar under the supervision of Dr. Matthew Guzdial. This work may be restructured to get published under different research venues in the near future. Generative AI has been used sporadically in the implementation of this work, mainly to generate boilerplate code.

Acknowledgements

The two years I took to finish my Master's degree were full of highs and lows. I would like to acknowledge all those who stood by me as I went through this journey. First and foremost, I would like to thank my supervisor Dr. Matthew Guzdial for his guidance and support during these two years. I learned a lot from my interactions with Matthew on both the academic and personal levels and I am grateful for our time working together. I would like to thank Alaa Alajmy who was there for me from the very beginning to the very end. We started this journey together and we grew so much along the way. I hope the future is kinder and brighter for both of us.

I would not have been able to get through these two years without the friends that I made along the way. I am full of gratitude for Mohamed Elsayed, Yazeed Mahmoud, Esraa Eleimey, Lucas, and Kian Razavi. I am also grateful to the wider Edmonton community that became my second home away from home. Organizations like Food Not Bombs, Rapid Fire Theatre and Fringe Theatre gave me the opportunity to give back to this community and to meet a lot of wonderful like-minded people. I am lucky to have come across them and I hope they continue to grow and affect more lives in the future.

My deepest appreciation extends to my friends around the world, Diaa Eldin Malek, Belal Magdy, Fahmy Ahmed, Mohamed Mahrous, Ahmed Ali, Omar Ali, Mohamed Kasem, Mohamed Mostafa (Bakkar), Hossam Arafa, Ahmed Hisham, Asmaa Ibrahim, Mohamed Chaffei, Mahmoud Ashraf, Mohamed Alasmar, Omar Hegazy, Mohamed Ashraf, Seif Tarek, Ahmed Elghandour, Omar El Gammal, Ghada Ali, Ahmed Magdy Ragab, Ammar Shehata, Toka Alokda, Mohanad ElAbd, Omneya Adel, Afnan Sultan, Hazem Abu-Bakr, Youssef Ahmed, MAG, and Ziad Elkomy, who, despite the distances, will always have their places in my heart. Thank you for being one call or text message away. I hope the universe is kind enough to bring us all back together. For now, I am grateful that two of my friends, Abdelrahman Elaraby and Khaled Elbastawisy, are joining the University of Alberta community this year and are bringing with them a taste of back home. My appreciation also goes to my parents and siblings whose calls were a source of relief amidst stressful times.

Finally, I would like to thank the department of Computing Science for giving me this opportunity to learn and grow. Moving here was the start of a very significant chapter in my life that I am still learning to navigate through.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Text generation is a core task in Natural Language Processing (NLP) research that powers applications such as machine translation, summarization and question answering [1]. Controllable text generation is a subfield of NLP that aims to guide text generation systems towards outputs that satisfy a certain set of constraints [2]. Since transformer-based large language models (LLMs) have become the current state-of-the-art in many NLP tasks, research in this area has become of great importance due to the difficulty of reliably controlling their output [2].

Style-conditioned text generation is the task of constraining the generated text to a certain writing style, usually corresponding to a specific author [3]. Style-conditioned text generation in LLMs is particularly challenging due to the prohibitive training data requirements of LLMs [2]. Successfully style-conditioning LLM text generation with a few samples holds the potential for new applications such as real time adaptation, and empowering end users to produce text matching their writing style.

Text generated through LLMs can be controlled either through prompting, finetuning, or postprocessing [2]. While prompting is the least demanding approach computationally, previous work has shown that it is incapable of reliably inferring style from a few text samples [4], [5]. On the other hand, finetuning, while theoretically capable of adapting the model to arbitrary styles, requires collecting a sizable corpus to be effective, even using low-resource techniques, such as Low Rank Adaptation (LoRA) [6]. Finally, postprocessing techniques, which modulate the output probabilities of an LLM, have been shown to be capable of few-shot style-conditioned text generation [7]. However even these reduced training data requirements might prove burdensome in some applications, such as applications where a user produces text in real time. In addition, some of them assume the existence of predefined style features, such as punctuation frequency, ratio of uppercase to lowercase characters, and n-gram word counts [8].

In this work, we propose a novel model-agnostic approach capable of performing few-shot adaptation of an LLM to a target style. We differ from prior approaches in representing style in terms of a model's weights rather than secondary features, and employing a Variational Autoencoder (VAE) to construct a latent space of the differences in the model weights, which we call weight deltas. In our implementation, we extract these weight deltas using Low-Rank Adaptation (LoRA). This way, our approach does not assume the availability of predefined style features and leaves it to the VAE to find similarities among the finetuned models. We argue this allows our approach to be more adaptable in terms of style representation. During inference, we run an interpolation process that picks the best model weights corresponding to the provided text samples.

1.2 Research Questions & Contributions

In this thesis, we aim to answer the following research questions regarding few-shot style-conditioned text generation:

R1 Can we use a Variational Autoencoder to construct a latent space that is capable of encoding authorial styles using finetuned LLM weights?

R2 Using this latent space, can we employ an interpolation strategy to find LLM weight deltas that produce LLMs that are more capable of generating text in a target authorial style compared to baselines using a small number of text samples from unseen authors?

Guided by these questions, our work presents the following contributions:

1. We propose a model-agnostic approach capable of performing few-shot stylized text generation. Based on automatic metrics, we find that our approach outperforms and is more reliable than naive finetuning in low-resource settings. However, our evaluations show that it is still incapable of reliably outperforming prompting given sufficiently large models.

2. We show that model weights can be used directly as representation for text style. Without extra information outside of the model weights, we find that the latent space constructed by the VAE produces embeddings corresponding to text style.

1.3 Thesis Outline

In this chapter, we provided a brief introduction into the problem of few-shot style-conditioned text generation. The rest of this thesis is organized as follows:

1. **Chapter 2: Background** presents a detailed background that surveys the main concepts behind Artificial Neural Networks, Language Models, and Transfer Learning.

2. **Chapter 3: Literature Review** presents a literature review of previous work approaching low-data controllable text generation with LLMs. We also focus on differentiating our problem from the closely-related one of style transfer.

3. **Chapter 4: System Overview** presents our proposed system and the details of our implementation.

4. **Chapter 5: Evaluation and Results** presents a detailed evaluation on three distinct datasets.

5. **Chapter 6: Conclusions and Future Work** shows a discussion of our results and ends with our suggestions for future work.

In addition to these chapters, we include two appendices. Appendix A presents a case study that showcases the functionality of our proposed system. Appendix B presents more evaluation results under different settings than the ones presented in the main text.

Chapter 2

Background

In this chapter, we introduce the concepts necessary to understand the work presented in this thesis. First, we cover Artificial Neural Networks as the basic computational model that we use in this thesis. Second, we introduce the Variational Autoencoder architecture since it is the component responsible for the latent space required for our system. Third, we cover Language Models with an emphasis on transformer-based Large Language Models. Finally, we cover the field of transfer learning as it is the area of research our work falls under.

2.1 Artificial Neural Networks

One of the most important features of computers is their ability to reliably store and rapidly manipulate data. Utilizing these capabilities, it is possible to build predictive models that leverage existing data to make predictions about unseen data. Artificial Neural Networks (ANNs) are computational models, loosely inspired by biological brains, that can iteratively fit a given corpus of data [9]. The basic building block of ANNs is the neuron, also know as the perceptron, shown in Figure 2.1. Given the input features $x_0, x_1, x_2, ..., x_N$, the output y of the perceptron is calculated as follows:

$$y = f(\sum_{i=0}^{N} w_i * x_i + b) \tag{2.1}$$

where $w_0, w_1, w_2, ..., w_N$ are the weights of the pereceptron, b is the constant

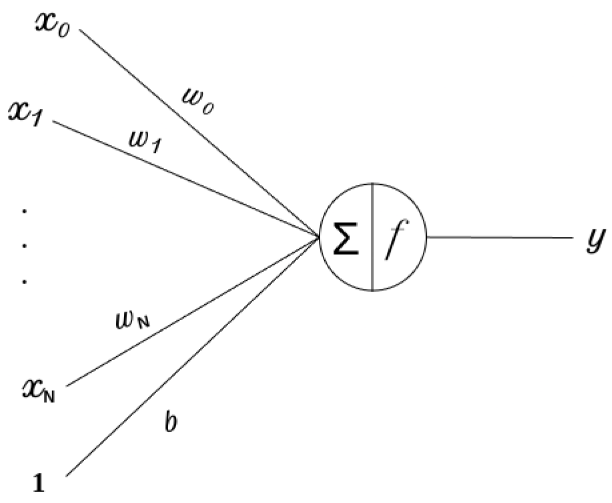


Figure 2.1: A single perceptron.

bias term, and f is the activation function. The weights and the bias are the learnable parameters of the perceptron. The activation function f is commonly set to a non-linear function, such as ReLU or softmax. This allows the perceptron to model non-linear transformations.

A neural network is a set of perceptrons connected in a certain orientation, commonly referred to as an architecture. Figure 2.2 shows the feedforward neural network architecture, in which the perceptrons of each layer are connected to all perceptrons of the next layer. Without loss of generality, the layers of a neural network can be categorized as:

1. **Input Layer**: The first layer of perceptrons in the neural network which receives the raw inputs.
2. **Hidden Layer(s)**: One or more layers of perceptrons that further process and iteratively manipulate the inputs.
3. **Output Layer**: The final layer of perceptrons in the network that calculates the network prediction given the processed inputs from the hidden layers.

To train a neural network, we must define a loss function in order to quantitatively measure the difference between the network prediction $\hat{y}$ and the expected output y. For instance, the mean squared error loss is calculated as:

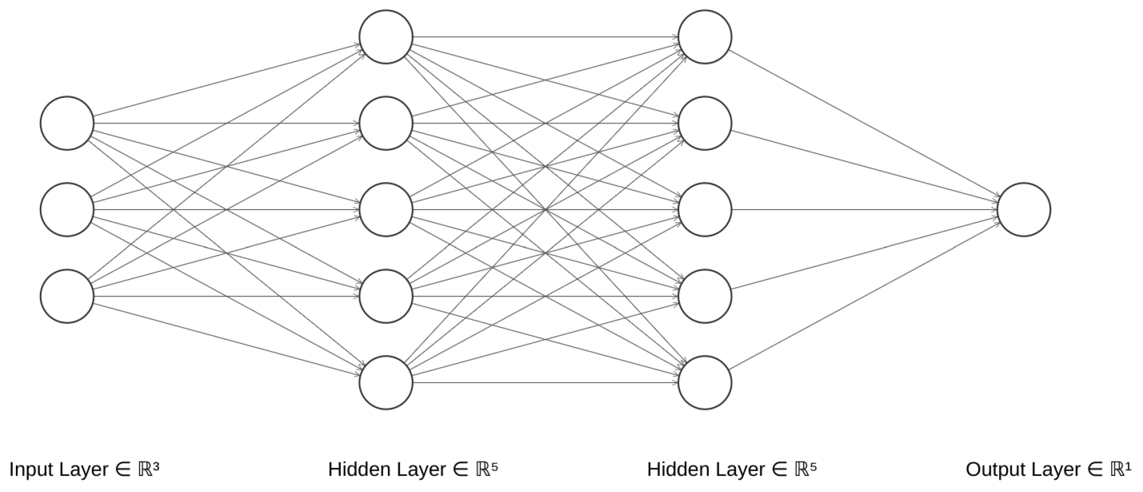


Figure 2.2: Feedforward Neural Network.

$$Loss = \|y - \hat{y}\|^2 \tag{2.2}$$

The network parameters are then updated according to the value of the calculated loss. This process is called backpropagation, where each parameter is updated according to its contribution to the final prediction, and hence to the loss, through a series of partial derivatives.

Multiple other neural network architectures exist besides feedforward networks, each capable of modelling different types of input and output data. In the next subsection, we explore sequence-to-sequence neural network architectures which are capable of processing sequences of data as inputs and generating sequences of data as outputs.

2.1.1 Sequence-to-Sequence Models

Some data types, such as audio and text, inherently contain time-dependent information, and so are better treated as sequences. For sequence data, the output y_n at time step t_n might not only depend upon the inputs X_n at that time step, but also on $X_{n-1}, X_{n-2}, \ldots$ and possibly $y_{n-1}, y_{n-2}, \ldots$ as well. Recurrent Neural Networks (RNNs) are a family of neural network architectures that are capable of handling sequence data as shown in Figure 2.3. RNNs differ from feedforward networks in that the outputs of the hidden layers from

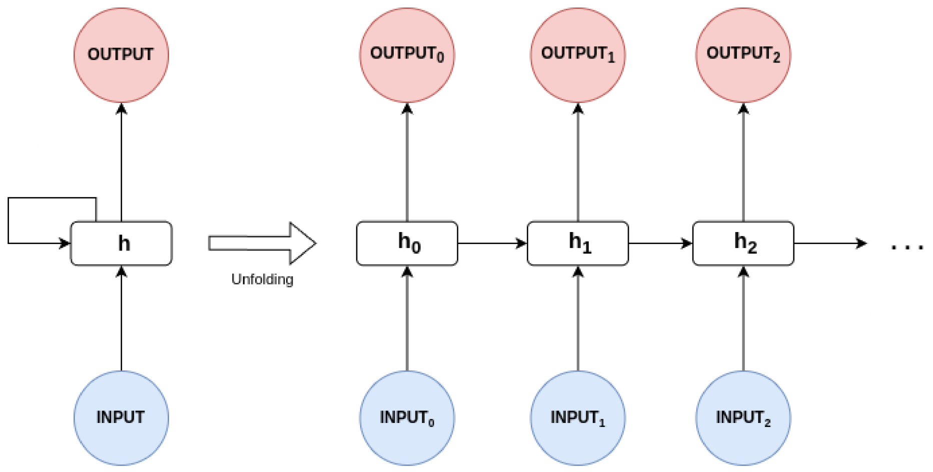


Figure 2.3: Recurrent Neural Network.

the previous time step are fed back again as inputs to the hidden layers when generating the outputs at the current time step. This way, the network possesses an internal state that is updated with each subsequent part of the input sequence. At the same time, the RNN uses this internal state to generate the output sequence. When both the input and the output are sequences, the RNN is considered a sequence-to-sequence model. Since the output of RNNs do not only depend on the inputs at the current time step, a variation of backpropagation, called backprobagation through time, is applied to calculate the model gradients during training. We refer interested readers to Goodfellow et al. for more details on this process [9].

For some applications, such as machine translation, it is usually preferred to process the input sequence as a whole and to start generating the output sequence later in order to consider dependencies among parts of the input sequence. In this case, one recurrent network, called the Encoder, processes the input sequence and outputs a hidden state, called the context vector. After that, another recurrent network, called the Decoder, takes this context vector as an input and sequentially generates the output sequence. At each time step, in addition to the context vector, the decoder is conditioned on its output from the previous time step. This is known as the Encoder-Decoder architecture, shown in Figure 2.4. While initially applied to RNNs, the Encoder-Decoder architecture later become a generic formalism that can be applied to different

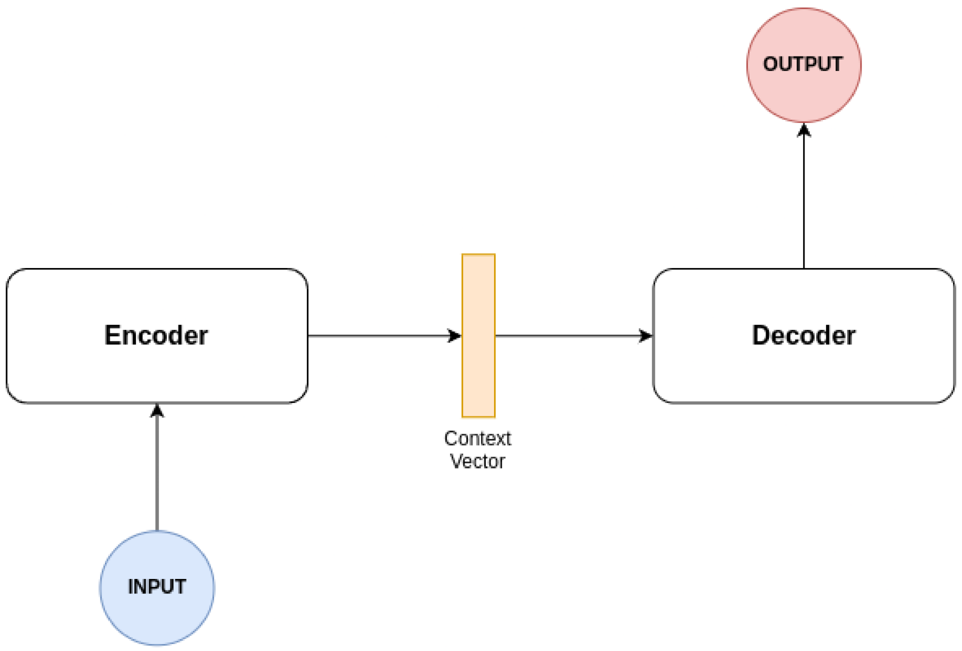


Figure 2.4: The Encoder-Decoder Architecture.

neural network architectures, such as LSTMs [10] and GRUs [11]. We will not cover these architectures here since they are not relevant to this thesis but interested readers can refer to Goodfellow et al. for more details [9]. In section 2.3.1, we discuss transformer-based encoder-decoder architectures as they are the basis of the generative models in this thesis.

2.2 Variational Autoencoders

As a central component to our proposed system, in this section, we discuss Variational Autoencoders (VAEs) and how they operate. However a background about standard autoencoders is required as a basis for our discussion of VAEs. An autoencoder is a neural network architecture designed to output the same data it receives as input. The goal of employing autoencoders is to discover a low-dimensional latent space that is capable of representing high-dimensional data [9]. As shown in Figure 2.5, Autoencoders are composed of an encoder network e that compresses the inputs into low-dimensional vectors in the latent space, and a decoder network d that reconstructs the inputs using latent vectors. Hence, the autoencoder can be expressed as follows:

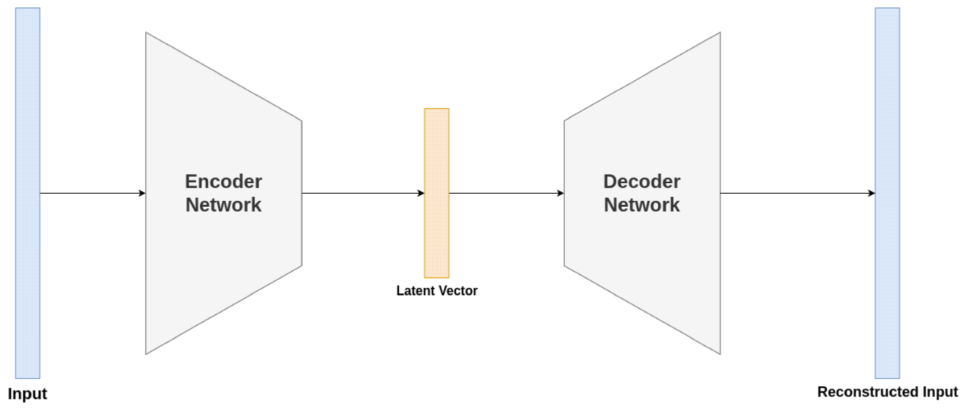


Figure 2.5: Autoencoder Architecture.

$$z = e(x) \tag{2.3}$$

$$\hat{x} = d(z) \tag{2.4}$$

where z is the latent vector that represents x in the latent space, and $\hat{x}$ is the reconstructed version of x. Thus, the mean squared error function for the autoencoder can be written as:

$$Loss = \|x - \hat{x}\|^2 \tag{2.5}$$

This is known as the reconstruction loss because it measures the ability of the autoencoder to reconstruct its inputs.

Although autoencoders excel at reconstructing complex input data, they perform poorly when generating new data. The reason for that is that the latent spaces learned by autoencoders tend to be fragmented and lack meaningful structure, as their primary objective is only to reconstruct inputs. Consequently, these latent spaces are ineffective for interpolation and generation [12]. Variational Autoencoders (VAEs) solve this problem by imposing structural constraints on the latent space representations of the input data, and thus allow for meaningful sampling.

Instead of deterministically encoding each input data point x to a unique latent space vector z, VAEs encode input data as probability distributions in the latent space. For simplicity and ease of representation, these probability distributions typically take the form of Gaussian distributions. Latent vectors are then sampled from these distributions and passed to the decoder for reconstruction, just like in a regular autoencoder. Accordingly, the VAE can be expressed as:

$$\mu_x, \sigma_x = e(x) \tag{2.6}$$

$$z \sim N(\mu_x, \sigma_x) \tag{2.7}$$

$$\hat{x} = d(z) \tag{2.8}$$

where μ_x and σ_x are the mean and covariance matrices defining the Gaussian distribution corresponding to x in the latent space, and $N(\mu, \sigma)$ is the Gaussian distribution with mean μ and standard deviation σ. This process is summarized in Figure 2.6. Encoding input data as distributions instead of point vectors in the latent space aims to create a continuous and connected latent space. This continuity enables meaningful interpolation and sampling within the latent space, facilitating the generation of new coherent data points.

However, this approach alone is inadequate to consistently construct a latent space suitable for interpolation and generation. Even though input data are projected as Gaussian distributions, it is still possible that the encoded distributions might be widely dispersed in the latent space, or might collapse into point vectors. Consequently, the VAE cannot be effectively trained using only the reconstruction loss, and so a regularization term is incorporated into the loss function. This term enforces that the latent distributions' means and standard deviations approximate a Gaussian distribution with a mean of 0 and a standard deviation of 1. Thus, the VAE loss function is expressed as:

$$Loss = \|x - \hat{x}\|^2 + KL(N(\mu_x, \sigma_x), N(0, 1)) \tag{2.9}$$

where KL stands for the Kullback–Leibler divergance function, which quantifies the divergance between two distributions. In this case, it is used to

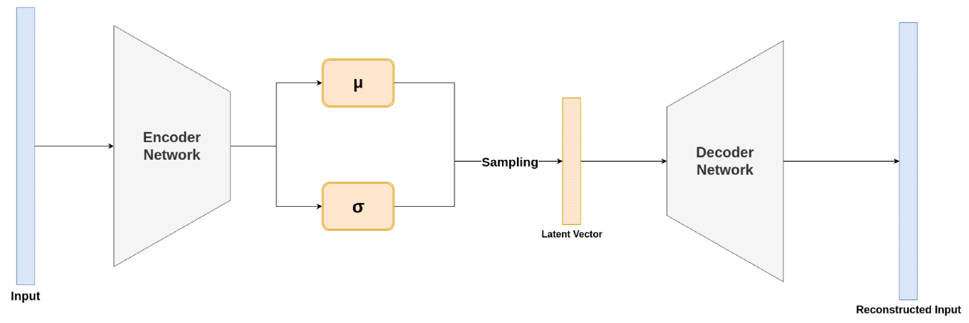


Figure 2.6: Variational Autoencoder Architecture.

measure the divergence between the output distribution from the encoder and the Gaussian distribution with a mean of 0 and a standard deviation of 1 [12].

In this thesis, we utilize the modelling capabilities of VAEs to construct a salient latent space that encodes information about authorial styles using LLM weight deltas and allows us to interpolate to find new LLM weight deltas. We discuss this in details in Chapter 4.

2.3 Language Models

Language models are probabilistic models that learn the distribution of tokens in a certain text corpus which enables them to generate more text from the learned distribution. Prior to training a language model, the sentences in the training corpus are split into tokens, where each token corresponds to a word or a subword. All the unique tokens that a language model can process are known as its vocabulary. A language model estimates the probability of each token in the corpus given the prior tokens of the same sentence.

While state-of-the-art language models employ neural network architectures, language models can be implemented using any machine learning models, including probabilistic models such as Markov models. n-gram Markov models are perhaps the simplest implementation of a language model. To make the language modelling problem tractable, these models assume that the probability of the nth token in a sentence depends only on the prior $n-1$ tokens, which is known as the Markov assumption. A trained n-gram Markov model can be represented as a probability distribution table that shows the probability of all tokens in the vocabulary given the prior $n-1$ tokens. To train an n-gram model, we count the number of occurrences of each token given the prior $n-1$ tokens and then divide it by the total number of occurrences of the prior $n-1$ tokens. This can be expressed as:

$$P(y_t) = \frac{C(y_{n-t}, y_{n-t+1}, ..., y_{t-1}, y_t)}{C(y_{n-t}, y_{n-t+1}, ..., y_{t-1})} \tag{2.10}$$

where C is a counting function and y_t is the t-th token in the sentence. Despite their simplicity, n-gram language models are not very practical since the size of the probability distribution table increases exponentially as n increases. They also do not encode any semantic information about the tokens and so are incapable of learning linguistic relationships among tokens.

Neural networks have been shown to be capable of learning embedding vectors that can represent the semantic and contextual meaning of tokens [13]. The number of dimensions of the embedding space is set to be much smaller than the size of the vocabulary. This salient compression allows neural networks, especially encoder-decoder models, to perform complex language tasks without an exponential explosion in model size.

2.3.1 Transformers

Despite the effectiveness of RNN-based encoder-decoder models, they suffer from a structural problem that limits their capabilities: the information bottleneck imposed by the context vector. As the fixed-size context vector produced by the encoder iteratively encodes more and more information as it processes a sentence, it loses its ability to retain the necessary context information needed by the decoder. To mitigate this issue, a new mechanism was proposed for the encoder-decoder architecture that allows the decoder to attend to *all* the hidden states of the encoder, not only the last one. This mechanism is known as the attention mechanism [14].

The Attention Mechanism

The aim of the attention mechanism is to construct a context vector with the relevant information for the decoder at each time step using the encoder's hidden states. This is generally done by learning attention weights that are used to average the encoder hidden states and calculate the output. The context vector at time step t is thus calculated as follows:

$$C_t = \sum_j \alpha_{t,j} h_j \tag{2.11}$$

where h_j is the hidden state of the encoder after the j-th input token. The attention weights are learned during training using the encoder hidden states and a subset of the decoder hidden states. Bahdanau et al. [15] used the decoder hidden state from the previous time step, s_{t-1}, for this calculation, and so the attention weights were calculated as follows:

$$\alpha_{t,j} = softmax(a(s_{t-1}, h_j)) \tag{2.12}$$

where $a(\cdot)$ is the alignment model. For Bahdanau et al. [15], the alignment model is a neural network component designed to learn and compute attention weights for each word in the source sentence during the generation of the target sentence. Luong et al. [16] used the decoder hidden state from the current time step, s_t, for calculating the attention weights instead, and used a simpler alignment calculation based on the dot product of s_t and h_j. This is known as Global Attention or Scaled Dot-Product Attention.

The attention mechanism improved the performance of RNN-based encoder-decoder models on various language processing tasks. This success led to it becoming the basic building block for the transformer architecture.

Self-Attention and the Birth of the Transformer

Vaswani et al. [14], aiming to improve the performance of neural networks in the task of machine translation, proposed the transformer architecture which does away with RNNs and relies primarily on the attention mechanism. The transformer is an encoder-decoder neural architecture, where the encoder and decoder are composed of stacked identical encoder and decoder units. Since there is no recurrence in the transformer architecture during training or at inference time, it proved to be a highly parallelizable architecture. However, this meant that there are no encoder or decoder hidden states to perform attention calculations. Instead, Vaswani et al. [14] proposed a new attention mechanism, called self-attention. For each input token in a sentence, instead of updating the hidden state of the encoder, self-attention computes a new representation of the token that incorporates information from all other tokens in the sentence.

Each self-attention layer consists of three main parameter matrices W_q, W_k, W_v which are learned during training. Given a sentence $x = (x_0, x_1, x_2, \ldots, x_n)$, to calculate the self-attention output for the token x_t, first we compute its query:

$$q_t = x_t W_q \tag{2.13}$$

Then, for each token in the sentence, including x_t itself, we calculate the keys and values:

$$k_i = x_i W_k, \quad \forall i \in \{0, 1, 2, \ldots, n\} \tag{2.14}$$

$$v_i = x_i W_v, \quad \forall i \in \{0, 1, 2, \ldots, n\} \tag{2.15}$$

Using the query, keys and values, the self-attention output for x_t is calculated as follows::

$$a_t = \sum_i softmax(\frac{q_t \cdot k_i}{\sqrt{d_k}})v_i \tag{2.16}$$

where d_k is the number of dimensions of the key vectors. The final output of this process is a list of vectors $A = (a_0, a_1, a_2, \ldots, a_n)$, each corresponding to one of the tokens in X, which becomes the input for the next encoder or decoder unit.

Each encoder unit of the transformer is composed of a self-attention layer and a feedforward layer. Each decoder unit is composed of a masked self-attention layer, an encoder-decoder attention layer, and a feedforward layer. The attention layers in the decoder have slight variations to the self-attention mechanism described above. The masked self-attention layer only attends to tokens that come before the current token, $x_{<t}$. Since the decoder generates the output tokens one by one, it is not supposed to "look ahead" of the current time step. The encoder-decoder attention layer performs a similar function to the context vector, injecting information from the encoder into the decoder. Instead of calculating its own queries, keys and values matrices, the encoder-decoder attention layer takes the keys and values matrices of the last encoder unit of the encoder, and the queries matrix of the masked self-attention layer in the same unit.

The transformer architecture additionally employs a residual layer normalization technique across each of its layers to enhance training stability by facilitating smoother information flow and to mitigate issues like vanishing gradients. This method involves adding the input directly to its output and normalizing activations across feature dimensions. Finally, transformers use positional encoding to inject sequence order information into the model because, unlike recurrent architectures, transformers lack inherent sequential understanding due to their parallel nature. Positional encoding addresses this by embedding positional information directly into the input embeddings, enabling effective handling of tasks requiring sequence understanding.

Large Language Models

Transformer-based models have become state-of-the-art for many language tasks due to their parallelizable nature and the effectiveness of the self-attention mechanism. By scaling the transformer architecture, language models started achieving impressive generalization capabilities across different tasks. For example, T5, an encoder-decoder transformer-based model, was capable of performing machine translation in addition to summarization and question answering [17]. Such versatile language models are commonly known as foundational language models, or large language models (LLMs).

While the original transformer architecture was designed with an encoder-decoder structure for machine translation, other tasks often require further modifications. For instance, BERT is an encoder-only language model designed to learn strong token embeddings, and so it does not need a decoder since it does not generate output sequences [18]. In contrast, the GPT family of models employs a decoder-only architecture since their primary goal is text generation and completion. Decoder-only architectures remove the encoder-decoder attention layer in the decoder blocks. Surprisingly, with sufficient scaling of model size and training data, decoder-only LLMs, such as GPT-3, were found capable of performing tasks that previously required encoding such as machine translation and summarization.

2.4 Transfer Learning

Transfer learning is a framework in which knowledge learned in one domain is adapted to perform a task in another domain. In the context of deep learning using ANNs, the transfer learning framework can be expressed in terms of five components:

- Source Task ($\mathcal{T}_s$): The task that the neural network is originally trained to perform.
- Source Dataset ($\mathcal{D}_s$): The data used to train the neural network for the source task.
- Predictive Function ($f_{\mathcal{T}}$): A neural network trained to perform the source task using the source data.
- Target Task ($\mathcal{T}_t$): A new task that the neural has not been trained to perform.
- Target Dataset ($\mathcal{D}_t$): The data which can be utilized by a neural network to learn the target task.

Deep transfer learning is then defined as the task of improving the performance of $f_{\mathcal{T}}$ on task $\mathcal{T}_t$ using the dataset $\mathcal{D}_t$ where $\mathcal{T}_t \neq \mathcal{T}_s$ and/or $\mathcal{D}_t \neq \mathcal{D}_s$ [19]. In addition, the size of $\mathcal{D}_t$ is usually much smaller than $\mathcal{D}_s$.

Multiple methodologies exist to perform deep transfer learning. We focus mainly on network-based transfer learning where part or all of the neural network trained on the source domain is reused.

2.4.1 Transfer Learning in Large Language Models

Deep transfer learning is an integral component in the training process of most LLMs. The process is usually split into a pre-training phase and a finetuning phase. In the pre-training phase, the LLM is trained on a huge corpus of text with the self-supervised task of predicting the next or masked tokens. This self-supervised learning process allows the LLM to learn the complex semantic meanings of tokens and their relationships with each other. After pretraining, however, the LLM outputs are often too generic to be useful for downstream tasks. This is why it is crucial to perform a finetuning phase where the LLM is further trained on a smaller corpus of selected examples from a downstream task. Finetuning is then a form of transfer learning where the LLM's learned language representations during pretraining are leveraged for a new target task.

Given an LLM $P_\Phi(y_t|x, y_{<t})$, the finetuned LLM can be represented as $P_{\Phi+\Delta\Phi}(y_t|x, y_{<t})$, where $\Delta\Phi$ is the difference in weights (weight deltas) due to finetuning. In this case, the number of parameters in $\Delta\Phi$ is equal to that in Φ. This constitutes an issue for LLMs where the number of parameters is usually in the scale of billions (e.g. the number of parameters of GPT-3 is 175 billion), and so the process of finetuning can be computationally prohibitive. Parameter-efficient finetuning techniques mitigate this issue by selecting and finetuning a subset of LLM parameters, achieving approximate results to full finetuning.

2.4.2 Low-Rank Adaptation (LoRA)

Low-Rank Adaptation (LoRA) is a parameter-efficient finetuning technique capable of reducing the number of finetuning parameters to as little as 0.01% of the total number of LLM parameters [6]. LoRA freezes the parameters of the LLM during training so that backprobagation does not affect them. Instead, it

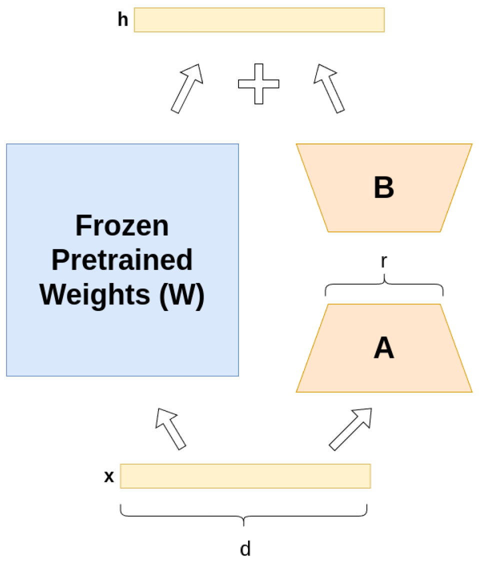


Figure 2.7: Low-Rank Adaptation.

adds adapters to certain layers of the LLM that project the gradients to a low-dimensional space. As shown in Figure 2.7, LoRA passes the d-dimensional input vector x simultaneously to the frozen pretrained layer with weights W and to the projection matrix A which projects x into an r-dimensional latent vector where $r << d$. The projection matrix B projects the r-dimensional change vector back into the d-dimensional space, which is summed with the output of the frozen layers to produce the vector h:

$$h = Wx + \Delta Wx = Wx + BAx \tag{2.17}$$

where the projection matrices A and B contain the only trainable parameters.

When dealing with transformer-based LLMs, there are multiple options for the subset of layers on which to apply LoRA. The original LoRA paper experimented with different combinations of the attention matrices W_q, W_k and W_v. The best performing combination was that of W_q and W_v which is the one we use when applying LoRA in this work. In our implementation, the weights of the LoRA adapters after finetuning act as the weight deltas which we extract style information from using the VAE.

Chapter 3

Literature Review

In this chapter, we overview previous literature related to our current work. We first differentiate between style-conditioned text generation and the adjacent task of style transfer. Then, we survey past approaches to the problem of few-shot stylized text generations and how our approach differs from them.

3.1 Style Transfer and Style-conditioned Text Generation

Text Style Transfer refers to the task of converting a piece of text from its given style to a target style while preserving its content [20], [21]. Two definitions of style have been put forward for this task. Attribute-based style transfer aims to transform text along one or more stylistic dimensions that are explicitly defined [22]. Attribute-based approaches are limited due to their reliance on labeled data and their inability to model complex styles [20], [21]. On the other hand, authorial style transfer aims to transform text to a style that is not easily explicitly defined, usually attributed to a unique author [23], [24].

Style-conditioned Text Generation is the task of generating more text in the style of a target author [25]. This task differs from style transfer since it does not aim to preserve provided content during inference and so it does not disentangle style from content. Lample et al. demonstrated that this disentanglement, besides being challenging, is unnecessary in order to model style [22]. The straight-forward approach to this task is to train a language model on a corpus of text in the target style [25]. However, as transformer-based LLMs have become the state-of-the-art for text generation, the data requirements for utilizing the text generation capabilities of LLMs with this approach have become increasingly prohibitive [2]. Our work addresses this shortcoming as we aim to model arbitrary text styles in a few-shot setting while simultaneously leveraging the generation capabilities of LLMs.

3.2 Few-shot Style-conditioned Text Generation

LLMs have been shown to be excellent zero-shot and few-shot learners in multiple NLP tasks [2]. STYLL [4] demonstrates that LLMs are capable of performing style transfer on arbitrary styles through prompting. However, STYLL prompts the LLM to classify the target style using specific attributes which are later used to perform style transfer, similar to the work of Reif et al. [26]. This approach performed satisfactorily for attribute-based style transfer, but it did not demonstrate the ability to improve its similarity to complex authorial styles. This is evidenced by STYLL's poor performance in approaching the target style even though it is capable of moving away from the source. Liu et al. [5] mitigate these shortcomings by proposing ASTRAPOP, a reinforcement learning actor-critic approach to perform style-transfer. However, while ASTRAPOP performs consistently well on medium-sized corpora, its performance is inconclusive on small-sized corpora belonging to a single author. Finally, Khan et al. [7] propose StyleMC, a unified approach to style transfer and stylized text generation using future discriminators [27]. StyleMC relies on pretrained style embeddings in its operation. Using model weights directly, we forgo the need for any predefined notions of style which allows for more powerful generalizations. We compare our approach to StyleMC later in this thesis.

Chapter 4

System Overview

In this section, we present our proposed system for few-shot style-conditioned text generation. We frame the problem of few-shot style-conditioned text generation as the task of obtaining a finetuned LLM, given a small corpus of text in a certain style, that is capable of generating text in that style. Given a pre-trained autoregressive LLM $P_{\Phi}(y_t|x, y_{<t})$, where Φ stands for the base model weights, we represent an instance of it finetuned on a corpus C as $P_{\Phi+\Delta\Phi_C}(y_t|x, y_{<t})$ where $\Delta\Phi_C$ stands for the difference in model weights (weight delta) due to finetuning. As a shorthand notation, moving forward, we represent the weight deltas as Δ_C and to the corresponding finetuned LLM as P_{Δ_C}. Thus, our task can be expressed as follows: given a small text corpus C^* from a certain author, we find Δ_{C^*} that, when applied to the base LLM weights, produces the finetuned LLM $P_{\Delta_{C^*}}$, which generates text that mimics the style of that author.

To perform this task, our approach constructs a latent space of LLM weight deltas and, during inference, approximates novel finetuned models. We discuss these two steps in details in the following sections.

4.1 Constructing the Latent Space

In this section, we go through the two steps required for constructing the latent space: extracting the weight deltas and training the VAE. These two steps are summarized in Figure 4.1. We start with a collection of text corpora belonging to distinct authors with various writing styles. We finetune a base LLM on

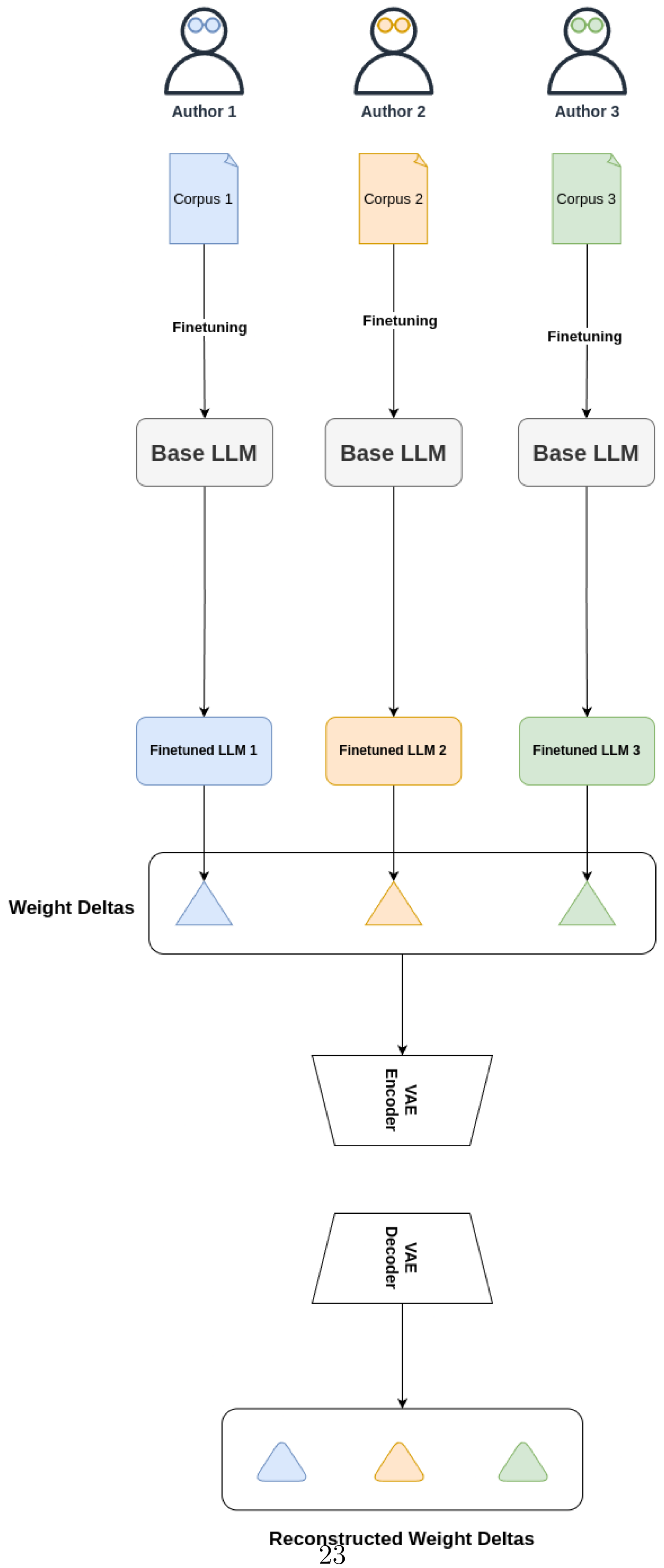


Figure 4.1: A schematic of how our approach constructs the latent space.

each of these corpora to get our finetuned models. The weight deltas are then extracted from these models and provided as the input to the VAE. The VAE learns to reconstruct the weight deltas, constructing the latent space in the process.

4.1.1 Extracting Weight Deltas

For the pre-trained LLM $P_\Phi(y_t|x, y_{<t})$, we extract a collection of LLM weight deltas $\Delta_1, \Delta_2, \Delta_3, \ldots, \Delta_n$ corresponding to LLM instances finetuned on text corpora $C_1, C_2, C_3, \ldots, C_n$, which belong to distinct authors. These corpora must be large enough for finetuning to capture the style of each text. When applied to the base LLM, these weight deltas produce the finetuned LLMs $P_{\Delta_1}, P_{\Delta_2}, P_{\Delta_3}, \ldots, P_{\Delta_n}$. Our approach does not make many assumptions about the finetuning process so any finetuning method can be used as long as the differences are captured in all or a subset of model weights, and the weight deltas indeed generate text in their respective author styles when applied to the base LLM. To make the behavior of our system more predictable, we also assume that finetuning is performed using the same hyperparameters for all authors so that the differences between weight deltas would be dependent on the training corpora only. In our implementation, to reduce the dimensionality of the weight deltas and the computational resources required for finetuning, we apply a parameter-efficient finetuning method (LoRA) which captures an approximation of the weight deltas.

4.1.2 Training the VAE

We employ a VAE to represent the weight deltas in a low-dimensional latent space that is salient and continuous enough for interpolation. In our context, saliency means that the VAE is capable of reconstructing the weight deltas with minimal effects on LLM generation, measured using metrics that we define in Section 5.1.1. Continuity refers to the ability of the VAE to encode the weight deltas of similar users closer together in the latent space. As discussed in Section 2.2, given the input Δ_C, the VAE encoder outputs two latent-space vectors corresponding to the mean μ and covariance σ matrices that define a Gaussian distribution in the latent space. The decoder samples from this distribution and outputs Δ'_C, a reconstructed version of Δ_C. Thus, the loss function of the VAE can be written as:

$$Loss = \|\Delta_C - \Delta'_C\|^2 + KL(N(\mu_{\Delta_C}, \sigma_{\Delta_C}), N(0,1)), \tag{4.1}$$

The first term represents the reconstruction loss which ensures that Δ'_C is as close as possible to Δ_C. The second term is the regularization loss which ensures that the VAE indeed represents each Δ_C as a Gaussian distribution in the latent space and not as point vectors like a regular autoencoder.

In our experiments, we found it useful to modify this loss function by multiplying the regularization term by a tunable parameter β so it becomes:

$$Loss = \|\Delta_C - \Delta'_C\|^2 + \beta \cdot KL(N(\mu_{\Delta_C}, \sigma_{\Delta_C}), N(0,1)). \tag{4.2}$$

We found that setting $\beta < 1$ helps ensure the VAE is capable of adequately reconstructing the weight differences due to their small scale and high variability.

4.2 Interpolation

The purpose for learning our VAE was to be able to generate new weight deltas and associated models from it, which we chose to do via interpolation on the latent space in order to leverage signals from the models it was trained on. There are multiple ways to perform interpolation from a latent space. We propose a simple linear interpolation method guided by a random sample of finetuned models $P_{\Delta_1}, P_{\Delta_2}, P_{\Delta_3}, \ldots, P_{\Delta_n}$. To simplify interpolation this method assumes that the topology of the latent space is smooth. We picked this method because it does not rely on any additional hyperparamters or domain knowledge for its operation, making it generalizable to other domains and datasets. Given a small corpus C^* containing a few text samples corresponding to an unseen author, we pick K models at random from the collection of finetuned models and perform one pass of finetuning on each using C^*. This step does not require much time or computational resources due to the small size of the corpus C^*. This is important for any future real-time applications of our system. By the end of this step, we get $\Delta_1^*, \Delta_2^*, \ldots, \Delta_K^*$ which vary slightly from the original $\Delta_1, \Delta_2, \ldots, \Delta_K$ and so are incapable of modelling the style of C^*. However, using the VAE, we interpolate the weight delta corresponding to C^* using these slight variations.

To interpolate using the VAE, we pass $\Delta_1, \Delta_2, ..., \Delta_K$ through the encoder to get $\mu_1, \mu_2, ..., \mu_K$. We do the same for $\Delta_1^*, \Delta_2^*, ..., \Delta_2^*$ and we get the corresponding $\mu_1^*, \mu_2^*, ..., \mu_K^*$. We treat each pair (μ_t, μ_t^*) as defining a vector in the latent space moving from the original model μ_t in the direction $\vec{r_t} = \mu_t^* - \mu_t$. This means that we now have K latent-space vectors, which, given the saliency and continuity of the latent space, should point towards a point that approximates the target model corresponding to C^* when passed through the decoder.

For each pair of N-dimensional lines $(\vec{\mu_1}, \vec{r_1})$ and $(\vec{\mu_2}, \vec{r_2})$, assuming they point towards our target model, we want to find their intersection point in the latent space. Expressed mathematically, we find t_1 and t_2 that satisfy the following equation:

$$\vec{\mu_1} + t_1\vec{r_1} = \vec{\mu_2} + t_2\vec{r_2} \tag{4.3}$$

which can be re-written as:

$$\vec{r_1}t_1 - \vec{r_2}t_2 = \vec{\mu_2} - \vec{\mu_1} \tag{4.4}$$

or in matrix form:

$$Ax = b, \tag{4.5}$$

where

$$A = \begin{bmatrix} \vec{r_1} & -\vec{r_2} \end{bmatrix}^T$$

$$b = \begin{bmatrix} \vec{\mu_2} - \vec{\mu_1} \end{bmatrix}$$

$$x = \begin{bmatrix} t_1 \\ t_2 \end{bmatrix}$$

Solving this equation returns the vector $\vec{\mu_{C^*}} = \vec{\mu_1} + t_1\vec{r_1} = \vec{\mu_2} + t_2\vec{r_2}$ which, when passed through the decoder, constructs our target Δ_{C^*}. However, since there is no guarantee that two N-dimensional lines will intersect, we modify this equation to instead find the least-square approximation as follows:

$$\underset{x \in \mathbb{R}^N}{\arg\min} \|Ax = b\|, \tag{4.6}$$

Solving this equation returns t_1 and t_2 that represent the closest point on each line to the other one. In this case, we return the midpoint of the line connecting the two points as our target $\vec{\mu_{C^*}}$.

4.3 Implementation

4.3.1 Datasets

We used three text datasets for our evaluation. First, the Twitter dataset is a subset of the Sentiment140 dataset which contains tweets tagged with the Twitter handles of their authors [28]. We filtered the dataset for authors with more than 200 tweets, to ensure there was enough text samples to finetune the LLM, and ended up with 17 authors in total. Second, we collected the Gutenberg dataset from the website of Project Gutenberg which provides free access to electronic books [29]. We retrieved the top 100 most popular books of all time. We assume that this collection of books approximate a dataset of distinct styles. We treated each book as a separate author with a distinct style since an author's style might change from one book to another, especially in fiction. Finally, we employed the Reddit dataset, which is a subset of the Reddit Million User Dataset (MUD) [30]. Similar to Khan et al. [7], we focused on four subreddits that have distinct styles: r/wallstreetbets, r/news, r/AskHistorians, and r/australia. We filtered the dataset for authors with more than 200 posts and picked 30 at random from each subreddit, ending up with 120 authors in total.

For each dataset, we separated 10% of the authors for test data and used the rest for training. We split the datasets by author to ensure that the styles of the authors in the test data is not seen by the VAE during training. For both the training and evaluation authors, we split each author's text corpus into train, validation and test sets which we used to finetune and evaluate the LLMs.

4.3.2 Finetuning Llama 2 with LoRA

We used Llama 2 as the base LLM in our implementation, which is an open-source autoregressive LLM [31]. Due to its large number of model weights (7 billion for the version we use), we apply Low-Rank Adaptation (LoRA) finetuning instead of full finetuning. As discussed in Section 2.4.2, instead of backpropagating losses to all model weights, LoRA freezes the base model weights and instead adds an adaptation layer to each Q and V attention layers of the LLM. LoRA provides two advantages for our system. First, we need not worry about the finetuning computational and time resources. Second, we can directly use the adapter weights as the weight deltas for our system. The 7-billion Llama 2 model contains 32 decoder units, each containing four 4096x4096 attention layers (corresponding to the Q,K,V,O matrices). This amounts to a total of more than 2 billion weights. Applying LoRA to the Q and V matrices, each unit now contains four *rank*x4096 vectors instead, where *rank* refers to the rank of the LoRA adapters. Setting the *rank* parameter to 2 decreases the number of weights to about 1 million weights only.

Thus, for each author corpus, we finetune an instance of Llama 2 using LoRA to obtain a 32x4x2x4096 weight delta. The extraction process of the LoRA weights is demonstrated in Figure 4.2. When the LoRA adaptation layers are populated by the weight delta, we get a finetuned model that generates text in the style of that author.

4.3.3 VAE Training

Since there is no inherent structure to the model weight deltas (as opposed to images for example), we found that the VAE encoder and decoder had to

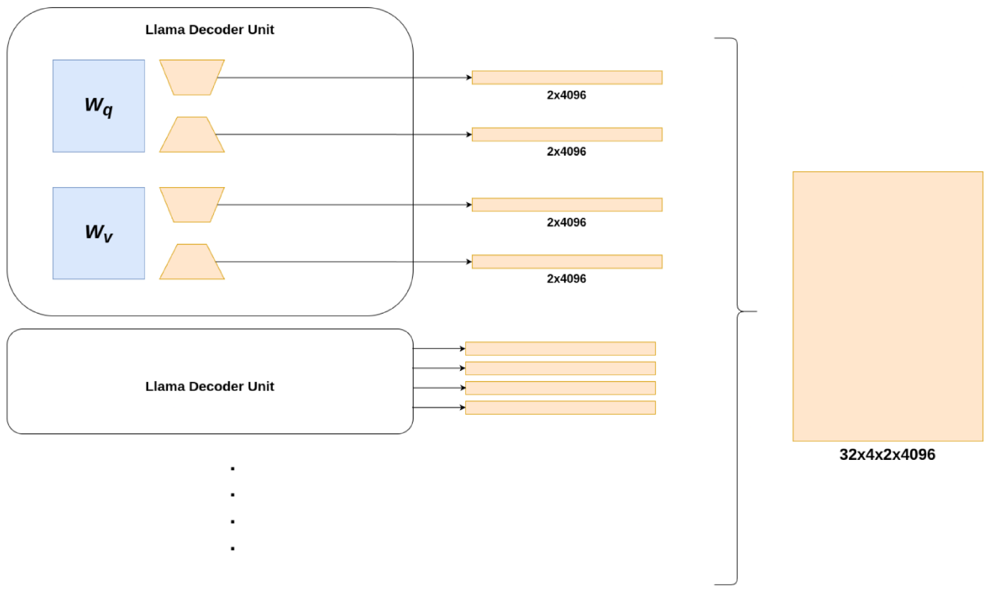


Figure 4.2: We extract the LoRA weights from the Llama LoRA adapters to create our weight delta whose dimensions are 32x4x2x4096.

be fully connected to achieve sufficient generalization. The VAE architecture we used in our system is shown in Figure 4.3. Through experimentation, we found the hyperparameters in Table 4.1 to be the most capable of allowing the VAE to fit the training data without overfitting.

Hyperparameter	**Value**
latent_dim	8
epochs	600
learning_rate	1e-4
β	0.03

Table 4.1: VAE training hyperparameters.

To reduce GPU memory and training time requirements of the VAE, we apply a compression step that aims to reduce the number of values in the weight deltas. Even though we set the LoRA rank to the smallest possible value for effective finetuning, we found that the dimensionality of the weight deltas still posed a bottleneck on the GPU memory available for training the VAE. According, we compress the weight deltas using Principal Component Analysis (PCA) due its simplicity and negligible GPU processing power requirements. We train 32 PCA models (one for each Llama 2 decoder unit) to reduce the

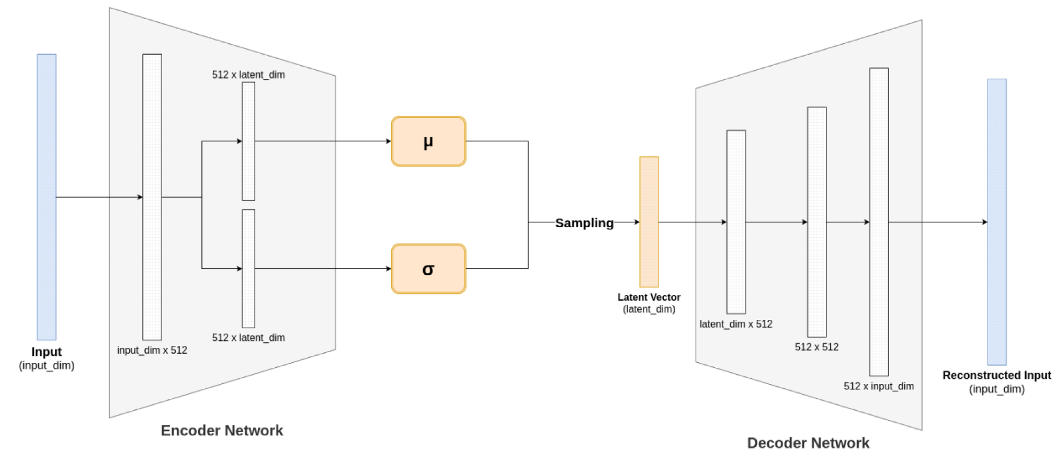
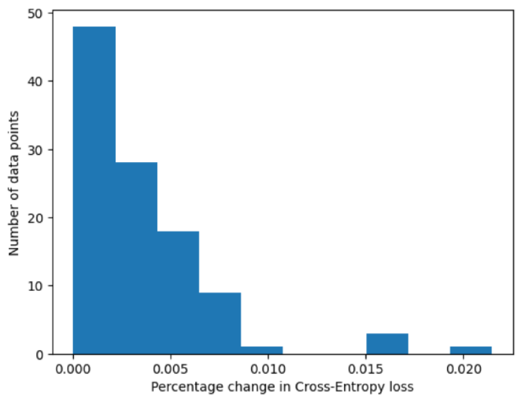


Figure 4.3: The Variational Autoencoder architecture that we used in our implementation.

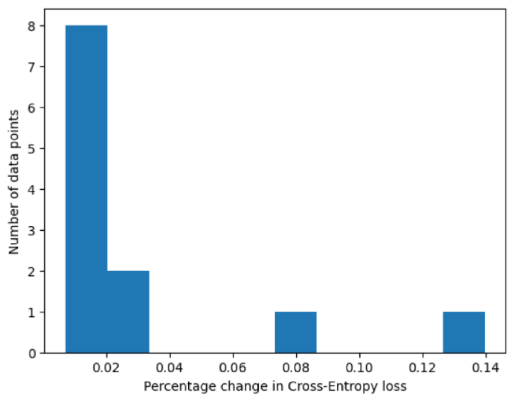
dimensionality of the weight deltas to 4x2xPCA_{dim} instead of 4x2x4096, where PCA_{dim} is the number of output components of the PCA model. We find that PCA is capable of reconstructing the model weight deltas, even unseen ones. For the Reddit corpus, with $PCA_{dim} = 400$, the average variance explained by the PCA models is 98%. This 2% loss corresponds to negligible effects on the performance of reconstructed models as verified by our assessment of their generated text and a quantitative assessment of their cross-entropy losses. We show the difference in the percentage change in cross-entropy loss for the original and the PCA reconstructed weight deltas in Figure 4.4.

After reducing the dimensionality of the weight difference tensors, we start training the VAE. Training the VAE with the default loss function (setting β to 1) resulted in poor reconstruction of the model weights. We hypothesize that this is due to the small scale of the model weights and so we set $\beta < 1$ to balance the loss calculation. We find that with $\beta < 0.1$, the model is capable of reconstruction while still preventing the latent representations from collapsing into point vectors.

However, we found that multiplying the KL loss by this factor led to the latent space being partially disconnected. We mitigated this issue by filtering disconnected data points to extract a continuous subset of the latent space. The filtering algorithm starts by sorting the data points based on the sum of



(a)



(b)

Figure 4.4: Histograms of the difference in percentage change in cross-entropy loss between the original and the PCA reconstructed weight deltas for (a) the train split and (b) the validation split of the Reddit dataset.

their reconstruction and KL loss ascendingly. This way, we can differentiate between the data points that the VAE fit well (low combined loss) and those that it did not (high combined loss). For each dataset or split of a dataset, we extract the five core data points with the least combined loss, which are then averaged to find a centroid in the latent space that represents them. After that, the maximum of their Euclidean distance from this centroid is calculated. For each data point in the dataset, we calculate the distance between itself and its respective centroid and compare that distance to its respective maximum distance multiplied by a factor. If its distance to the centroid is less than that threshold, it is considered a part of the continuous subset of the latent space. If not, then it is discarded. We found that a factor of three was capable of adequately discerning between the points that did and those that did not belong to the continuous subspace of the latent space for all datasets, assessed through inspecting the latent space graphically and using Euclidean distance. We experimented with different values for the hyperparameters of this algorithm and selected the ones that led to better results based on that criterion.

4.3.4 Interpolation

After the VAE is trained, we perform linear interpolation as described in Section 4.2. We use the *numpy* implementation of least-squares estimation to find the closest point on each line to the other line. Our approach returns the midpoint between these two points as the interpolated latent model, which is passed through the decoder to be constructed. For some lines, this point falls in the negative direction of the vector from the source to the target models, indicating that the two source models are diverging in one or more latent dimensions. In such cases, we do not consider the interpolated models as valid.

Interpolated models differ based on the choice of source models. For the results in Section 5.2, for each test data point, we choose base models belonging to a different dataset or a different dataset split than the one that the test data point belongs to. We believe this is more representative of the performance of our system on unseen data which might not belong to any of the datasets that the VAE was trained on. The Reddit dataset is already split according to subreddit so it naturally allows us to make this choice. As for the Gutenberg and Twitter datasets which are not split internally, we train the VAE on their combination in order to perform a similar analysis. In Appendix B, we show additional results where we ignore this consideration.

We experimented with two different methods of interpolation that differ in how they handle successive text samples. For simple interpolation, we perform linear interpolation directly using only the changes in latent space produced in the current timestep. For accumulative interpolation, we accumulate and average the changes in the latent space produced through all the previous timesteps until the current one. This allows our system to accumulate information and stabilizes the interpolated models. This stability problem is showcased in detail in Appendix A. The results shown in the next chapter are produced using accumulative interpolation. Results produced using simple interpolation can be found in Appendix B.

Chapter 5

Evaluation & Results

5.1 Evaluation

In this section, we discuss the details of the evaluation of our proposed system. First, we discuss the evaluation metrics and baselines we use for comparison. Then, we show the results of our experiments for all datasets.

5.1.1 Evaluation Metrics

We use two evaluation metrics in our experiments. First, we use the cross entropy loss on the test split of the corpus of the unseen author. Using this metric, we compare the performance of our interpolated models compared to the finetuned source models. Intuitively, this provides us with a quantitative estimate about the extent to which our models use the same words in the same order as the reference author corpus. This metric assumes access to the output probabilities of the LLM, thus we cannot use it to compare our system with closed-source models. Since LLMs have a different base cross-entropy loss for different writing styles, in order to aggregate cross-entropy losses among different authors, we normalize each author's cross entropy loss using its base LLM cross entropy loss.

Second, we use the Universal Authorship Representation (UAR) model, proposed by Rivera Soto et al. [32], to generate author style embeddings and measure the cosine similarity between the test split and text generated from the models under study.

5.1.2 Baselines

We compare the performance of our interpolated models to the finetuned source models and GPT 3.5 prompted for style-conditioned text generation. In addition, we compare the performance of our system on the Reddit dataset to the results reported in the literature. To the best of our knowledge, StyleMC is the only system in the literature that targets the problem of few-shot style-conditioned text generation, not only style transfer. Unfortunately, however, due to the recent publication of this system, the source code is not yet available for us to reproduce the results. To mitigate this limitation, we quote the results reported by the authors in their paper.

5.2 Results

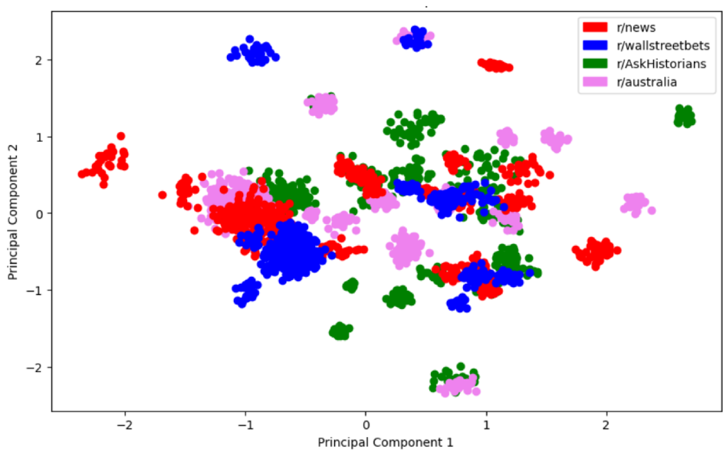
In this section, we present the evaluation results of our proposed system, organized by dataset. For each dataset, we show the latent space constructed by the VAE followed by the metric results.

5.2.1 Reddit Dataset

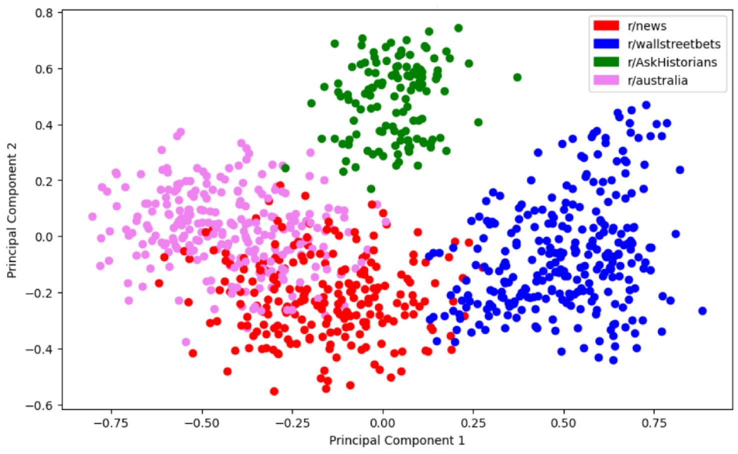
Latent Space

Figure 5.1a shows the latent space constructed by the VAE trained on the Reddit dataset. As is clear in the figure, the latent space is mostly fragmented and discontinuous. However, we were able to extract a continuous subset of the latent space through the filtering algorithm described in the Section 4.3.3. As shown in Figure 5.1b, in this subset of the latent space, the VAE was able to differentiate between the weight differences that belong to the different subreddits without any prior information.

We confirm this observation by calculating the Euclidean distance in the latent space between data points according to the subreddits they belong to. The heatmap in Figure 5.2b confirms that data points in this subset of the latent space are closer to each other than data points that belong to other subreddits. This is contrasted with the heatmap in Figure 5.2a, which shows that this property does not apply to the full latent space. For the following



(a)



(b)

Figure 5.1: Two-dimensional projection of the VAE latent space trained on the Reddit dataset for (a) the full latent space, and (b) the connected subset of the latent space.

results, we only operate within the continuous subset of the latent space, which encodes 40 data points (37% of the training dataset).

Cross-Entropy Results

Figure 5.3 shows the cross-entropy loss results for the Reddit dataset, grouped based on subreddit. We show the percentage change in cross-entropy loss, relative to the base cross-entropy loss, on the y-axis. All results are below the y-axis since finetuned models are better at modelling the text corpus compared to the base model due to the similarities among Reddit user styles even if they belong to different subreddits.

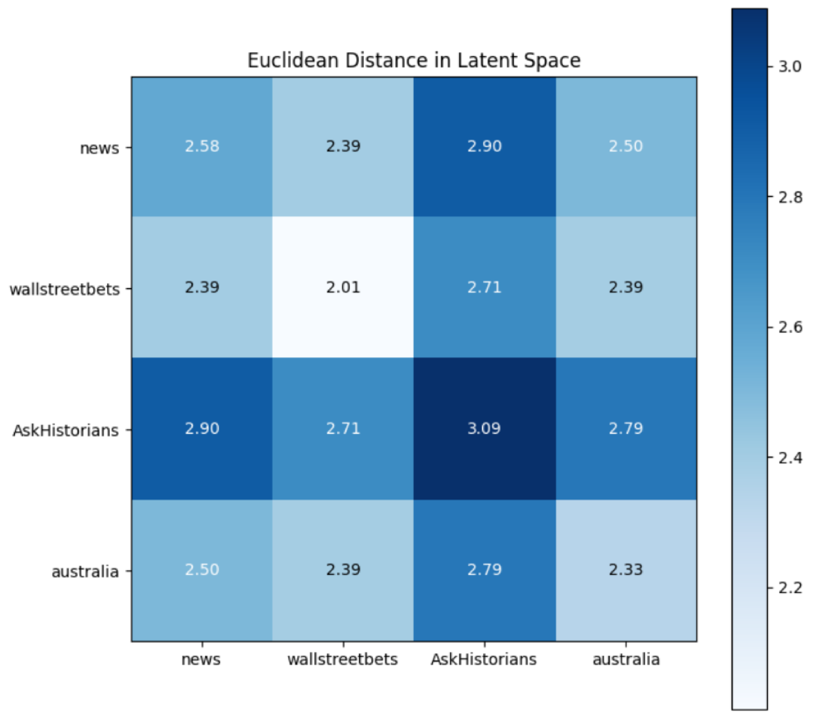
The results in Figure 5.3 show that our approach outperforms finetuning for all subreddits for low number of text samples (less than 10). When more text samples are available, the performance of our approach approximately converges to that of finetuning.

UAR Results

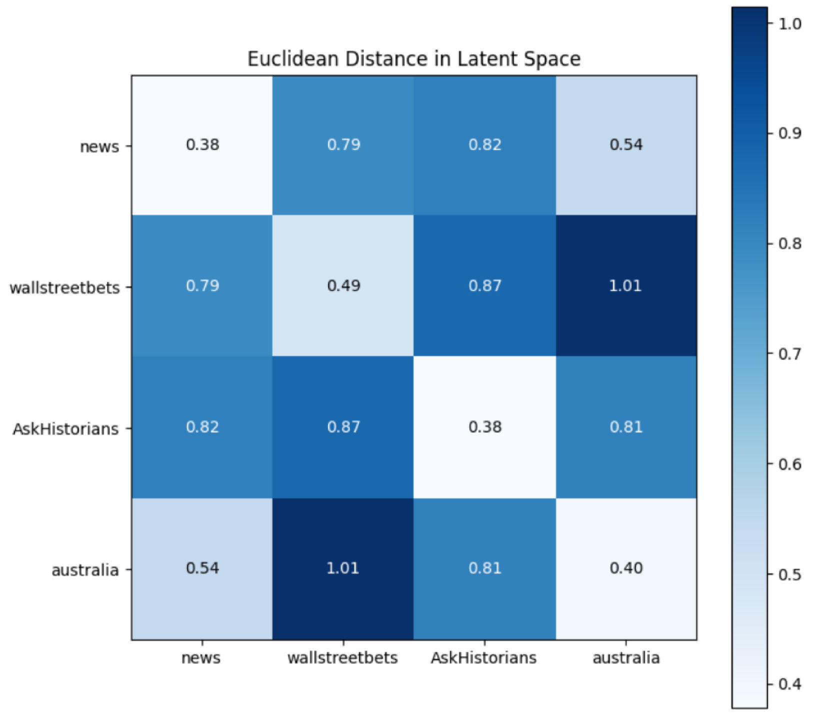
Figure 5.4 shows the cosine similarity scores for the UAR style embeddings. The only subreddit where there is a significant difference between the finetuned models and our approach is *r/wallstreetbets*. For the other subreddits, their performance is largely similar.

Finally, we show the performance of our system compared to GPT 3.5 prompted to perform style-conditioned text generation in Figure 5.5. We find that that while GPT 3.5 outperforms our system in some instances, the variance in its performance is significantly larger and, based on that, argue that it is less reliable for this task.

Table 5.1 shows the UAR similarity scores for our system and all baselines. The results shown in this table are aggregated across all subreddits. The table confirms our previous observations about the performance of our system compared to finetuning and prompting GPT 3.5. We include results generated by prompting an instruction-tuned version of Llama 2 for comparison. The table also shows that the reported performance of StyleMC with 16 text samples exceeds the performance of our system. The StyleMC paper however



(a)



(b)

Figure 5.2: Heatmap showing the Euclidean distance between subreddit weights in the VAE latent space for (a) all data points, and (b) data points that belong to the continuous subset of the latent space.

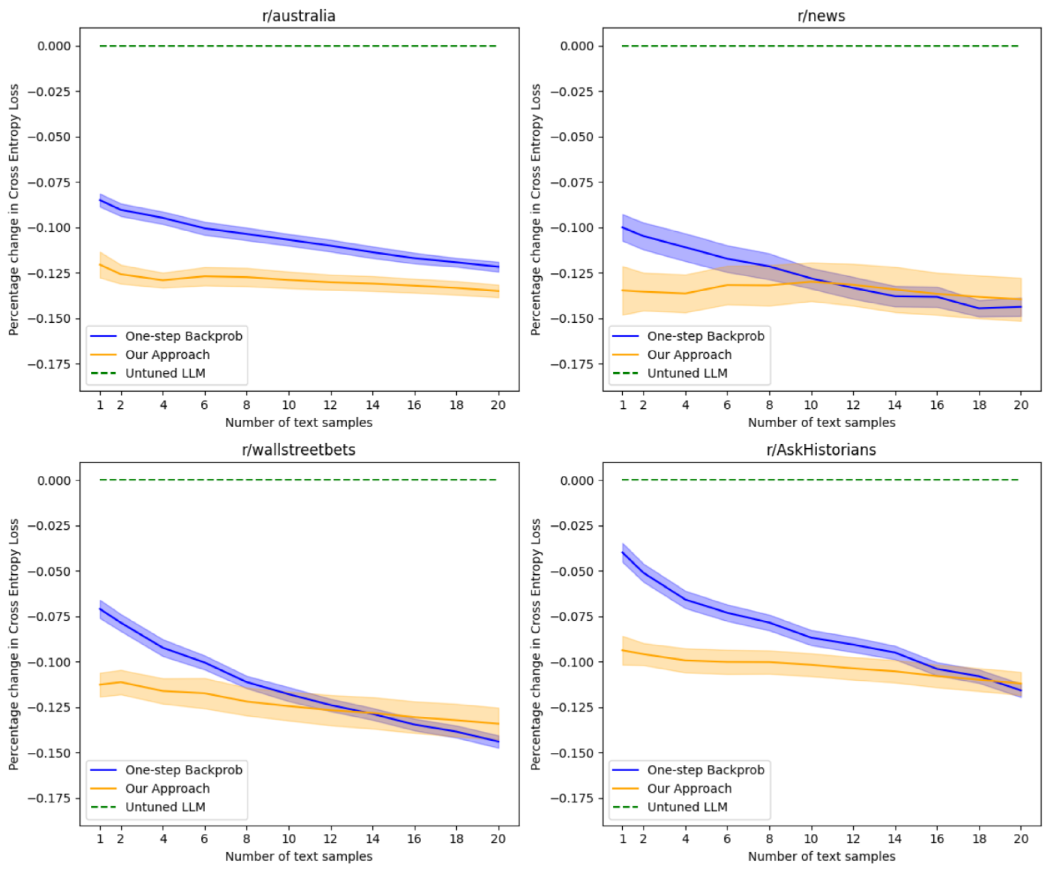


Figure 5.3: Cross-Entropy Loss Results for the Reddit dataset.

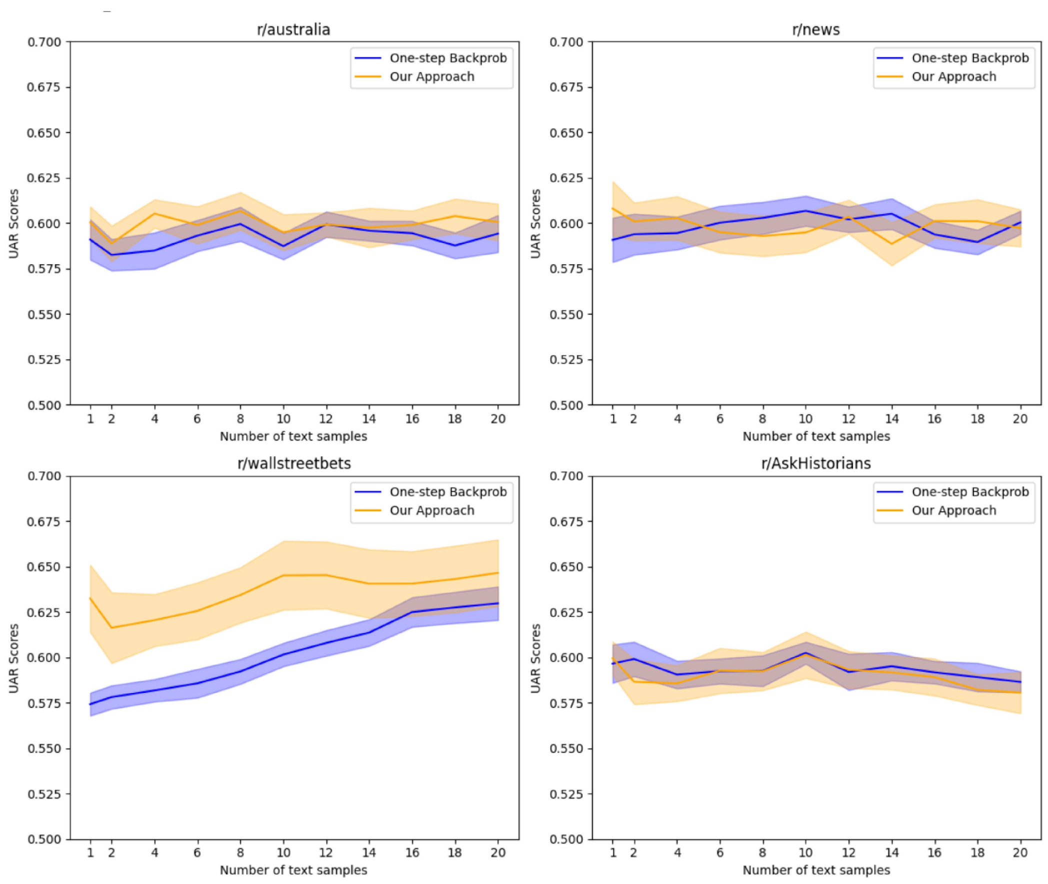


Figure 5.4: UAR Scores for the Reddit dataset.

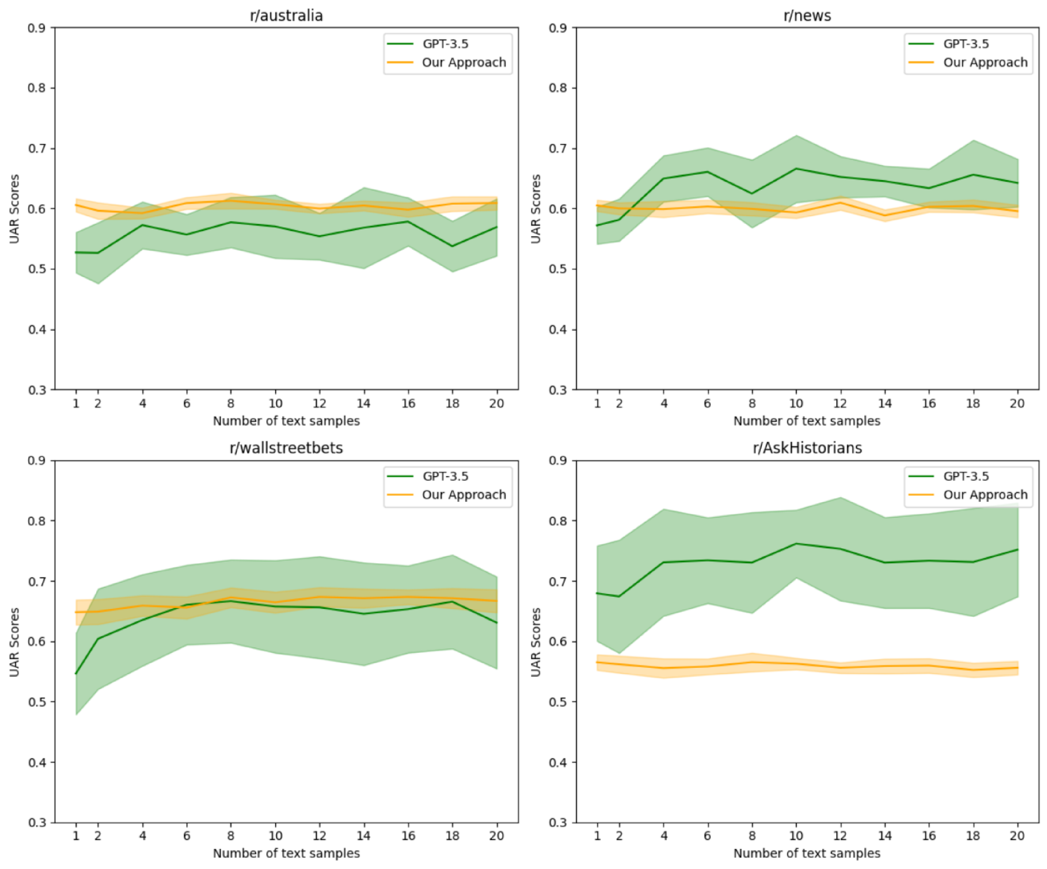
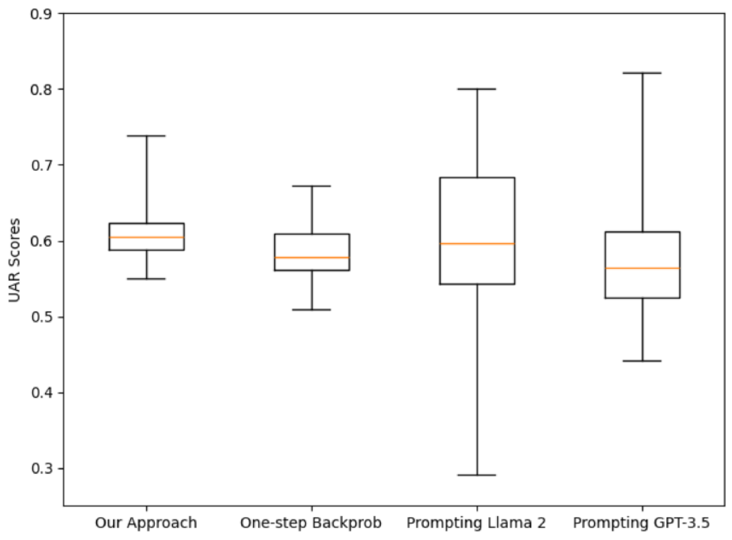
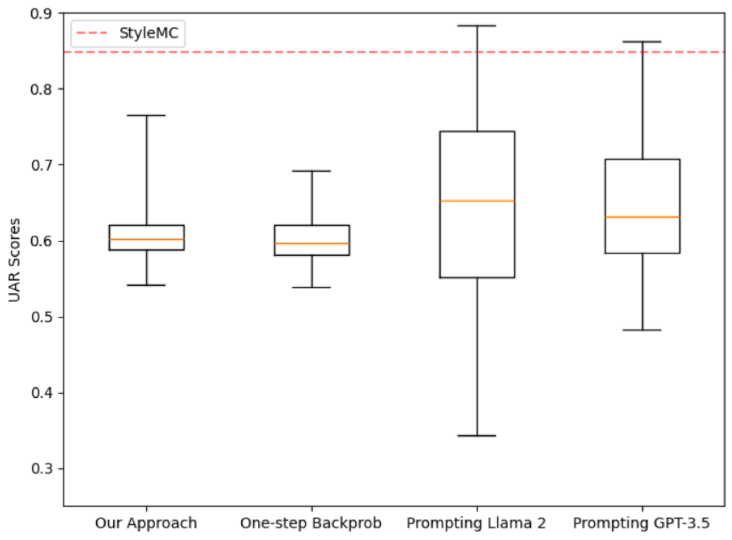


Figure 5.5: UAR Scores for the Reddit dataset comparing the performance of our approach with GPT-3.5.

does not provide any details about the distribution of its performance scores. It must be noted that the paper reports a higher mean score for GPT 3.5 (0.742) compared to ours (0.649). This might indicate that, even though we use Reddit data extracted from the same subreddits, our choice of users might differ from those used in evaluating StyleMC, leading to a negative skew in our results. It must also be noted that even though StyleMC outperforms our system using 16 text samples, our system maintains roughly the same performance using fewer text samples as shown in Table 5.1 and Figure 5.6. Figure 5.6 also confirms our previous observation that while prompting outperforms our system as the number of samples increases, the performance of our system has a smaller margin of error and so is more reliable for this task.



(a)



(b)

Figure 5.6: Comparison of different methods using the UAR similarty metric using (a) two text samples, and (b) 16 text samples.

Method	UAR (2 samples)	UAR (16 samples)
Our Apporach	0.609	0.607
One-step Backprob	0.588	0.601
Prompting GPT-3.5	0.581	0.649
Prompting Llama-2	0.592	0.633
StyleMC	-	0.849†

Table 5.1: Comparison of different methods using the UAR similarity metric. The dagger symbol indicates that the values are reported verbatim from their respective sources.

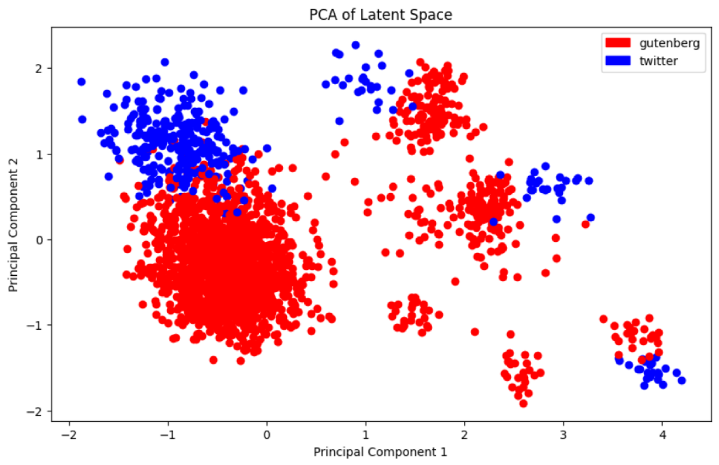
5.2.2 Twitter and Gutenberg Datasets

Latent Space

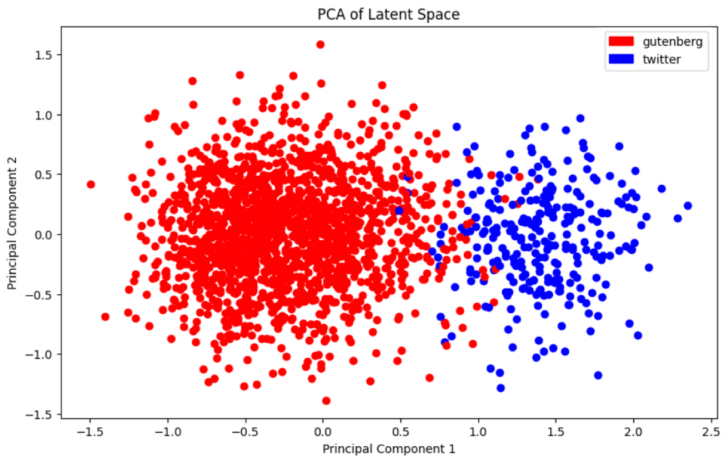
As with the Reddit dataset, we find that the full latent space of the VAE trained on the combined Twitter and Gutenberg datasets is disconnected. However, as shown in Figure 5.7, a larger portion of the data points fit within the continuous subset of the latent space. For the Twitter dataset, 11 data points were encoded in this continuous subset (78% of the training data), and for the Gutenberg dataset, 66 data points were encoded in it (81% of the training data). We hypothesize that these percentages are higher compared to that of the Reddit dataset due to the higher inner similarity among data points within the Twitter and Gutenberg datasets compared to the Reddit data. Again, the VAE is shown to be capable of discerning the difference between both datasets just from the weight deltas without any extra information.

Cross-Entropy Results

Figure 5.8 shows the cross-entropy loss results for the Gutenberg and Twitter datasets. Again, we show the percentage change in cross-entropy loss on the y-axis. The results for the Gutenberg dataset are similar to the results for the Reddit dataset with our approach outperforming finetuning for low number of text samples and converging to finetuning as the number of text samples increases. However, for the Twitter dataset, we find that finetuning slightly outperforms our approach especially as the number of text samples increases. We suspect that this is due to the fact that the Twitter dataset is quite small



(a)



(b)

Figure 5.7: Two-dimensional projection of the VAE latent space trained on the combined Twitter and Gutenberg datasets for (a) the full latent space, and (b) the connected subset of the latent space.

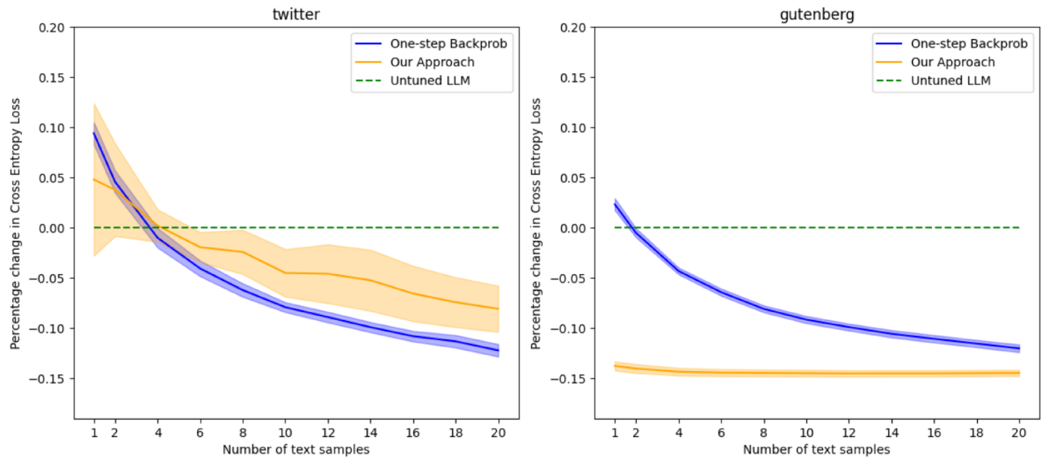


Figure 5.8: Cross-Entropy Loss Results for the combined Gutenberg and Twitter datasets.

compared to the Gutenberg dataset and so the VAE was unable to learn model representations that are salient enough for interpolation.

UAR Results

We attempted to generate text samples and calculate UAR scores for the resulting models from the Gutenberg and Twitter datasets. However, the results did not pass basic sanity checks and so we do not show them here. This is probably due to the fact that UAR representations were trained on Reddit data and so they are unable to adequately represent text data of out-of-distribution styles such as those of the Gutenberg dataset. We show these results and discuss this discrepancy in details in Appendix B.

Chapter 6

Conclusion & Future Work

In this chapter, we summarize our findings based on the results presented in the previous section. We also suggest future research directions that we believe have the potential to improve our proposed system and provide more insights into its functionalities.

6.1 Future Work

There are multiple directions that future research can take to build on our current work:

1. **Latent Space Connectivity:** One issue that we spent much time studying was how to balance latent space connectivity with the VAE's ability to reconstruct weights. Tuning the parameter β and selecting a subset of the latent space allowed us to circumvent this issue. However, it would be beneficial to find a training methodology that predictably produces a continuous latent space. This could also allow for an easier scaling of our system by including more data.

2. **Latent Space Salience:** While we showed that the latent space was salient enough to differentiate between weight deltas that belonged to different datasets or splits of the same dataset, it would be beneficial to further investigate the linguistic style features included in each latent space dimension, if any. This insight into the latent space could be used to construct more elaborate interpolation mechanisms.

3. **Interpolation Efficiency:** Our proposed interpolation mechanism relies on finetuning source models on a few text samples. While this process is efficient in our case due to our decision to work with LoRA, in order for this methodology to be more generalizable, it would be of interest to find other interpolation methods that can forgo this step and rather use other sources of information to guide the interpolation process.

4. **Interpolation Complexity:** Our proposed interpolation mechanisms make a simplifying assumption about the smoothness of the latent space. However, the latent space might be much more complex with multiple hills and valleys. In this case, it would be beneficial to employ an interpolation method that takes such topological information about the latent space into consideration.

5. **Generalized Style Evaluation Metrics:** Since the Reddit dataset is the standard dataset used to study style-conditioned text generation in the literature, UAR scores have also become the standard style-embedding metric to evaluate the output text. However, for methodologies such as ours that aim for generalizability of style extraction and interpolation, it is not enough to rely on them for evaluation. Future research might find it beneficial to try and construct more generic datasets and evaluation metrics for generalized style-conditioned text generation.

6.2 Conclusions

Through the work we presented in this thesis, we were able to provide the following answers our research questions.

R1 **Can we use a Variational Autoencoder to construct a latent space that is capable of encoding authorial styles using finetuned LLM weights?**

We found that a Variational Autoencoder was able to encode meaningful style information relying only on finetuned LLM weights, and able to reconstruct the weight deltas that produce these finetuned LLMs. While the VAE required multiple adjustments, such as discounting the KL loss values during training and subsequent filtering of the latent space, we believe that these adjustments are relatively minor.

R2 **Using this latent space, can we employ an interpolation strategy to find LLM weight deltas that produce LLMs that are more capable of generating text in a target authorial style compared to baselines using a small number of text samples from unseen authors?**

We found that it is possible to interpolate models that outperform finetuning from the VAE latent space. While UAR scores did not show significant improvement in style similarity, the cross-entropy scores for both datasets were significantly better than in the case of finetuning in low-data cases ($n < 10$). The performance of our system has also shown to be more reliable than prompting.

While our results were inconclusive in some areas, we believe this line of research can open the door for more powerful and generic low-resource LLM adaptation algorithms.

6.3 Ethical Considerations

This work proposes an approach to mimic the style of a certain author using a few text samples. In addition to useful application, we are aware that the system developed in this work can be used for malicious purposes such as scamming and cheating. We strongly oppose such use cases of our work. This work was developed to be used for educational and entertainment-related purposes only. Parts of this work however can be used for authorship verification and so can be repurposed to combat these malicious use cases.

References

[1] J. Li, T. Tang, W. X. Zhao, J.-Y. Nie, and J.-R. Wen, *Pretrained language models for text generation: A survey*, 2022. arXiv: 2201.05273 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2201.05273.

[2] H. Zhang, H. Song, S. Li, M. Zhou, and D. Song, "A survey of controllable text generation using transformer-based pre-trained language models," *ACM Comput. Surv.*, vol. 56, no. 3, Oct. 2023, ISSN: 0360-0300. DOI: 10.1145/3617680. [Online]. Available: https://doi.org/10.1145/3617680.

[3] L. Mou and O. Vechtomova, "Stylized text generation: Approaches and applications," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: Tutorial Abstracts*, A. Savary and Y. Zhang, Eds., Online: Association for Computational Linguistics, Jul. 2020, pp. 19–22. DOI: 10.18653/v1/2020.acl-tutorials.5. [Online]. Available: https://aclanthology.org/2020.acl-tutorials.5.

[4] A. Patel, N. Andrews, and C. Callison-Burch, "Low-resource authorship style transfer: Can non-famous authors be imitated?," 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:254853995.

[5] S. Liu, S. Agarwal, and J. May, "Authorship style transfer with policy optimization," *ArXiv*, vol. abs/2403.08043, 2024. [Online]. Available: https://api.semanticscholar.org/CorpusID:268379272.

[6] E. J. Hu, Y. Shen, P. Wallis, *et al.*, *Lora: Low-rank adaptation of large language models*, 2021. arXiv: 2106.09685 [cs.CL].

[7] A. Khan, A. Wang, S. Hager, and N. Andrews, "Learning to generate text in arbitrary writing styles," *ArXiv*, vol. abs/2312.17242, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:266573772.

[8] K. Lagutina, N. Lagutina, E. Boychuk, *et al.*, "A survey on stylometric text features," in *2019 25th Conference of Open Innovations Association (FRUCT)*, 2019, pp. 184–195. DOI: 10.23919/FRUCT48121.2019.8981504.

[9] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[10] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997, ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735.

[11] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, *Empirical evaluation of gated recurrent neural networks on sequence modeling*, 2014. arXiv: 1412.3555 [cs.NE]. [Online]. Available: https://arxiv.org/abs/1412.3555.

[12] D. P. Kingma and M. Welling, *Auto-encoding variational bayes*, 2022. arXiv: 1312.6114 [stat.ML]. [Online]. Available: https://arxiv.org/abs/1312.6114.

[13] T. Mikolov, K. Chen, G. Corrado, and J. Dean, *Efficient estimation of word representations in vector space*, 2013. arXiv: 1301.3781 [cs.CL]. [Online]. Available: https://arxiv.org/abs/1301.3781.

[14] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, *Attention is all you need*, 2023. arXiv: 1706.03762 [cs.CL].

[15] D. Bahdanau, K. Cho, and Y. Bengio, *Neural machine translation by jointly learning to align and translate*, 2016. arXiv: 1409.0473 [cs.CL]. [Online]. Available: https://arxiv.org/abs/1409.0473.

[16] T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, L. Màrquez, C. Callison-Burch, and J. Su, Eds., Lisbon, Portugal: Association for Computational Linguistics, Sep. 2015, pp. 1412–1421. DOI: 10.18653/v1/D15-1166. [Online]. Available: https://aclanthology.org/D15-1166.

[17] C. Raffel, N. Shazeer, A. Roberts, *et al.*, *Exploring the limits of transfer learning with a unified text-to-text transformer*, 2023. arXiv: 1910.10683 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1910.10683.

[18] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, *Bert: Pre-training of deep bidirectional transformers for language understanding*, 2019. arXiv: 1810.04805 [cs.CL]. [Online]. Available: https://arxiv.org/abs/1810.04805.

[19] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu, *A survey on deep transfer learning*, 2018. arXiv: 1808.01974 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1808.01974.

[20] Z. Fu, X. Tan, N. Peng, D. Zhao, and R. Yan, "Style transfer in text: Exploration and evaluation," *ArXiv*, vol. abs/1711.06861, 2017. [Online]. Available: https://api.semanticscholar.org/CorpusID:6484065.

[21] D. Jin, Z. Jin, Z. Hu, O. Vechtomova, and R. Mihalcea, “Deep learning for text style transfer: A survey,” *Computational Linguistics*, vol. 48, no. 1, pp. 155–205, Mar. 2022. DOI: `10.1162/coli_a_00426`. [Online]. Available: `https://aclanthology.org/2022.cl-1.6`.

[22] S. Subramanian, G. Lample, E. M. Smith, L. Denoyer, M. Ranzato, and Y.-L. Boureau, “Multiple-attribute text style transfer,” *ArXiv*, vol. abs/1811.00552, 2018. [Online]. Available: `https://api.semanticscholar.org/CorpusID:53295789`.

[23] H. Jhamtani, V. Gangal, E. H. Hovy, and E. Nyberg, “Shakespearizing modern language using copy-enriched sequence to sequence models,” *ArXiv*, vol. abs/1707.01161, 2017. [Online]. Available: `https://api.semanticscholar.org/CorpusID:9737200`.

[24] B. Syed, G. Verma, B. V. Srinivasan, A. Natarajan, and V. Varma, “Adapting language models for non-parallel author-stylized rewriting,” *ArXiv*, vol. abs/1909.09962, 2019. [Online]. Available: `https://api.semanticscholar.org/CorpusID:202719307`.

[25] A. Tikhonov and I. P. Yamshchikov, “Guess who? multilingual approach for the automated generation of author-stylized poetry,” *2018 IEEE Spoken Language Technology Workshop (SLT)*, pp. 787–794, 2018. [Online]. Available: `https://api.semanticscholar.org/CorpusID:49879813`.

[26] E. Reif, D. Ippolito, A. Yuan, A. Coenen, C. Callison-Burch, and J. Wei, “A recipe for arbitrary text style transfer with large language models,” in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, S. Muresan, P. Nakov, and A. Villavicencio, Eds., Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 837–848. DOI: `10.18653/v1/2022.acl-short.94`. [Online]. Available: `https://aclanthology.org/2022.acl-short.94`.

[27] K. Yang and D. Klein, “Fudge: Controlled text generation with future discriminators,” *ArXiv*, vol. abs/2104.05218, 2021. [Online]. Available: `https://api.semanticscholar.org/CorpusID:233210709`.

[28] A. Go, “Twitter sentiment classification using distant supervision,” 2009. [Online]. Available: `https://api.semanticscholar.org/CorpusID:18635269`.

[29] Project Gutenberg, *Project Gutenberg*, `http://www.gutenberg.org`, Retrieved July 20, 2024.

[30] J. Baumgartner, S. Zannettou, B. Keegan, M. Squire, and J. Blackburn, *The pushshift reddit dataset*, Jan. 2020. DOI: `10.5281/zenodo.3608135`. [Online]. Available: `https://doi.org/10.5281/zenodo.3608135`.

[31] H. Touvron, L. Martin, K. Stone, *et al.*, *Llama 2: Open foundation and fine-tuned chat models*, 2023. arXiv: `2307.09288 [cs.CL]`. [Online]. Available: `https://arxiv.org/abs/2307.09288`.

[32] R. A. Rivera-Soto, O. E. Miano, J. Ordonez, *et al.*, "Learning universal authorship representations," in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, M.-F. Moens, X. Huang, L. Specia, and S. W.-t. Yih, Eds., Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 913–919. DOI: `10.18653/v1/2021.emnlp-main.70`. [Online]. Available: `https://aclanthology.org/2021.emnlp-main.70`.

Appendix A

Case Study

In this section, we present a case study that showcases the functionality of our system's interpolation process. For the purposes of this case study, we pick a data point from the training data so that we would be able to compare its interpolated latent representation with its actual latent representation. First, we showcase a sample result where our system successfully interpolates latent models. Second, we study the effect of various system settings, specifically the choice of source models and the choice of data samples, on these results. Finally, we show how accumulative interpolation solves some of the issues that come up in different settings.

A.1 Success Sample

We picked a case study datapoint that belongs to the subreddit r/wallstreetbets. Similar to the results in Section 5.2, we randomly select source models that belong to other subreddits. In this case study, we pick 3 base models. Figure A.1 shows the cross entropy results produced by models interpolated using our approach compared to finetuning. As the figure shows, our system is capable of interpolating models that outperform naive finetuning on the given text samples.

Given the knowledge of the actual position in the latent space of the data point, we can inspect the operation of our system and check its validity. Figure A.2 shows this process. We also validate these results by plotting the Euclidean distance in the latent space between the interpolated models and the actual location of the target model, shown in Figure A.3. It is clear from these two figures that our system indeed captures the style of the target author as the number of text samples available increases.

A.2 Effect of Text Samples

While a text corpus belonging to a certain author generally represents their style, each text sample differs in how much it represents that style. For this reason, in few-shot settings, there is the risk that a few text samples that are not clearly in the style of the author might misguide the system. Figure A.4 shows the cross-entropy results for our system in the same setting but with a different subset of text samples. We can see that the text samples in the range from 6 to 10 push the system away from the target model. This is supported by

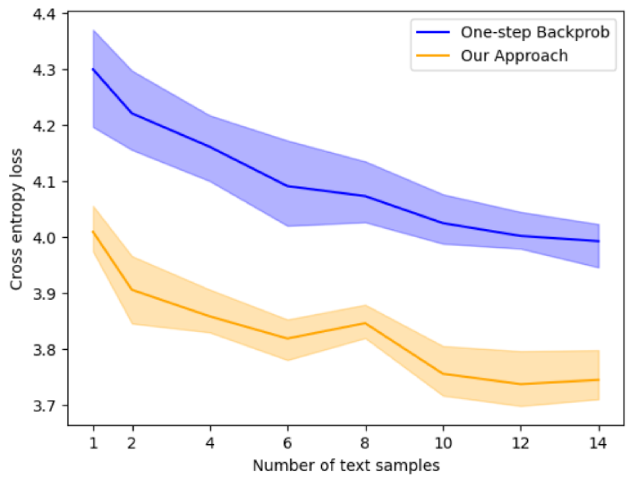


Figure A.1: Cross-Entropy Loss Results for the case study data point.

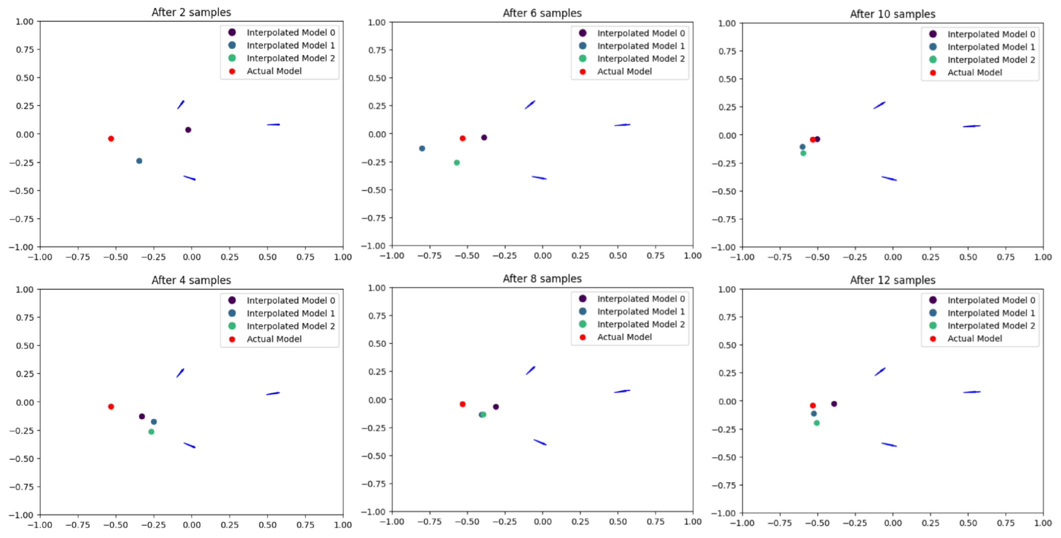


Figure A.2: A sequence of snapshots from the latent space during the operation of our system. The red dot represents the actual location of the target model in the latent space. The interpolated models successively approach the actual location as the number of samples increase.

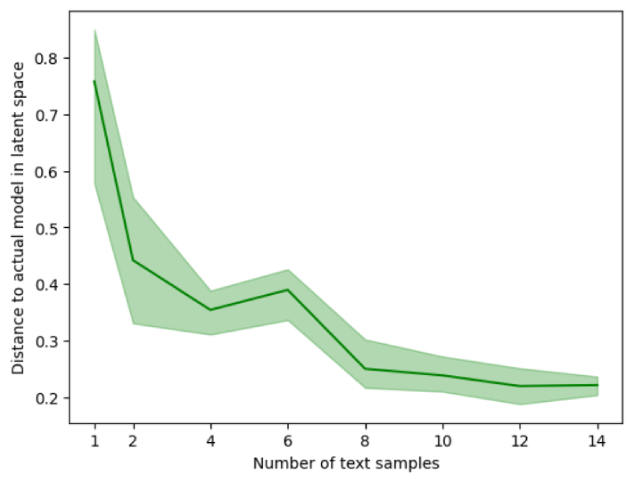


Figure A.3: The Euclidean distance between the interpolated models and the actual model in the latent space.

observing the latent space in Figure A.5. For this reason, it is crucial that text samples in few-shot settings are chosen to be clean and representative of the target style. We mitigate this issue by performing accumulative interpolation, showcased in Section A.4.

A.3 Effect of Source Models

Since our approach relies on changes in the source models during finetuning in order to interpolate finetuned models from the latent space, the choice of source models can largely affect the interpolation process. In Figure A.6, we show the cross-entropy results for source models that belong to the same subreddit as the target model (r/wallstreetbets). Two observations can be made here: (1) the source models perform much better than source models that did not belong to the same subreddit, and (2) the performance of the interpolated models is not as stable as before. Observing the latent space in Figure A.7, we find that due to the proximity of the base models to the target model in the latent space, the interpolation process becomes more difficult due to the instability of the direction of changes in the latent space. We discuss one possible solution for this issue in Section A.4.

A.4 Effect of Interpolation Method

In the previous sections, through observations of the latent space, we found that poor choice of text samples or source models can lead to unstable results for our system. As discussed in Section 4.3.4, we propose accumulative interpolation which averages the changes from the previous time steps until the current one and so leads to more stable interpolations. Figure A.8 showcases the effectiveness of this method in mitigating the stability issue for all the cases discussed in this section.

In Appendix B, we showcase the effect of this interpolation method on test samples.

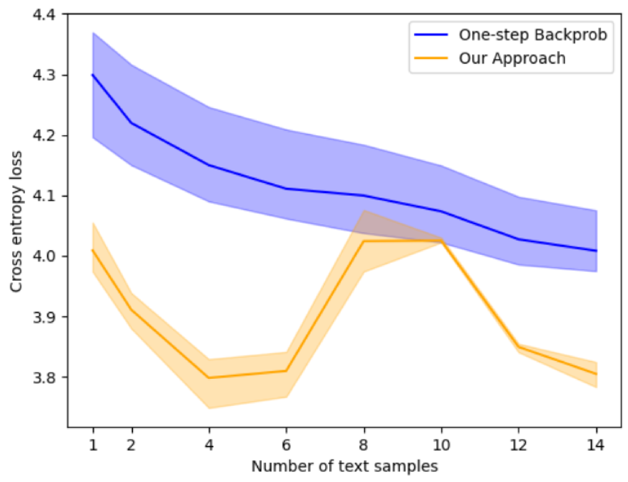


Figure A.4: Cross-Entropy Loss Results for different text samples of the case study data point.

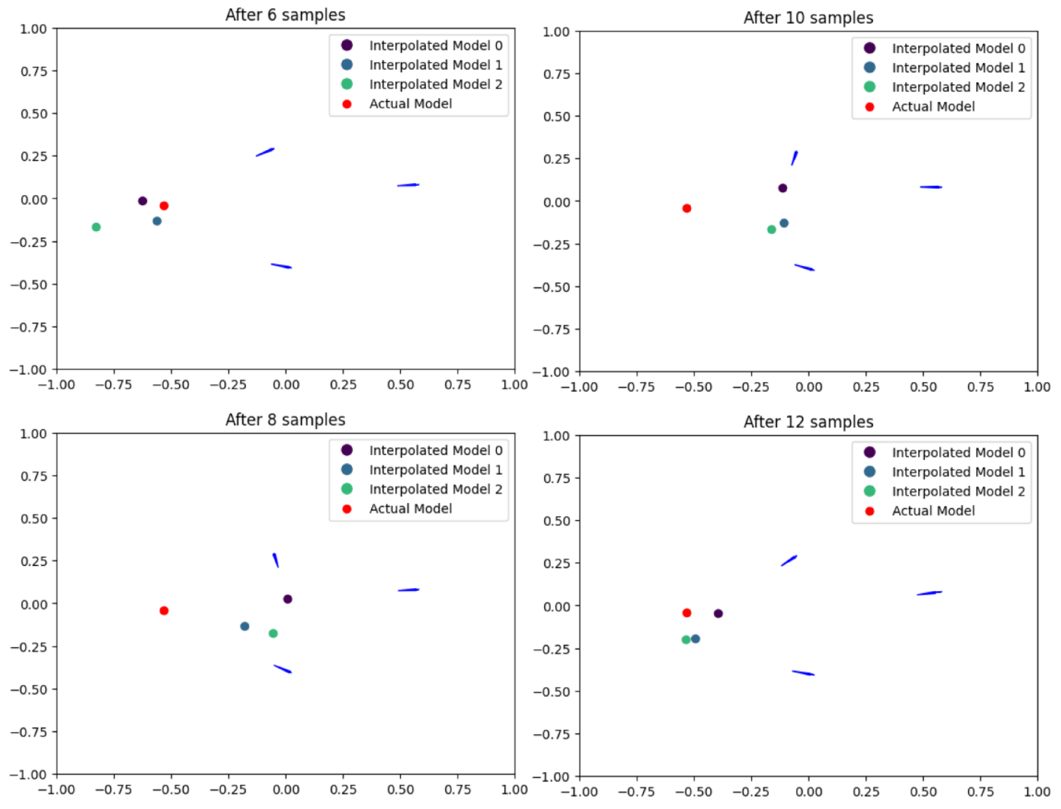


Figure A.5: A sequence of snapshots from the latent space during the operation of our system with a bad choice of text samples.

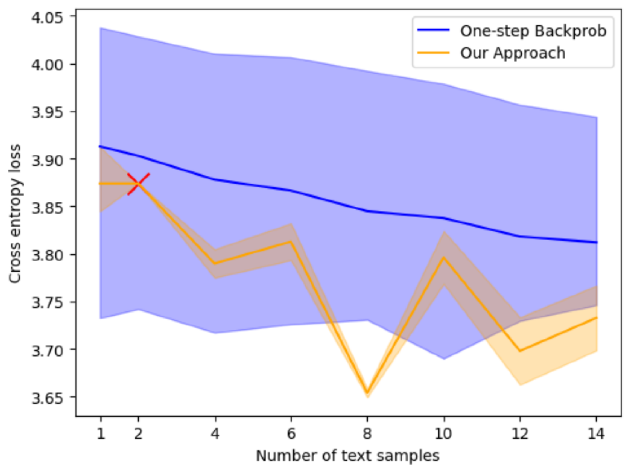


Figure A.6: Cross-Entropy Loss Results for the case study data point with source models that belong to the same subreddit. The red X represents failure in interpolation.

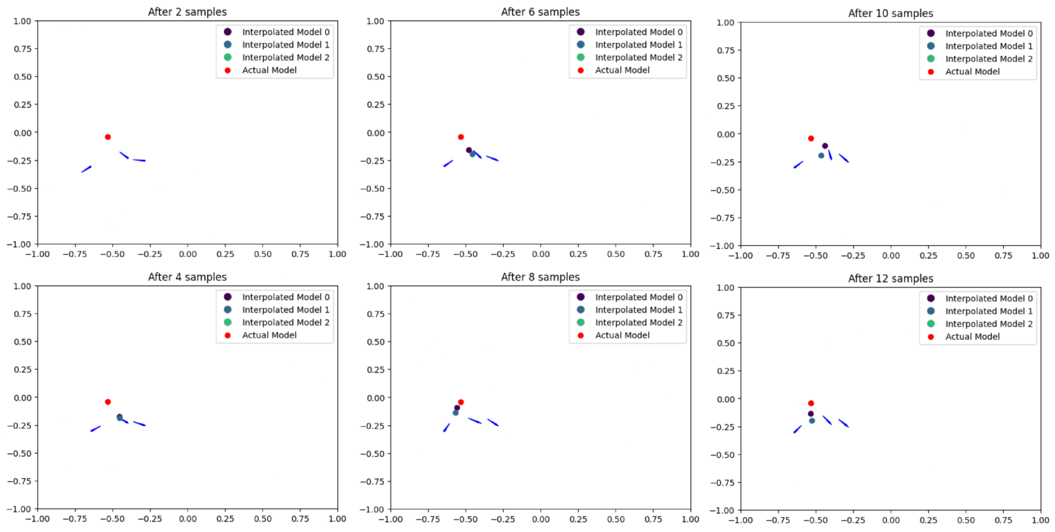
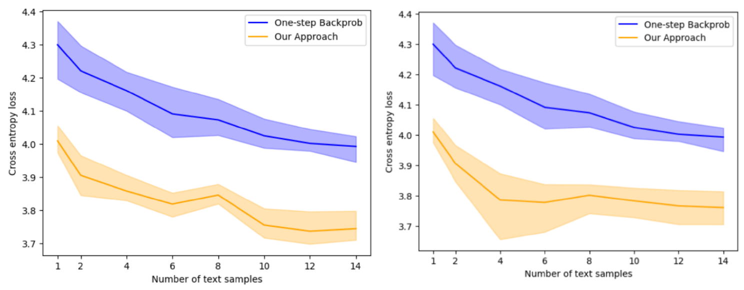
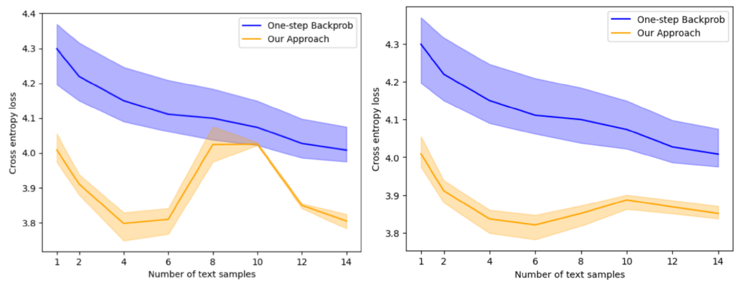


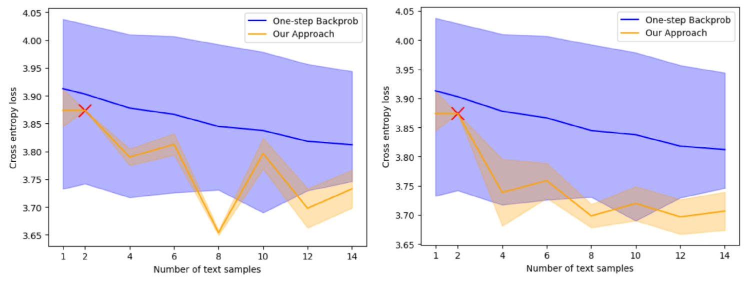
Figure A.7: A sequence of snapshots from the latent space during the operation of our system with source models that belong to the same subreddit as the target model.



(a)



(b)



(c)

Figure A.8: Cross-entropy results using simple interpolation (left) and accumulative interpolation (right) for (a) a good choice of source models and text samples, (b) the same source models but with different text samples, and (c) source models that belong to the same subreddit.

Appendix B

More Results

In this appendix, we present more results from the evaluations we performed on our system. In Section 5.2, we showed the results for the setting where the source models are selected to be belonging to a different dataset or a different split than the target model. We also only showed results for accumulative interpolation. In the next sections, we show cross-entropy results for the Reddit dataset in different settings. We also show the UAR results for the Twitter and Gutenberg datasets which we omitted for inconsistency.

Reddit Dataset Results with simple interpolation

Figure B.1 shows the cross-entropy results for the Reddit dataset with simple interpolation. Compared to Figure 5.3, we find that simple interpolation leads to more visible variations in the performance of the interpolated models. This observation is in tandem with our case study observations in Appendix A.

Reddit Dataset Results with different base models

Figure B.2 shows the cross-entropy results of the Reddit Dataset when base models are selected to be belonging to the same subreddit as the target model. To compare with Figure 5.3, we show the results for accumulative interpolation. We find that when the base models already belong to the same distribution as the target model, the performance of our system is almost the same as finetuning.

Twitter and Gutenberg Dataset UAR Results

Figure B.3 shows the UAR results for the combined Twitter and Gutenberg datasets. Comparing the case when the source models belong to the other dataset (Figure B.3a) to that when the source models belong to the same dataset (Figure B.3b), we find a discrepancy in the performance of the source models for the Gutenberg dataset where source models that do not belong to the dataset perform better on average than those that belong to it. This anomaly is probably caused by UAR scores being trained on Reddit data that is significantly out-of-distribution for the Gutenberg dataset. This hypothesis is supported by the relative consistency of the UAR scores for the Twitter dataset which is closer to the style of Reddit data.

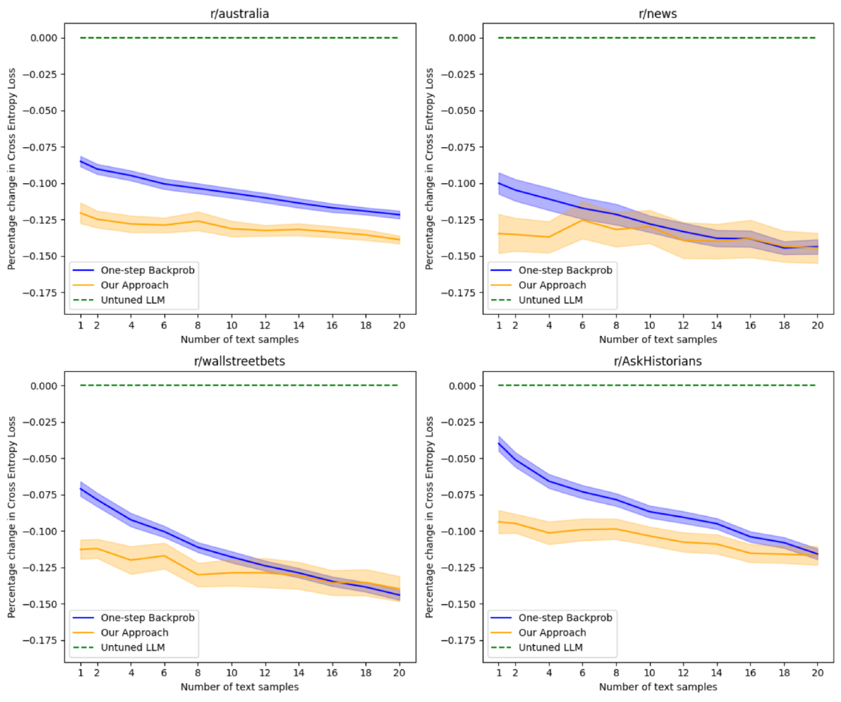


Figure B.1: Cross-Entropy Loss Results for Reddit Dataset with Simple Interpolation.

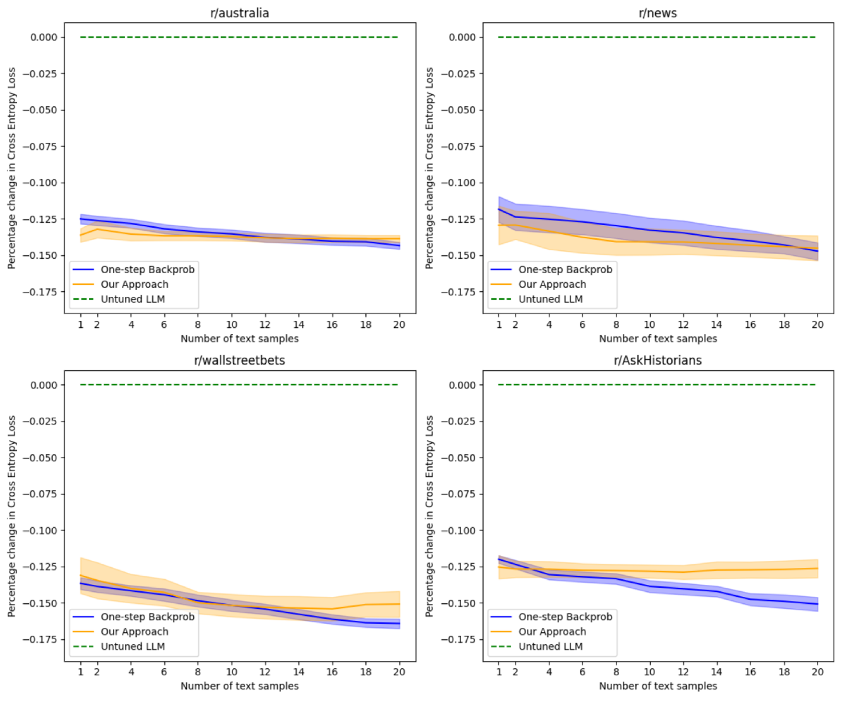
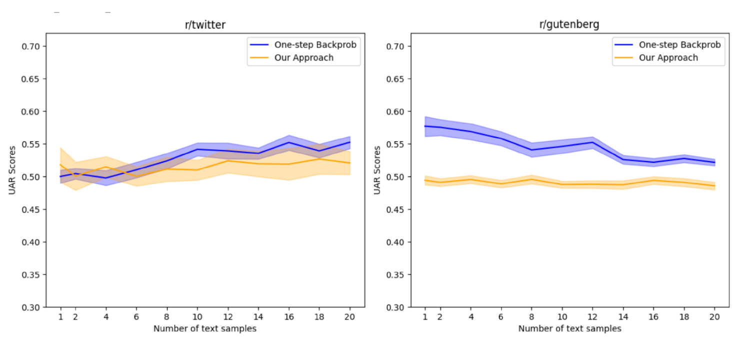
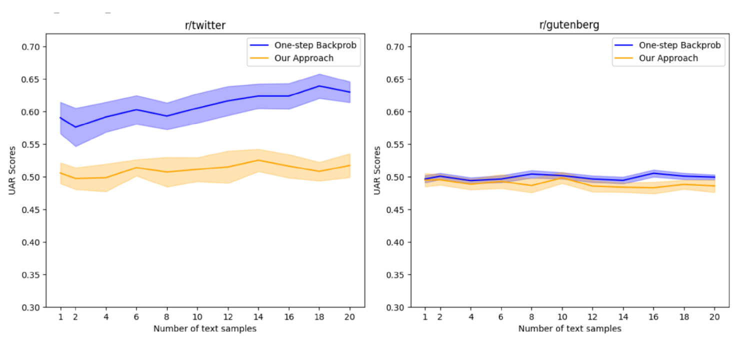


Figure B.2: Cross-Entropy Loss Results for Reddit Dataset with base models that belong to the same subreddit.



(a)



(b)

Figure B.3: UAR results for the combined Twitter and Gutenberg dataset with base models belonging to (a) the other dataset, and (b) the same dataset.