Reusing Transferable Weight Increments for Low-resource Style Generation

Anonymous EMNLP submission

Abstract

Text style transfer (TST) is crucial in natural language processing, aiming to endow text with a new style without altering its meaning. In real-world scenarios, not all styles have abundant resources. This work introduces TWIST (reusing Transferable Weight Increments for Style Text generation), a novel framework to mitigate data scarcity by utilizing style features in weight increments to transfer low-resource styles effectively. During target style learning, we derive knowledge via a specially designed weight pool and initialize the parameters for the unseen style. To enhance the effectiveness of merging, the target style weight increments are often merged from multiple source style weight increments through singular vectors. Considering the diversity of styles, we also designed a multi-key memory network that simultaneously focuses on task- and instance-level information to derive the most relevant weight increments. Results from multiple style transfer datasets show that TWIST demonstrates remarkable performance across different backbones, achieving particularly effective results in low-resource scenarios.

1 Introduction

Text style transfer (TST) is a significant area in natural language processing, aiming to endow text with a new style without altering its meaning. Numerous studies have been successfully applied to sentiment transfer (Luca, 2016; Lai et al., 2022; Li et al., 2018), text formalization (Rao and Tetreault, 2018; Jain et al., 2019), writing style imitation (Zhu et al., 2023; He et al., 2020; Riley et al., 2021), and role-specific dialogue scripts creation (Xu et al., 2023; Niu and Bansal, 2018).

As a sequence-to-sequence generation task, TST often faces the problem of parallel data scarcity. However, annotating style-specific data is often labor-intensive. Despite efforts to address this challenge, recent studies still face significant limitations. ***i)*** Self-supervised Pre-training (Riley et al., 2021; Xu et al., 2023), leverage large amounts of style corpus for self-supervised pre-training in the latent space, while performances with a lack of creativity or formulaic text easily fall short compared to supervised methods (Lai et al., 2022; Sudhakar et al., 2019). ***ii)*** In-context Learning (Shao et al., 2023; Wang et al., 2024), utilize the powerful capabilities of ChatGPT-4 (OpenAI, 2023) in few-shot learning, while the stability is affected by high accuracy in prompt design. Additionally, there is no guarantee that a single basis of the learned simplex will correspond to a target attribute such as dialect due to a lack of scalability. ***iii)*** Synthetic Data Generation (Suzgun et al., 2022; Chen and Huang, 2024), utilize closed-source models to generate large synthetic datasets, which is hard to guarantee the quality of synthetic data and may lead to bias. These issues undermine the efficacy and hinder the practical applications of TST.

This paper aims to utilize limited and authentic datasets with supervised signal guidance to achieve robust and scalable performance. To this end, we propose a supervised method **TWIST** (reusing Transferable Weight Increments for Style Text generation), which unleashes the potential to extract knowledge from known styles for unseen styles. TWIST is a dual-stage framework. In the **preparation** stage, we employ Low-Rank Adaptation (Hu et al., 2021) (LoRA) to train weight increments corresponding to source styles. These weight increments contain task-specific knowledge stored in a source weight pool. This storage structure is designed to simultaneously capture task- and instance-level information. After internal iterations, the weight pool can export the most relevant weight increments in a key-value format. We adaptively handle variable styles by focusing on dual-level authentic information. In the **optimization** stage, TWIST initializes partial parameters specific to target style by reusing weight increments, thereby reducing data dependency. Inspired by Ilharco

et al. (2023), we involve the weighted summation of parameter matrices to derive appropriate initial weights. In addition, we employ Singular Value Decomposition (SVD) to extract a small subset of parameters from source weight matrices, which are then injected into the initialization matrix. We retain the top-q singular values and their corresponding singular vectors, achieving an effect similar to sparse matrices. This reduces interference between weight matrices, making merging more effective.

Experiments demonstrate the remarkable efficacy and stability of TWIST on widely recognized benchmarks, achieving state-of-the-art (SOTA) performance across various backbone models, including T5 (Ruder et al., 2019) and LLaMA-2 (Touvron et al., 2023). Compared to other baselines relying on small-scale models, TWIST on T5-Large demonstrates superior performance with only 10% of the data. Notably, the performance based on LLaMA-2-7B surpasses other fine-tuning methods and achieves performance comparable to powerful ChatGPT4 (OpenAI, 2023).

Our contributions are summarized as follows:

I. We introduced TWIST, a model-agnostic method that extracts transferable knowledge from source styles. Experiments on different backbone networks show that TWIST improves parameter initialization, alleviating the impact of data scarcity.

II. The proposed weight pool is a reusable and scalable module focusing on the task- and instance-level information. Despite requiring additional computational resources, all the weight increments are shared, enabling flexible combination and reuse of information from different styles.

III. Our work demonstrates generalizability, achieving remarkable results in low-resource style scenarios and notable improvements in common high-resource styles.

2 Preliminary

We outline the common paradigms for learning a target task and describe our problem setup within the context of these paradigms.

2.1 LoRA Fine-tuned Model

Fine-tuning (Yosinski et al., 2014) is a machine learning technique where a pre-trained language model (PLM) is further trained on a task-specific dataset, adapting its general knowledge to the new task. The most common practice for learning a new task $t \in T$ involves fine-tuning all parameters (He et al., 2015) of a PLM on the target task training data $\{(\boldsymbol{x}, \boldsymbol{y})\}$. Given parameters, fine-tuning results in a specialized model $\boldsymbol{\theta}_\text{t}$ by optimizing:

$$\max_{\boldsymbol{\theta}_\text{t}} \Pr(\boldsymbol{y}|\boldsymbol{x}; \boldsymbol{\theta}_\text{t}) \tag{1}$$

To decrease training costs, parameter-efficient tuning (Houlsby et al., 2019) (PEFT) updates a small number of parameters for the target task. LoRA-based fine-tuning (Hu et al., 2021) is a parameter-efficient method. In this approach, the fine-tuned task model θ, fine-tuning results in a specialized model $\boldsymbol{\theta}_\text{t} \in \mathbb{R}^{d_\text{out} \times d_\text{in}}$ is decomposed as:

$$\boldsymbol{\theta}_\text{t} = \boldsymbol{\theta}_0 + \Delta\boldsymbol{\theta}_\text{t} = \boldsymbol{\theta}_0 + \mathbf{A}_\text{t}\mathbf{B}_\text{t}^\top \tag{2}$$

where $\boldsymbol{\theta}_0$ is frozen, $\mathbf{A}_\text{t} \in \mathbb{R}^{d_\text{out} \times r}$, $\mathbf{B}_\text{t} \in \mathbb{R}^{d_\text{in} \times r}$, and $r \ll \{d_\text{in}, d_\text{out}\}$. The number of parameters required in LoRA fine-tuning is $r \times (d_\text{out} + d_\text{in})$, which is much smaller than fully fine-tuning $(d_\text{out} \times d_\text{in})$. Thus each task only requires minimal trainable parameters and utilizes acceptable memory.

2.2 Transfer Source Weight for Target Style

In addition to the efficiency of LoRA parameters, we further explore the potential of LoRA as transferable parameters. A typical approach is interpolation-based methods (Finn et al., 2017), which aim to learn better parameter initialization to adapt to unseen tasks, essentially learning to initialize efficiently. They merge all model weights as follows: $\boldsymbol{\theta}_\text{t} = \boldsymbol{\theta}_0 + \Lambda \sum_{s=1}^{S} \Delta\boldsymbol{\theta}_\text{s}$, where Λ is a hyper-parameter, and S is the number of source tasks. $\Delta\boldsymbol{\theta}_\text{s}$ can be implemented in various ways, such as Adapter (Houlsby et al., 2019) or Prompt tuning (Li et al., 2022). We chose LoRA to reduce the network's depth. The performance of abovementioned methods is affected by parameter interference (Yadav et al., 2023). Asai et al. (2022); Peng et al. (2024) used weighting or attention operations to mitigate this impact, showing superior performance in comprehension tasks. However, for generation tasks requiring continuous word prediction, the direct addition of weights amplifies the negative impact of parameter interference.

2.3 Problem Setup

We are the first to introduce merging source LoRA to address TST tasks. Given a collection of source tasks $\{s_1, \ldots, s_S\}$, our primary goal is to obtain most relevant $\Delta\boldsymbol{\theta}_\text{s}$ for efficiently updating parameters $\boldsymbol{\theta}_\text{t}$ using the target task data $\{(\boldsymbol{x}, \boldsymbol{y})\}^t$. Additionally, we aim to find an appropriate method to improve the effectiveness of merging.

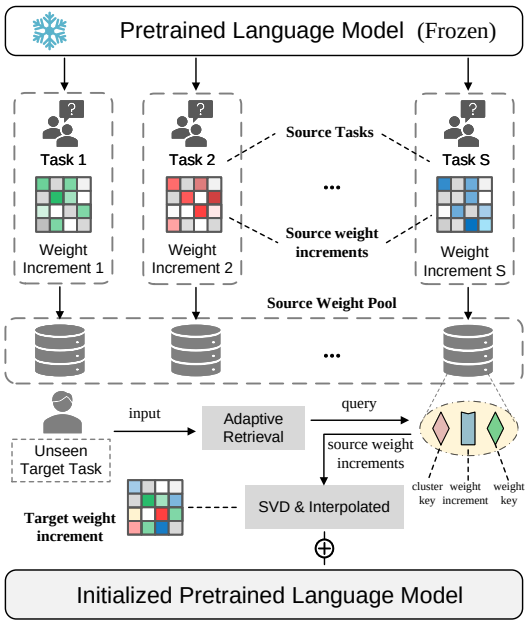


Figure 1: Framework of our method.

3 Method

Our proposed method is depicted in Figure 1 and is divided into two stages:

In the preparation stage, we construct a source weight pool (§ 3.1) to store reusable weight increments. We use a LoRA-based PEFT method to train weight increments independently for each source style in § 3.1.1. Subsequently, these weight increments are clustered in § 3.1.2 and stored in a specially designed multi-key memory network called weight pool in § 3.1.3. Finally, § 3.1.4 describes a retrieval scheme that considers task-level and instance-level information. In the optimization stage (§ 3.2), we discuss transferring the retrieved information to initialize the model more efficiently.

3.1 The Construction Of Weight Pool

To extract style-related knowledge from the source TST task, we learn a set of source weight increments and store them in a weight pool. These increments can be shared across all target tasks.

3.1.1 Source Weight Increments Pre-training

We first obtain source weight increments $\Delta\boldsymbol{\theta}_s$ for a collection of source tasks $\{s_1, \ldots, s_S\}$, where S is the number of source tasks. In practice, the datasets for style transfer tasks are typically high-resource to enable effective knowledge extraction.

Each source weight increment is trained only once for a source task and can be transferred to different target tasks. We perform the extraction using the method described in §2.1, aiming to obtain $\Delta\boldsymbol{\theta} = \{\Delta\boldsymbol{\theta}_1, \ldots, \Delta\boldsymbol{\theta}_S\}$. During training, only $\Delta\boldsymbol{\theta}_\text{s} = \mathbf{A}_\text{s}\mathbf{B}_\text{s}^\top$ for specific task s is updated by maximizing the likelihood as follows:

$$\max_{\Delta\boldsymbol{\theta}_\text{s}} \Pr(\boldsymbol{y}|\boldsymbol{x}; \boldsymbol{\theta}_0, \Delta\boldsymbol{\theta}_\text{s}) \tag{3}$$

3.1.2 Clustering Source Weight Increments

We construct the weight pool to identify similarities between source tasks for more effective style knowledge transfer. In particular, through the spectral clustering algorithm, we categorize the source weight increments into several clusters $\mathcal{C} \in \mathcal{G}$, where $\mathcal{G}$ is a weighted undirected graph. Specifically, each input is treated as a node p, and the weight between node p_i and p_j is $w_{\text{i,j}} = 1/(1+\|p_\text{i} - p_\text{j}\|)$. To ensure each cluster has sufficient nodes, we employ the min-max cut strategy to segment $\mathcal{G}$, resulting in $\mathcal{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_C\}$, where C is the number of clusters. When retrieving weight increments, it is better to identify the suitable weight cluster and select the most relevant source knowledge.

3.1.3 Structured Storage Space

The key-value memory network is a common storage structure (Miller et al., 2016). We have optimized this approach by proposing a **Multi-Key Memory Network**, which simultaneously considers task-level and instance-level information. Its storage structure is as follows:

$$\mathbf{P} = \{\boldsymbol{k}_c^\mathcal{C}; \boldsymbol{k}_s^\Theta; \Delta\boldsymbol{\theta}_s\}_{c=1}^{C} \tag{4}$$

where cluster key $\boldsymbol{k}_c^\mathcal{C} \in \mathbb{R}^d$, weight key $\boldsymbol{k}_s^\Theta \in \mathbb{R}^d$, and d is the embedding size. Each $\Delta\boldsymbol{\theta}_s \in \mathbf{P}$ is correlated with a $\boldsymbol{k}_c^\mathcal{C}$ and $\boldsymbol{k}_s^\Theta$. These learned source weight increments serve as value vectors in our memory network. The generation and retrieval of keys will be discussed in the next section.

3.1.4 Adaptive Knowledge Retrieval

Firstly, we initialize a set of learnable parameters with a semi-orthogonal matrix (Saxe et al., 2014), following Wang et al. (2022). Next, given a tokenized input $\boldsymbol{x}$, we use BERT, denoted as $\boldsymbol{f}_\text{BERT}$ (Devlin et al., 2019), to extract its semantic features. This maps the original text $\boldsymbol{x}$ to a hidden feature space, generating the query vector $\boldsymbol{q}$. Mathematically, $\boldsymbol{q} = \boldsymbol{f}_\text{BERT}(\boldsymbol{x})$ ($\boldsymbol{x} \in \mathbb{R}^{l\times c}$, $\boldsymbol{q} \in \mathbb{R}^d$), where l represents the sequence length. To maintain consistency, $\boldsymbol{f}_\text{BERT}$ remains frozen at all stages.

Multi-Keys Optimization. We prioritize optimizing weight key $\boldsymbol{k}^{\Theta}$. To reduce the cost of computation, we use the K-nearest neighbors algorithm to calculate the cosine similarity between the query and keys, retrieving the top-W most similar keys $\boldsymbol{K}^{\Theta} = \{\boldsymbol{k}_1^{\Theta}, \ldots, \boldsymbol{k}_W^{\Theta}\}$, where $W \leq S$. The weight keys $\boldsymbol{k}^{\Theta} \in \boldsymbol{K}^{\Theta}$ are optimized to align with the input instance distribution as follows: $\boldsymbol{k}^{\Theta} = \boldsymbol{k}^{\Theta} + \gamma \nabla_{\boldsymbol{k}^{\Theta}} \cos(\boldsymbol{q}, \boldsymbol{k}^{\Theta})$, where γ is the learning rate and $\cos(\cdot)$ denotes cosine similarity. To avoid local optima, we follow the method proposed by Peng et al. (2024), introducing random key masking. This approach reduces the retriever's tendency to over-prioritize specific keys, encouraging the allocation of attention to other elements.

To compute the cluster key $\boldsymbol{k}^{\mathcal{C}}$, we take the geometric mean of the weight keys $\boldsymbol{k}^{\Theta}$ within the cluster. The formula is as follows: $\boldsymbol{k}^{\mathcal{C}} = \left(\prod_{i=1}^{n} \boldsymbol{k}_i^{\Theta}\right)^{\frac{1}{n}}$, where n is the number of weight keys in the cluster and $\boldsymbol{k}_i^{\Theta}$ represents the individual weight keys within the cluster.

Adaptive Retrieval. For an unseen target task t and its instance data $\{(\boldsymbol{x}_i, \boldsymbol{y}_i)_{i=1}^{N}\}^t \in (\mathcal{X}, \mathcal{Y})^t$, we use a task-level query $\boldsymbol{q}^{\text{task}}$ and an instance-level query $\boldsymbol{q}^{\text{ins}}$ to adaptively retrieve the weight increments. The task-level query aims to capture overall information relevant to the specific target task and is designed as a vector $\boldsymbol{q}^{\text{task}} \in \mathbb{R}^c$ (Vu et al., 2022). However, due to the diversity and limitations of the resources in the pool, to enhance the robustness of retrieval, we design an instance-level query calculated as follows: $\boldsymbol{q}^{\text{ins}} = \frac{1}{N} \sum_{\boldsymbol{x} \in \mathcal{X}} \boldsymbol{f}_{\text{BERT}}(\boldsymbol{x})$.

For each source weight increment $\Delta\boldsymbol{\theta}_s \in \mathbf{P}$, we use $\boldsymbol{q}^{\text{task}}$ and $\boldsymbol{q}^{\text{ins}}$ to lookup its corresponding cluster key and source key respectively. The Retrieval Score $\mathcal{R}_s$ between $\Delta\boldsymbol{\theta}_s$ and instance $\boldsymbol{x}$ is calculated as follows:

$$\mathcal{R}_s = \text{softmax}(\lambda \cdot \boldsymbol{q}^{\text{task}} \cdot \boldsymbol{k}_s^{\mathcal{C}} + (1-\lambda) \cdot \boldsymbol{q}^{\text{ins}} \cdot \boldsymbol{k}_s^{\Theta}) \quad (5)$$

where λ is a hyper-parameter.

3.2 Reusing LoRA-based Weight Increments

Previous methods for merging weight often face parameter interference, particularly in generation tasks, due to the denser parameters for complex tasks (Yadav et al., 2023). Using interpolation methods directly can lead to instability in initialization. One feasible approach is to prune $\mathbf{A}_{\text{s}}\mathbf{B}_{\text{s}}^{\top}$ (Zhang et al., 2023); however, initial experiments in Table 9 indicate that pruning results in significant performance degradation on interpolation methods. Therefore, we used singular value decomposition (SVD) to extract a small subset of parameters from the task-specific LoRA matrix. These parameters are combined with $\mathcal{R}_s$ and injected into the model requiring initialization.

Specifically, we first perform SVD on $\mathbf{A}_{\text{s}}\mathbf{B}_{\text{s}}^{\top}$, resulting in $\mathbf{A}_{\text{s}}\mathbf{B}_{\text{s}}^{\top} = \mathbf{U}_{\text{s}}\mathbf{\Sigma}_{\text{s}}\mathbf{V}_{\text{s}}^{\top}$, where $\mathbf{U}_{\text{s}} \in \mathbb{R}^{d_{\text{out}} \times r}$ and $\mathbf{V}_{\text{s}} \in \mathbb{R}^{d_{\text{in}} \times r}$ are orthogonal matrices, and $\mathbf{\Sigma}_{\text{s}} \in \mathbb{R}^{r \times r}$ is a diagonal matrix with its diagonal entries sorted from highest to lowest. We select the top-q singular values and their corresponding singular vectors to reduce the number of parameters. This involves constructing submatrices: $\mathbf{U}_{\text{s}}^{(q)} \in \mathbb{R}^{d_{\text{out}} \times q}$, $\mathbf{V}_{\text{s}}^{(q)} \in \mathbb{R}^{d_{\text{in}} \times q}$, and $\mathbf{\Sigma}_{\text{s}}^{(q)} \in \mathbb{R}^{q \times q}$ contain the first q columns of $\mathbf{U}_{\text{s}}$, $\mathbf{V}_{\text{s}}$, and the top q singular values along its diagonal, respectively. We then approximate the original product $\mathbf{A}_{\text{s}}\mathbf{B}_{\text{s}}^{\top}$ as $\mathbf{U}_{\text{s}}^{(q)}\mathbf{\Sigma}_{\text{s}}^{(q)}\mathbf{V}_{\text{s}}^{(q)\top}$. This reduces the number of parameters from $r \times (d_{\text{out}} + d_{\text{in}})$ to $q \times (d_{\text{out}} + d_{\text{in}} + 1)$, because now we only need to store the top-q singular values and their corresponding vectors. Finally, according to the Retrieval Score $\mathcal{R}_s$, the injected target weight incement is calucated as:

$$\Delta\boldsymbol{\theta}_t = \sum_{s=1}^{S} \mathcal{R}_s \cdot \mathbf{U}_{\text{s}}^{(q)}\mathbf{\Sigma}_{\text{s}}^{(q)}\mathbf{V}_{\text{s}}^{(q)\top} \quad (6)$$

Discussion. Selecting q much smaller than r can greatly improve computational efficiency. However, the choice of q is crucial. If q is too close to r, the approximation will retain more information from the original LoRA matrix, but it will lose the sparsity benefit, and parameter interference might rise. On the other hand, if q is too small, important information may be lost, negatively impacting the initialization and overall model performance.

3.2.1 Training for Target Style Generation

The model is expected to learn unseen target tasks, and for each target task t_k, we use the learned parameters $\boldsymbol{\theta}_0 + \Delta\boldsymbol{\theta}_t^k$ to initialize the weight matrix for the task. Given the dataset $\mathcal{D}_k$ of t_k, the learning objective is defined as:

$$\mathcal{L}_{\Delta\boldsymbol{\theta}_t^k}(\mathcal{D}_k) = -\sum_{i=1}^{N} \log \Pr(\boldsymbol{y}_i \mid \boldsymbol{x}_i; \boldsymbol{\theta}_0, \Delta\boldsymbol{\theta}_t^k) \quad (7)$$

where $\boldsymbol{\theta}_0$ is the frozen, and $\{(\boldsymbol{x}_i, \boldsymbol{y}_i)_{i=1}^{N}\}_k^t \sim \mathcal{D}_k$ is a training sample. During testing, the source weight increments $\Delta\boldsymbol{\theta}_s$ is unchanged to save original information, and $\Delta\boldsymbol{\theta}_t^k$ will further optimization

through gradient iteration. The overall training pipeline is illustrated in **Algorithm 1**.

4 Experiments

4.1 Experiment Setup

Datasets. We use the following four datasets from nine different styles: **YELP** (Luca, 2016) includes parallel sentences of positive and negative reviews. **GYAFC** (Rao and Tetreault, 2018) provides sentences of formal and informal expressions. **Shakespeare** (Zhu et al., 2023) contains the works of Shakespeare. **Genshin** (Xu et al., 2023) is based on game roles and contains six sub-datasets: Xiangling, Hutao, Mona, Diluc, Venti, and Noelle. The data usage is shown in Table 12.

Source and Target Tasks. In all experiments, the target task remains unseen for any source task. Practically, we rotate through different target styles. If **Shakespeare** is selected as the target style, we remove its corresponding weight increment from the weight pool until training is complete. Then, we restore the **Shakespeare** weight increment and remove the next target style.

Backbones. Our method is a model-agnostic approach to TST that can be applied to various backbones. We selected two models for our study: the encoder-decoder **T5**-Large model (Ruder et al., 2019) and the decoder-only **LLaMA2**-7B model (Touvron et al., 2023). Different backbones are compared against their respective baselines to ensure a fair comparison, maintaining equivalent parameter counts.

4.2 Implementation Details

Baselines. For **T5**-Large, the selected baselines include: Fine-tune (FT) T5-Large (Ruder et al., 2019), CrossAligned (Lai et al., 2022), Delete&Retrieve (Li et al., 2018), B-GST (Sudhakar et al., 2019) and TextSETTR (Riley et al., 2021). For **LLaMA2**-7B, the selected baselines include: QLoRA-based Fine-Tuning (QLFT) LLaMA2-7B (Dettmers et al., 2023), Few-shot (FS) Alpaca-7B (Taori et al., 2023) and Few-shot ChatGPT-4 [1] (OpenAI, 2023). The prompt used by Few-shot Tuning is shown in Appendix B.

Evaluation Metrics. We use automatic metrics to assess attribute control, such as style accuracy (**ACC**) and content preservation (**CP**) after transfer. To estimate the output style, we follow the method proposed by Riley et al. (2021) and train a classifier for the specific style on the training set. More details can be found in Appendix A. For content preservation, we calculate Self-BLEU using SacreBLEU (Post, 2018), following Sudhakar et al. (2019) and Xu et al. (2020a). Additionally, we report the "**G-score**" (the geometric mean of **ACC** and **CP**), following (Xu et al., 2018).

Parameter Settings. For experiments based on **T5**-Large, we utilized 2 NVIDIA RTX4090 GPUs. The learning rate was set to 2×10^{-2}, with a weight decay of 0.01 and a batch size of 8. We ran the model for 50 epochs with 100 warmup steps. For experiments involving **LLaMA2**-7B, we employed 2 NVIDIA A100 GPUs. The learning rate was adjusted to 2×10^{-4}, with a weight decay of 0.01 and a batch size of 2. These experiments were conducted over 3 epochs with 0.03 warmup steps. The key optimization learning rate was 1×10^{-3}. All experiments used AdamW (Loshchilov and Hutter, 2019) as the optimizer.

5 Results

5.1 Main Results

Results on T5. We use T5-Large as the backbone model for comparison with other small-scale PLM methods. The training data consists of the full dataset, and the results are presented in Table 1, where $^{\heartsuit}$ indicates that we folded the results for this task. Detailed results are provided in Table 15. TWIST demonstrates stronger generalization capabilities than directly fine-tuning T5-Large (Ruder et al., 2019). Benefiting from better initialization, it shows significant improvements in low-resource writing styles and role dialogue styles, with increases of 20.6% and 6.1%, respectively. On average, compared to the previous best method Delete&Retrieve (Li et al., 2018), TWIST shows an improvement of 12.5%. On formality transfer, B-GST (Sudhakar et al., 2019) outperformed it by a margin of 4.8%.

Results on LLaMA2. The comparison results based on LLaMA2-7B are shown in Table 2, with unfolded results provided in Table 16. Compared to the direct fine-tuning method (Dettmers et al., 2023), TWIST achieves an overall improvement of 8.3%. We achieve a noticeable content preservation (**CP**) lead compared to some few-shot methods

[1] The specific version is gpt-4-1106.

Dataset	YELP			GYAFC			Shakespeare			Genshin♡			avg		
Methods	CP	ACC	G	CP	ACC	G	CP	ACC	G	CP	ACC	G	**CP**	**ACC**	**G**
FT T5-Large	56.7	52.0	54.3	53.2	71.5	61.7	15.9	62.1	31.4	53.1	55.6	54.3	44.7	60.3	50.4
CrossAligned	2.9	68.2	14.1	21.5	68.2	38.3	4.8	53.7	16.1	14.7	57.5	28.9	11.0	61.9	24.3
Delete&Retrieve	**56.9**	49.4	53.0	56.2	73.4	64.2	14.0	56.8	28.2	54.1	52.7	53.3	45.3	58.1	49.7
B-GST	54.2	60.2	57.1	63.5	**80.0**	**71.3**	17.1	69.5	34.5	30.7	60.1	42.9	41.4	67.5	51.5
TextSETTR	54.4	44.9	49.4	42.7	75.6	56.8	14.2	78.6	33.4	32.5	52.6	41.3	36.0	62.9	45.2
Ours T5-Large	51.8	**70.3**	**60.3**	**65.2**	70.7	67.9	**17.8**	**80.7**	**37.9**	**55.3**	**60.2**	**57.6**	**47.5**	**70.5**	**55.9**

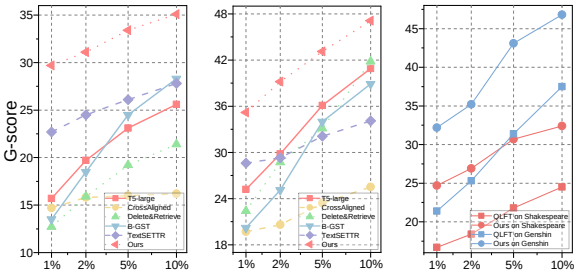
Table 1: Comparison of full datasets in TST between small-scale PLM methods.

Dataset	Shakespeare			Genshin♡			
Methods	CP	ACC	G	CP	ACC	G	**avg.G**
QLFT LLaMA2	20.3	86.8	42.0	54.9	64.0	59.2	50.6
FS Alpaca	14.9	83.1	35.2	41.5	69.3	53.6	44.4
FS ChatGPT-4	15.6	**90.2**	37.5	43.0	**76.3**	57.2	47.4
Ours LLaMA2	**22.1**	88.5	**44.2**	**57.9**	74.0	**65.4**	**54.8**

Table 2: Comparison of full datasets in TST between large-scale PLM methods.

using carefully designed prompts. Regarding style accuracy, we achieve comparable results to the powerful closed-source model ChatGPT-4 (OpenAI, 2023), trailing by 1.9% and 3.0%.

5.2 Results on Low-resource Setting



(a) Shakespeare (b) Genshin (c) LLaMA2-7B

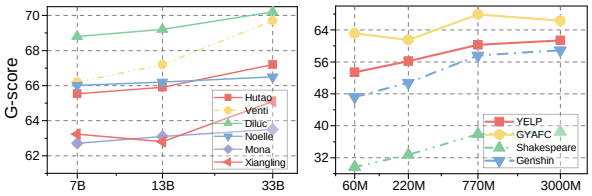
Figure 2: Comparison under various shots of instances.

In the low-resource setting, we use only a small number of training instances from the target task. We perform a secondary sampling of the target task dataset to obtain a subset of {1%, 2%, 5%, 10%} of the full training instances. We conducted three random samplings in total to reduce the randomness of the experiment.

Figure 2a and Figure 2b show the performance of small-scale model-based methods on the Shakespeare writing style and dialogue styles of specific roles in Genshin. Due to the better initialization capability of our method, it adapts more effectively to low-resource scenarios. However, as the amount of data increases, the advantage gradually diminishes. Additionally, the BLUE line and RED line in Figure 2c represent two different tasks: writing style and role dialogue style transfer. The performance is similar to that of another backbone network.

5.3 Analysis

We compare the performances of TWIST across different scales to explore upper limits. Subsequently, we analyze the effect of the proposed module on enhancing style transfer ability.



(a) LLaMA2 series (b) T5 series

Figure 3: Comparison under various parameters.

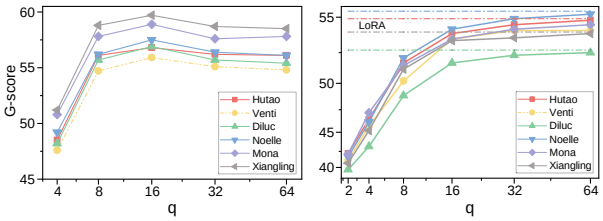
Power of Scale. As shown in Figure 3, our method benefits from the size of the parameter scale, but the improvement brought by increasing the parameters is not absolute. Test style transfer task requires a comprehensive consideration of content preservation and style accuracy. One of the challenges is balancing content and style. From our experience, increasing the parameters can lead to more diverse expressions and redundant representations. This redundancy can affect the calculation of ScareBLEU, impacting content preservation at the data scale level.

Dataset	Shakespeare			Genshin		
Methods	CP	ACC	G	CP	ACC	G
w/o Adaptive Retrieval	21.7	84.5	42.8	55.3	69.2	61.9
w/o Multi-key	21.5	84.7	42.7	55.6	70.3	62.5
w/o LoRA-based Initial	20.3	86.8	42.0	54.9	64.0	59.2
w/o SVD	23.1	83.1	43.8	55.2	67.1	60.9
Ours LLaMA2	22.1	88.5	44.2	57.9	74.0	65.4

Table 3: Ablation study based on LLaMA2-7B.

Effects of Weight Increment Settings. As mentioned in § 3.1, our core objective is to extract beneficial knowledge from the source style to provide better initialization for training. In the third row of Table 3, we ablated the LoRA parameter matrix and adopted random initialization, which resulted in a decline in overall performance. Combined with Figure 2c, a good initialization can indicate both the upper and lower limits of the ability, especially in low-resource scenarios.

In § 2.2, we chose LoRA to avoid increasing the network's depth. In Appendix C.1, we transfer the method to adapters and prompts, further analyzing the differences between LoRA and these methods. Additionally, LoRA can be applied to different parameter parts of the network. In Appendix C.2, we found that the FFN layer may effectively improve style accuracy. In this work, we applied weight increments to the output of the attention layer with a higher G-Score.



(a) SVD-based Interpolation (b) Rank(q) on Single Task

Figure 4: The left shows the performance of our method under different values of q. The right displays the performance of LoRA under various q singular vectors.

Effects of q on SVD. In the fourth row of Table 3, we show the results of ablating SVD. Mathematically, we replaced $\mathbf{U}_s^{(q)}\mathbf{\Sigma}_s^{(q)}\mathbf{V}_s^{(q)\top}$ in Equation 6 with $\mathbf{A}_s\mathbf{B}_s^{\top}$. The initial results indicate a significant performance improvement.

Furthermore, we analyzed how the selection of q works. We first conducted comparative experiments on a single task using the LoRA and the SVD-based LoRA method. In the comparison, we used efficient tuning without additional modules. The experimental results in Figure 4b show that as q increases, the performance of the SVD-based method gradually approaches that of the LoRA method. When q exceeds 16, the performance stabilizes and can be considered equivalent to the LoRA method. Subsequently, we retested the aforementioned task using the SVD-based Interpolation module. Figure 4a shows that the model's overall performance after interpolation surpasses that of the method on the right. However, as we discussed in §3.2, increasing q does not necessarily lead to performance improvement. Our findings align with Jiang et al. (2024), indicating that parameters influence each other during interpolation. Although increasing q can enhance the performance of a single task, suggesting that more information is contained within the parameters, the effectiveness of simply increasing information density diminishes when the total parameter count remains the same. By reducing q, we can make the parameter space sparser, thereby reducing interference and improving the model's overall performance.

Effects of Multi-key Retrieval. The first row of Table 3 shows the results of removing the retrieval score, setting $\mathcal{R}_s = 1$. The significant performance drop highlights the importance of Adaptive Retrieval for retrieving the most relevant knowledge. Adaptive Retrieval allows dynamic selection of the most suitable weight increments through query-key matching. The following row shows the results of ablating instance-level queries, specifically $\boldsymbol{q}^{\text{ins}}$ and $\boldsymbol{k}_s^{\Theta}$. We only used task-level weight increments for model initialization. The performance decline indicates that incorporating instance-level features indeed helps transfer the most useful knowledge to specific instances in the target task.

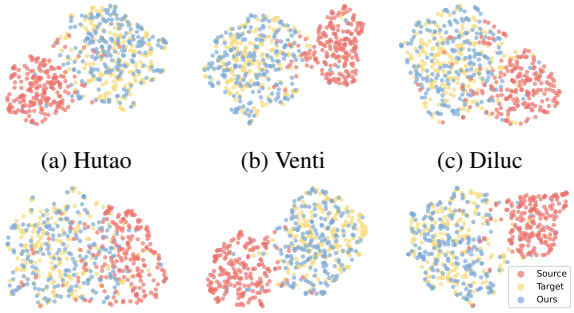
5.4 Human Evaluation

Methods	Style	Content	Fluency	**avg.Rank**
QLFT LLaMA2	2.34	1.76	**2.52**	2.21
FS Alpaca	3.36	3.90	3.40	3.56
FS ChatGPT-4	**2.02**	2.96	**2.00**	2.32
Ours LLaMA2	2.28	**1.38**	2.08	**1.92**

Table 4: Human evaluation on three metrics.

To supplement automatic metrics, we conducted human evaluations by sampling 50 instances from each dataset. The baselines used for comparison are: QLFT **LLaMA2**-7B (Dettmers et al., 2023), FS Alpaca-7B (Taori et al., 2023), and FS ChatGPT-4 (OpenAI, 2023). Five participants were enlisted to assess each model based on three criteria: (1) strength of style transfer, (2) semantic integrity, and (3) sentence fluidity. Performances were ranked from best-1 to worst-4. Table 4 presents the rankings of methods, as determined by participant feedback. In the Style Accuracy and Fluency assessment, participants seemed to struggle with making a clear choice; however, TWIST significantly outperformed the others in terms of Content.

5.5 Visualization



(a) Hutao (b) Venti (c) Diluc

(d) Noelle (e) Mona (f) Xiangling

Figure 5: 2D-Visualization of Stylistic Feature.

Stylistic Features. We visualize the stylistic features via 2D UMAP (McInnes et al., 2020) in Figure 5. We selected 12 features for visualization dimensions: these include three stylistic features, i.e., quantities of punctuation, the number of sentences, and the number of words (Zhu et al., 2023); and nine less-correlated vertical style types (Kang and Hovy, 2021), i.e., Humorous, Polite, Formal, Romantic, Gender, Dominance, Exciting, Sadness, and Offense. The first set of features is determined statistically, while the latter are evaluated using a classifier (Kang and Hovy, 2021). The results clearly illustrate that transformed texts (in BLUE) are distinctly different from the original text style (in RED) and closely align with the supervised target results (in GOLDEN).

Task Similarity. To visually represent the relationships and similarities among all generated key vectors, we present a similarity matrix visualization in Table 6. *C* represents the clustered key $\boldsymbol{k}^{\mathcal{C}}$, and others are weight key $\boldsymbol{k}^{\Theta}$. The visualization shows that keys in the same cluster typically exhibit higher similarity. The keys are roughly divided into five clusters, with the cluster containing *C2* representing the formality transfer task in two scenarios. The keys are approximately divided into two clusters for role dialogue style transfer. Through manual inspection, we found that roles in the cluster containing *C4* tend to be more aggressive, with shorter sentences and more varied interjections and punctuation. Conversely, roles in the cluster containing *C5* tend to be more conservative, using more formal language.

Case Study. We also provide some examples in Appendix E to demonstrate the effectiveness of our method.

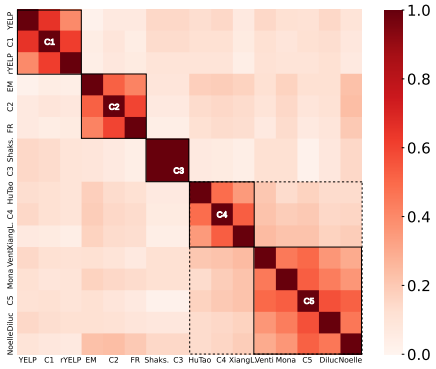


Figure 6: The similarity matrix for generated keys.

6 Related Work

Text Style Transfer Methods. TST is a crucial area in natural language processing. It aims to endow the text with a new style without altering its meaning (Riley et al., 2021). From the holistic definition of style, current work can be divided into two paradigms (Zhu et al., 2023). The first paradigm decouples explicit style from content, as in Xu et al. (2023), which added an additional adversarial loss function. (Syed et al., 2020) randomly dropped input words and reconstructed the input for each author separately. The second paradigm avoids explicitly decoupling style from content. Dai et al. (2019) added extra style embeddings in the input. Yi et al. (2021) employs generative flow techniques to extract stylistic features from instances of styles.

Style Categories. Previous studies primarily focused on simple style transfer tasks, like formality (Rao and Tetreault, 2018) or sentiment transfer (Xu et al., 2020b; Sudhakar et al., 2019). These tasks have achieved satisfactory results via word-level transfers. Recently, Xu et al. (2023) defined the cross-style as a compound style, such as personality (Shao et al., 2023) and writing styles(Zhu et al., 2023; Tao et al., 2024). It depends on the high-level representations to capture stylistic representation effectively.

7 Conclusion

This paper addresses the challenges of style transfer in the text by proposing a novel method called TWIST, which utilizes transferable weight increments for style text generation. Our experiments demonstrate that TWIST outperforms baselines across various models, achieving state-of-the-art performance and superior adaptability in low-resource scenarios.

8 Limitations

In this paper, we introduce TWIST, a method for style transfer in text generation. However, our method introduces an additional retrieval framework, which may increase computational and memory costs. This overhead is relatively minor compared to the resource demands of large models used for inference. Further quantitative analysis of this weakness is provided in Appendix D. In addition, using LoRA as the format of weight increments may not be optimal and introduces extra parameters, and the tunable parameter quantity varies across different parts of the model. We further analyze this in Appendix C.1. Lastly, our current evaluation of TWIST has been limited to English-language tasks. We recognize the importance of assessing its performance in other languages to understand its broader applicability and intend to test TWIST on non-English tasks in future work.

9 Ethics Statement

TWIST enhances the ability to transfer style in environments with limited data resources. All experiments were conducted using widely-used general datasets, which are unlikely to contain harmful content. However, using role styles may carry certain risks, as malicious actors could potentially exploit TWIST for activities such as fraud. Additionally, the styles can be influenced by the datasets users provide, which might contain harmful content, biases, or privacy issues.

For human evaluation, we enlisted five native English speakers as volunteers to assess the generated texts. These volunteers have a background in literary creation, enabling them to distinguish different text styles. We did not ask for personal information or collect any private data from the volunteers.

References

Akari Asai, Mohammadreza Salehi, and Matthew E. Peters. 2022. Attempt: Parameter-efficient multi-task tuning via attentional mixtures of soft prompts.

Xin Chen and Yongfeng Huang. 2024. Tiny-stylewizard: Unleashing the potential of small language models in complex style transfer.

Ning Dai, Jianze Liang, Xipeng Qiu, and Xuanjing Huang. 2019. Style transformer: Unpaired text style transfer without disentangled latent representation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5997–6007. Association for Computational Linguistics.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. Qlora: Efficient finetuning of quantized llms.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-agnostic meta-learning for fast adaptation of deep networks.

Junxian He, Xinyi Wang, Graham Neubig, and Taylor Berg-Kirkpatrick. 2020. A probabilistic formulation of unsupervised text style transfer.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep residual learning for image recognition.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for nlp.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models.

Gabriel Ilharco, Marco Tulio Ribeiro, Mitchell Wortsman, Suchin Gururangan, Ludwig Schmidt, Hannaneh Hajishirzi, and Ali Farhadi. 2023. Editing models with task arithmetic.

Parag Jain, Abhijit Mishra, Amar Prakash Azad, and Karthik Sankaranarayanan. 2019. Unsupervised controllable text formalization.

Weisen Jiang, Baijiong Lin, Han Shi, Yu Zhang, Zhenguo Li, and James Kwok. 2024. Effective and parameter-efficient reusing fine-tuned models.

Dongyeop Kang and Eduard Hovy. 2021. Style is not a single variable: Case studies for cross-style language understanding.

Siyu Lai, Zhen Yang, Fandong Meng, Yufeng Chen, Jinan Xu, and Jie Zhou. 2022. Cross-align: Modeling deep cross-lingual interactions for word alignment. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 3715–3725, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Juncen Li, Robin Jia, He He, and Percy Liang. 2018. Delete, retrieve, generate: a simple approach to sentiment and style transfer. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. Association for Computational Linguistics.

Junyi Li, Tianyi Tang, Jian-Yun Nie, Ji-Rong Wen, and Wayne Xin Zhao. 2022. Learning to transfer prompts for text generation.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization.

Michael Luca. 2016. Reviews, reputation, and revenue: The case of yelp. com. *Com (March 15, 2016). Harvard Business School NOM Unit Working Paper*, (12-016).

Leland McInnes, John Healy, and James Melville. 2020. Umap: Uniform manifold approximation and projection for dimension reduction.

Alexander Miller, Adam Fisch, Jesse Dodge, Amir-Hossein Karimi, Antoine Bordes, and Jason Weston. 2016. Key-value memory networks for directly reading documents.

Tong Niu and Mohit Bansal. 2018. Polite dialogue generation without parallel data.

OpenAI. 2023. Gpt-4 technical report.

Bohao Peng, Zhuotao Tian, Shu Liu, Mingchang Yang, and Jiaya Jia. 2024. Scalable language model with generalized continual learning.

Matt Post. 2018. A call for clarity in reporting BLEU scores. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 186–191, Brussels, Belgium. Association for Computational Linguistics.

Sudha Rao and Joel Tetreault. 2018. Dear sir or madam, may I introduce the GYAFC dataset: Corpus, benchmarks and metrics for formality style transfer. pages 129–140, New Orleans, Louisiana. Association for Computational Linguistics.

Parker Riley, Noah Constant, Mandy Guo, Girish Kumar, David Uthus, and Zarana Parekh. 2021. Textsettr: Few-shot text style extraction and tunable targeted restyling.

Sebastian Ruder, Matthew E. Peters, Swabha Swayamdipta, and Thomas Wolf. 2019. Transfer learning in natural language processing. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Tutorials*. Association for Computational Linguistics.

Andrew M. Saxe, James L. McClelland, and Surya Ganguli. 2014. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks.

Yunfan Shao, Linyang Li, Junqi Dai, and Xipeng Qiu. 2023. Character-llm: A trainable agent for role-playing.

Akhilesh Sudhakar, Bhargav Upadhyay, and Arjun Maheswaran. 2019. "transforming" delete, retrieve, generate approach for controlled text style transfer. pages 3269–3279. Association for Computational Linguistics.

Mirac Suzgun, Luke Melas-Kyriazi, and Dan Jurafsky. 2022. Prompt-and-rerank: A method for zero-shot and few-shot arbitrary textual style transfer with small language models.

Bakhtiyar Syed, Gaurav Verma, Balaji Vasan Srinivasan, Anandhavelu Natarajan, and Vasudeva Varma. 2020. Adapting language models for non-parallel author-stylized rewriting.

Zhen Tao, Dinghao Xi, Zhiyu Li, Liumin Tang, and Wei Xu. 2024. Cat-llm: Prompting large language models with text style definition for chinese article-style transfer.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford alpaca: An instruction-following llama model. `https://github.com/tatsu-lab/stanford_alpaca`.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, and Nikolay Bashlykov. 2023. Llama 2: Open foundation and fine-tuned chat models.

Tu Vu, Brian Lester, Noah Constant, Rami Al-Rfou, and Daniel Cer. 2022. Spot: Better frozen model adaptation through soft prompt transfer.

Zekun Moore Wang, Zhongyuan Peng, Haoran Que, Jiaheng Liu, Wangchunshu Zhou, and Yuhan Wu. 2024. Rolellm: Benchmarking, eliciting, and enhancing role-playing abilities of large language models.

Zifeng Wang, Zizhao Zhang, Chen-Yu Lee, Han Zhang, Ruoxi Sun, Xiaoqi Ren, Guolong Su, Vincent Perot, Jennifer Dy, and Tomas Pfister. 2022. Learning to prompt for continual learning.

Jingjing Xu, Xu Sun, and Qi and Zeng. 2018. Unpaired sentiment-to-sentiment translation: A cycled reinforcement learning approach. Melbourne, Australia. Association for Computational Linguistics.

Peng Xu, Jackie Chi Kit Cheung, and Yanshuai Cao. 2020a. On variational learning of controllable representations for text without supervision.

Peng Xu, Jackie Chi Kit Cheung, and Yanshuai Cao. 2020b. On variational learning of controllable representations for text without supervision. volume 119 of *Proceedings of Machine Learning Research*, pages 10534–10543. PMLR.

Ruiqi Xu, Yongfeng Huang, Xin Chen, and Lin Zhang. 2023. Specializing small language models towards complex style transfer via latent attribute pre-training. In *ECAI 2023*, pages 2802–2809. IOS Press.

Prateek Yadav, Derek Tam, Leshem Choshen, Colin Raffel, and Mohit Bansal. 2023. Ties-merging: Resolving interference when merging models.

Xiaoyuan Yi, Zhenghao Liu, Wenhao Li, and Maosong Sun. 2021. Text style transfer via learning style instance supported latent space. In *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, pages 3801–3807.

Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. 2014. How transferable are features in deep neural networks? *Advances in neural information processing systems*, 27.

Mingyang Zhang, Hao Chen, Chunhua Shen, Zhen Yang, Linlin Ou, Xinyi Yu, and Bohan Zhuang. 2023. Loraprune: Pruning meets low-rank parameter-efficient fine-tuning.

Xuekai Zhu, Jian Guan, Minlie Huang, and Juan Liu. 2023. Storytrans: Non-parallel story author-style transfer with discourse representations and content enhancing.

A Classifier Accuracy for Styles

In this section, following Riley et al. (2021), we trained a classifier to evaluate the styles. Table 5, mentioned in § 4.2, presents the classifier used for automated style testing in our main experiments. Table 6 shows the vertical style classifier trained for visualization purposes, using data from (Kang and Hovy, 2021) and others.

Style	**Dataset**	**Accuracy** %
Sentiment	YELP	96.7
Formality	GYAFC	89.4
Writing	Shakespeare	83.5
Role	Genshin	71.6

Table 5: Classifier for Main Experiments.

Style	**Dataset**	**Accuracy** %
Humorous	ShortHumor	97.3
Polite	SPolite	69.0
Formal	GYAFC	89.4
Sadness	DailyDialog	93.7
Romantic	ShortRomance	99.1
Gender	PASTEL	47.1
Dominance	EmoBank	43.6
Exciting	DailyDialog	98.2
Offense	HateOffens	91.9

Table 6: Classifier for Stylistic Features Visualization.

B Prompt Design for Few-shot Methods

The current closed-source LLMs demonstrate outstanding few-shot performance, but this depends on carefully crafted prompt engineering. In Table 17, we have designed a series of prompt schemes aimed at fully harnessing the potential of these large models. We selected 100 sets of examples and used the classifier for style verification. The results are shown in Table 7. In the main text, we selected prompt 3 as the comparison baseline, which achieved the highest style accuracy.

Dataset	**Shakespeare**	**Genshin**
Methods	ACC	ACC
FS Prompt1 Alpaca	73.0	57.0
FS Prompt2 Alpaca	71.0	54.0
FS Prompt3 Alpaca	78.0	62.0
FS Prompt1 ChatGPT-4	74.0	63.0
FS Prompt2 ChatGPT-4	77.0	67.0
FS Prompt3 ChatGPT-4	85.0	73.0

Table 7: Test for Designed Prompts.

C Effects of Various Weight Increments

This section analyzes how different settings for weight increments can lead to varying impacts.

C.1 Usage of Weight Increments

This section explores further details and compares various parameter-efficient fine-tuning methods. We substitute our LoRA-baed weight increment with other components and conduct ablation experiments on various tasks. For a new TST task t, we use the Adapter to replace LoRA, and the objective is to minimize the probability:

$$\mathcal{L}(\Delta\boldsymbol{\Theta}_t^k) = -\sum_{i=1}^{N} \log \Pr(\boldsymbol{y}_i \mid \boldsymbol{x}_i; \boldsymbol{\theta}_0, \Delta\boldsymbol{\Theta}_t^k) \quad (8)$$

where $\boldsymbol{\theta}_0$ is frozen and we optimize the adapter parameters $\Delta\boldsymbol{\Theta}_t^k$. For the prompts, we chose the prefix-tuning approach, and the optimization process is as follows:

$$\mathcal{L}(\mathcal{P}_t^k) = -\sum_{i=1}^{N} \log \Pr(\boldsymbol{y}_i \mid [\mathcal{P}_t^k, \boldsymbol{x}_i]; \boldsymbol{\theta}_0) \quad (9)$$

where $\mathcal{P}_t^k$ is a trainable prompt. We only used interpolation methods instead of SVD for our ablation experiments, as singular vectors are unsuitable for single-dimensional prompts. All other settings are consistent with the main text, and the experimental results on LLaMA2 are shown in Table 8.

Dataset	Shakespeare			Genshin			
Methods	CP	ACC	G	CP	ACC	G	**#params**
LoRA	23.2	83.1	43.8	55.2	67.1	60.9	32M
Adapter	18.2	82.5	38.7	50.1	60.7	55.2	40M
Prompt	19.7	75.4	38.5	46.7	61.0	53.4	1.7M

Table 8: Comparison between PEFT methods.

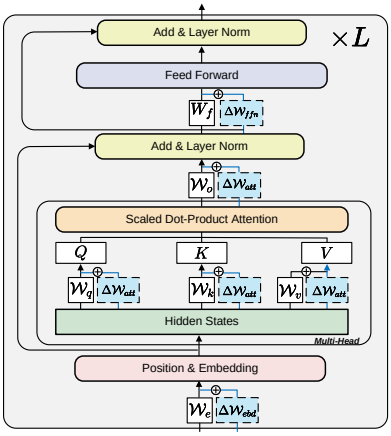


Figure 7: Additional Parameter Parts.

C.2 Additional Parameter Parts

In this paper, we use low-rank techniques to preserve transferable knowledge. Compared to other parameter-efficient methods, such as Adapters. LoRA can operate across different parameter parts, as shown in Figure 7. We selected three distinct parts: the embedding layer, the attention layer, and the feed-forward layer. In our ablation experiments, we adjusted the rank size as much as possible to ensure that the number of trainable parameters across the three groups remained consistent.

Table 13 shows that different parameter parts slightly impact performance. However, using the attention layer as the additional parameter was more stable overall. Moreover, while the feed-forward layer sometimes achieves higher style accuracy, it tends to reduce content preservation.

C.3 Comparison between Pruning and SVD

In preliminary experiments, we attempted to use pruning techniques to sparsify LoRA and reduce parameter interference. However, this did not achieve the desired results and even resulted in some performance loss. The results are shown in Table 9.

D Consumption of Additional Calculation

The additional overhead mainly consists of two parts: first, the generation and retrieval key process

Dataset	Shakespeare			Genshin		
Methods	CP	ACC	G	CP	ACC	G
Interpolation	**23.1**	83.1	43.8	55.2	67.1	60.9
Interpolation + Pruning	19.2	80.1	39.2	51.2	65.7	58.0
Interpolation + SVD	22.1	**88.5**	**44.2**	**57.9**	**74.0**	**65.4**

Table 9: Comparison between Pruning and SVD.

requires extra time; Second, the weight pool necessitates additional storage for these weights, which consumes memory resources.

Time Analysis. First, we measure the time taken to generate each key in Figure 5, using the same settings as in §4.2. The detailed results are shown in Table 14. Additionally, we measure the time required to retrieve the weight increments for the corresponding tasks, as shown in Table 10.

Task	Shakespeare	**avg**.Genshin
Times	8.2 ms	6.5 ms

Table 10: Retrieval weight increments time of Tasks.

Memory Usage. We measured the additional memory space occupied by the source weight pools of T5 and LLaMA2. The results are presented in Table 11. In fact, although our method introduces additional weights, these weights are modular and can be easily added or removed. Each weight increment accounts for about 0.8% of the total, which is negligible compared to the vast parameter space of large pre-trained models.

Method	Memory Space
T5-Large	0.23 GB
LLaMA2-7B	1.52 GB

Table 11: Additional memory space.

E Case Study

We provide several cases in Figure 8 comparing the outputs of ChatGPT-4 with TWIST based on LLaMA2-7B. Our method tends to add common words and punctuation to create a specific role style, whereas ChatGPT-4 prefers incorporating the particular role background.

F Detailed Tables

Following are some tables referenced in the text.

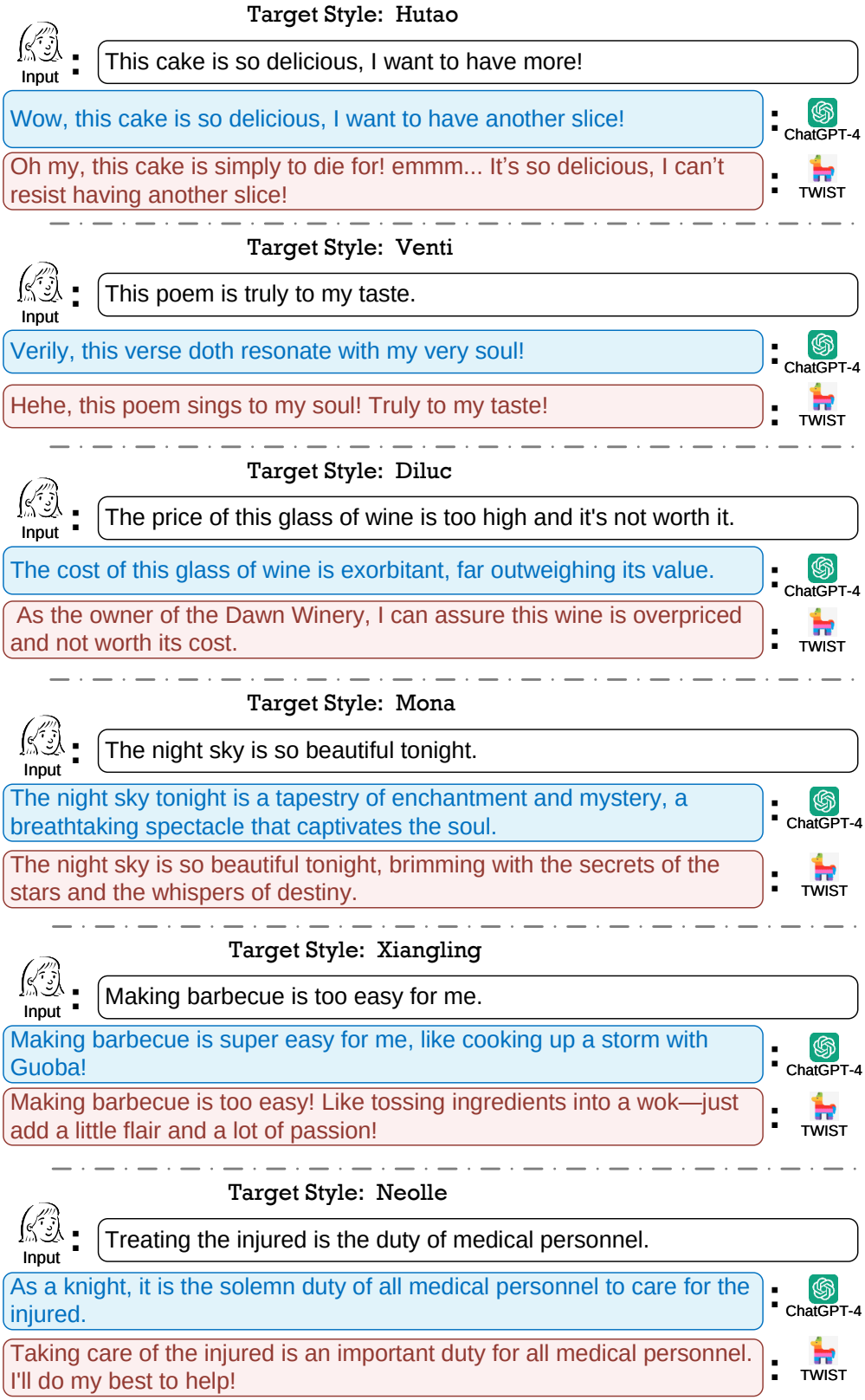


Figure 8: Case study for comparing ChatGPT-4 and TWIST.

Dataset	Attributes	Train	Dev	Test
YELP	Positive → Negative	440K	10K	3,000
GYAFC E&M	Formal → Informal	52,595	2,877	1,416
GYAFC F&R	Informal → Formal	51,967	2,788	1,332
Shakespeare	None → Auther	2322	291	291
Genshin	None → Role(6)	1653	223	223

Table 12: Statistics of our datasets after pre-processing.

Dataset	**Hutao**			**Venti**			**Diluc**			**Noelle**			**Mona**			**Xiangling**		
Layer	CP	ACC	G	CP	ACC	G	CP	ACC	G	CP	ACC	G	CP	ACC	G	CP	ACC	G
Embedding	56.3	74.2	64.6	57.9	73.2	65.1	59.2	76.8	67.4	56.7	74.1	64.8	54.5	69.8	61.7	56.4	64.7	60.4
Feed Forward	56.8	75.0	65.3	57.6	**74.9**	65.7	58.7	**79.3**	68.2	57.2	73.8	65.0	53.2	**70.9**	61.4	56.2	69.9	62.7
Attention	**57.2**	**75.1**	**65.5**	**58.6**	74.8	**66.2**	**60.3**	78.5	**68.8**	**58.5**	**74.5**	**66.0**	**55.7**	70.6	**62.7**	**56.8**	**70.4**	**63.2**

Table 13: Ablation experiments results of various parameter parts.

Keys	Yelp	EM1	EM2	Shak.	R1	R2	R3	R4	R5	R6	**avg**
Times(min)	6.1	3.7	4.1	2.6	2.1	1.5	1.3	1.3	1.4	1.2	2.5

Table 14: Generation time of keys.

Dataset	**Hutao**			**Venti**			**Diluc**			**Noelle**			**Mona**			**Xiangling**		
Methods	CP	ACC	G	CP	ACC	G	CP	ACC	G	CP	ACC	G	CP	ACC	G	CP	ACC	G
FT T5-Large	52.4	57.6	54.9	53.8	55.1	54.4	52.6	52.9	52.7	52.3	58.3	55.2	53.5	55.7	54.6	54.0	54.0	54.0
CrossAligned	15.7	54.2	29.2	17.1	53.6	30.3	12.5	66.2	28.8	18.4	61.8	33.7	13.6	55.7	27.5	10.7	53.2	23.9
Delete&Retrieve	57.6	47.2	52.1	58.2	48.2	53.0	51.2	51.8	51.5	53.7	58.2	55.9	50.2	48.9	49.5	53.5	61.8	57.5
B-GST	29.8	59.4	42.1	29.9	61.6	42.9	31.2	59.2	43.0	31.1	59.8	43.1	30.0	60.8	42.7	32.2	59.8	43.9
TextSETTR	34.7	51.4	42.2	31.7	53.0	41.0	32.0	52.5	41.0	31.0	49.9	39.3	34.3	55.1	43.5	31.3	53.7	41.0
Ours T5-Large	53.7	60.1	56.8	57.1	54.7	55.9	53.7	60.2	56.9	54.3	60.7	57.4	55.9	61.8	58.8	56.8	63.4	60.0

Table 15: Comparison of Full Dataset in TST between small-scale PLM methods.

Dataset	**Hutao**			**Venti**			**Diluc**			**Noelle**			**Mona**			**Xiangling**		
Methods	CP	ACC	G	CP	ACC	G	CP	ACC	G	CP	ACC	G	CP	ACC	G	CP	ACC	G
QLFT LLaMA2	53.5	68.7	60.6	56.9	68.2	62.3	53.2	61.2	57.1	54.8	60.2	57.4	54.5	61.8	58.0	56.2	63.8	59.9
FS Alpaca	39.2	71.5	52.9	41.3	70.5	54.0	40.2	65.9	51.5	43.2	67.8	54.1	41.2	71.4	54.2	43.7	68.7	54.8
FS ChatGPT-4	42.3	79.3	57.9	44.7	75.4	58.1	45.2	80.2	60.2	39.6	73.2	53.8	40.9	75.9	55.7	45.0	73.5	57.5
Ours LLaMA2	57.2	75.1	65.5	58.6	74.8	66.2	60.3	78.5	68.8	58.5	74.5	66.0	55.7	70.6	62.7	56.8	70.4	63.2

Table 16: Comparison of Full Dataset in TST between large-scale PLM methods.

Baseline	Prompt
Prompt 1	**System:**[Role][Background][Personality] **User:**This is a dialogue attributed to the character [role]: [Golden Text 1] [Golden Text 2] [.....] [Golden Text n] In light of the [role]'s background, distinctive personality traits, and other relevant information, coupled with the given dialogue, you are tasked with precisely emulating the stylistic and tonal aspects of [role]'s speech. The text you are required to mimic is as follows: [Original Text] Please follow the requirements:[Requirements] **Assistant:**[Generated Text]
Prompt 2	**User:**There are two paragraphs from different characters. Rewrite the target paragraph according to the speaking style, personality, mood, word usage and punctuation of the origin paragraph. Pay attention to imitating the speaking style of the first paragraph as much as possible and use different words. Here is an example. Original Paragraph:[Original Text 1] Target Paragraph:[Golden Text] The text you are required to rewrite is as follows: [Original Text 2] Please follow the requirements:[Requirements] **Assistant:**[Generated Text]
Prompt 3	*Supervised History:* **System:**[Role][Background][Personality] **User:**In light of the [role]'s background, distinctive personality traits, and other relevant information, coupled with the given dialogue, you are tasked with precisely emulating the stylistic and tonal aspects of [role]'s speech. The text you are required to mimic is as follows: [Original Text] Please follow the requirements:[Requirements] **User:**[Original Text 1] **Assistant:**[Golden Text 1] **User:**[Original Text 2] **Assistant:**[Golden Text 2] ..... **User:**[Original Text n] **Assistant:**[Golden Text n] *Generating:* **User:**[Original Text] **Assistant:**[Generated Text]

Table 17: Prompts for Few-shot LLMs methods.

Algorithm 1: The Training Pipeline of TWIST

Input: Source task set $\{s_1, \ldots, s_S\}$, target task t, correspond training sets $\{(\boldsymbol{x}_i, \boldsymbol{y}_i)_{i=1}^N\}_k^t \sim \mathcal{D}_k$
Output: Cluster key $\boldsymbol{k}_c^{\mathcal{C}}$, weight key $\boldsymbol{k}_s^{\Theta}$, source weight increments $\Delta\boldsymbol{\theta}_s$, source weight pool $\{\boldsymbol{k}_c^{\mathcal{C}}; \boldsymbol{k}_s^{\Theta}; \Delta\boldsymbol{\theta}_s\}_{c=1}^C$; Target weight increments $\Delta\boldsymbol{\theta}_t$

// Following are the construction of source weight pool (§ 3.1)
for $s = 1, \ldots, S$ **do**
  Initialize the s-th source weight key $\boldsymbol{k}_s^{\Theta}$ and correspond weight increments $\Delta\boldsymbol{\theta}_s$
  for $(\boldsymbol{x}_i, \boldsymbol{y}_i)_{i=1}^N \in \mathcal{D}_s$ **do**
    Use LoRA-based parameter efficient method to obtain source weight increments $\Delta\boldsymbol{\theta}_s = \mathbf{A}_s\mathbf{B}_s^\top$ via Eq. 3
    Use singular value decomposition to approximate $\mathbf{A}_s\mathbf{B}_s^\top$ as $\mathbf{U}_s\mathbf{\Sigma}_s\mathbf{V}_s^\top$
    Initialize weight key $\boldsymbol{k}_s^{\Theta}$ and calculate query via $\boldsymbol{q} = \boldsymbol{f}_{\text{BERT}}(\boldsymbol{x_i})$
    Use K-nearest neighbors algorithm to obtain the top-W most similar keys $\boldsymbol{K}^{\Theta} = \{\boldsymbol{k}_1^{\Theta}, \ldots, \boldsymbol{k}_W^{\Theta}\}$
    Upgrade source weight key $\boldsymbol{k}^{\Theta} \in \boldsymbol{K}^{\Theta}$ via $\boldsymbol{k}^{\Theta} = \boldsymbol{k}^{\Theta} + \gamma\nabla_{\boldsymbol{k}^{\Theta}} \cos(\boldsymbol{q}, \boldsymbol{k}^{\Theta})$
    Obtain key-value pair $[\boldsymbol{k}_s^{\Theta}; \Delta\boldsymbol{\theta}_s]$
  end
end
Use spectral clustering algorithm to obtain cluster $\mathcal{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_C\}$ and $\Delta\boldsymbol{\theta}_s^n \in \mathcal{C}^n$
for $c = 1, \ldots, C$ **do**
  Calculate cluster key via $\boldsymbol{k}_c^{\mathcal{C}} = \left(\prod_{i=1}^n \boldsymbol{k}_i^{\Theta}\right)^{\frac{1}{n}}$ and n is the key number in $\mathcal{C}^c$
  Align multi-keys and values to update source weight pool $\{\boldsymbol{k}_c^{\mathcal{C}}; \boldsymbol{k}_s^{\Theta}; \Delta\boldsymbol{\theta}_s\}$
end
// Following are reusing LoRA-based weight increments for target style generation (§ 3.2)
for $s = 1, \ldots, S$ **do**
  Embed task-level query $\boldsymbol{q}^{\text{task}}$ and calculate instance-level query via $\boldsymbol{q}^{\text{ins}} = \frac{1}{N}\sum_{\boldsymbol{x}\in\mathcal{X}} \boldsymbol{f}_{\text{BERT}}(\boldsymbol{x})$
  Calculate Retrieval Score $\mathcal{R}_s = \text{softmax}(\lambda \cdot \boldsymbol{q}^{\text{task}} \cdot \boldsymbol{k}_s^{\mathcal{C}} + (1-\lambda) \cdot \boldsymbol{q}^{\text{ins}} \cdot \boldsymbol{k}_s^{\Theta})$ as Eq. 5
end
Use interpolation to obtain initial target weight increments $\Delta\boldsymbol{\theta}_t = \sum_{s=1}^S \mathcal{R}_s \cdot \Delta\boldsymbol{\theta}_s$ as Eq. 6
Update $\boldsymbol{\theta}_t$ for target style transfer task t via $\mathcal{L}_{\Delta\boldsymbol{\theta}_t^k}(\mathcal{D}_k) = -\sum_{i=1}^N \log \Pr(\boldsymbol{y}_i \mid \boldsymbol{x}_i; \boldsymbol{\theta}_0, \Delta\boldsymbol{\theta}_t^k)$ as Eq. 7