DSBA 6165 AI & Deep Learning

Spring 2025

Final Report

Personalised AI Writing Assistant

Group 8:

Brinda Vijayakumar

Lasya Reddy Edunuri

Manasa Sree Vuppu

Project Github Link

Index

I. Introduction:

Problem Statement:

Generating high-quality personalized writing remains one of the most complex and omnipresent problems in today's world of generative writing. Despite the growth and rapid development of Large Language Models (LLMs), generating text that consistently reflects an individual's unique style and tone, preserving the nature of their personal voice, remains an open challenge. Commonly, generic output, lack of emotional resonance, and limited nuance to author-specific styles significantly limit the application of AI for content creators, marketers, educators, or prospective users. Most LLMs share a history of training on large, heterogeneous corpora and are optimized to achieve general fluency and coherent grammar, often at the expense of personalization.

Our motivation for identifying and solving this problem is twofold. With five years of firsthand experience in content creation and writing, the team recognizes the limitations of current resources while also identifying greater scope for fine-grained style transfer and tone preservation in domains such as marketing, education, advertising, and more. The challenge of preserving stylistic fidelity and personal nuance still persists, even as LLMs like GPT and T5 have revolutionized the field of content generation.

One of the biggest challenges in personalized content generation is maintaining the writer's unique tone. For example, if a user typically writes reflective quotes like "Even silence tells a story," a generic model might produce something flat like "Be quiet and think," which lacks nuance and emotional depth. Models also struggle to adapt style across formats (e.g., from quotes to essays), especially when training data is limited or unlabeled. Lastly, evaluating whether AI output truly sounds like the original author remains difficult without stylistic benchmarks.

To solve this problem, the team curated a personalized dataset of quotes, blogs, essays, and letters, all belonging to one of the team members who happened to be a writer in the past, and developed an end-to-end AI writing assistant. The approach combines few-shot prompting, instruction tuning, metadata-aware conditioning, contrastive scoring and fine-tuning with LoRA-based models. This system not only generates emotionally resonant content but also validates stylistic authenticity using an authorship classifier.

II. Background and Literature review:

We reviewed prior research papers on prompt engineering, language models, style transfer, author attribution and parameter-efficient fine-tuning to inform our model choices, fine-tuning

strategies and evaluation metrics. The following summaries outline the most influential works that shaped our methodology.

1. *Prompting and Model Performance:* An essential aspect influencing AI-generated text quality is prompt engineering. The comprehensive survey on prompting methods by Liu et al. (2023) categorizes and analyzes prompting strategies in NLP. The survey highlights how carefully designed prompts significantly enhance model performance across various NLP tasks, guiding prompt engineering practices employed in this project.

   Pro: Boosts performance without retraining the model.
   Con: Prompt effectiveness may vary across tasks and models.
   Relation to our work: Informed our use of few-shot prompting and instruction tuning to personalize tone and structure without overfitting.

2. *Innovations in Style Transfer and Conditioning:* TextSETTR Framework by Riley et al. (2020) presents a few-shot approach to text style extraction and targeted restyling without extensive style-labeled data, adapting the T5 model for effective style vector extraction. This approach significantly influenced the project's implementation of metadata-driven prompt conditioning.

   Pro: Requires minimal labeled data and captures nuanced style.
   Con: Sensitive to prompt design and limited in structure control.
   Relation to our work: Directly inspired our implementation of metadata-aware generation in T5 and post-hoc ranking using cosine similarity.

3. *Efficient Fine-Tuning Strategies:* The use of QLoRA in Machine Translation by Zhang et al. (2023) demonstrates efficient fine-tuning of LLMs for enhanced performance over zero-shot and few-shot methods, reinforcing the project's strategic use of QLoRA for fine-tuning models, thereby achieving higher quality outputs in style-specific text generation.

   Pro: Memory-efficient and scalable to large models.
   Con: Limited public tools and requires careful hyperparameter tuning.
   Relation to our work: Supported our decision to fine-tune long-form models like Mistral using QLoRA to retain stylistic consistency.

4. *Style-Specific Generation:* StyleMC Model by Khan et al. (2023) leverages contrastively trained representations capturing stylometric features, significantly improving stylistic consistency in generated texts. Similarly, Sawicki et al. (2023) examined GPT-3's capability to produce high-quality poetry in specific styles through fine-tuning, metadata conditioning shaping our strategy on metadata-driven prompting.

   Pro: Enables quantifiable evaluation of style alignment.

Con: Requires stylistically diverse examples to train contrastive models effectively.
Relation to our work: Inspired by the STYLEMC-based energy scoring and multi-version reranking pipeline implemented in our BART model for short-form generation.

5. *Instruction Tuning for Generalization:* Instruction tuning by Wei et al. (2021) demonstrates substantial improvements in zero-shot task generalization by fine-tuning language models on diverse instruction-based tasks. This method validates the project's choice to implement instruction-tuning methods to enhance adaptability and performance in generating personalized, contextually relevant text. It also informed the hybrid design of our system using the Meta Llama model, where task specific fine-tuning is combined with few-shot prompting during inference to improve personalization and alignment with user intent.

   Pro: Enables multi-task performance and domain flexibility.
   Con: Requires broad and well-structured instruction datasets.
   Relation to our work: Used in long-form generation where prompts followed structured formats to improve narrative alignment and task relevance. It validated the use of inference-time prompts after fine-tuning in Meta Llama to dynamically guide SOP generation, supporting our hybrid approach for controllable and context-sensitive text generation.

6. *Low-Rank Adaptation (LoRA)* by Hu et al. (2022) introduces a parameter-efficient fine-tuning method for large language models (LLMs). This approach involves freezing pre-trained weights and injecting trainable rank decomposition matrices into each layer, significantly reducing the number of trainable parameters and GPU memory requirements without compromising performance.

   Pro: Drastically lowers fine-tuning costs without loss of performance.
   Con: May need adjustment of learning rates and dropout for different architectures.
   Relation to our work: Enabled us to train multiple long-form models (Meta Llama, TinyLlama, Mistral) within resource constraints.

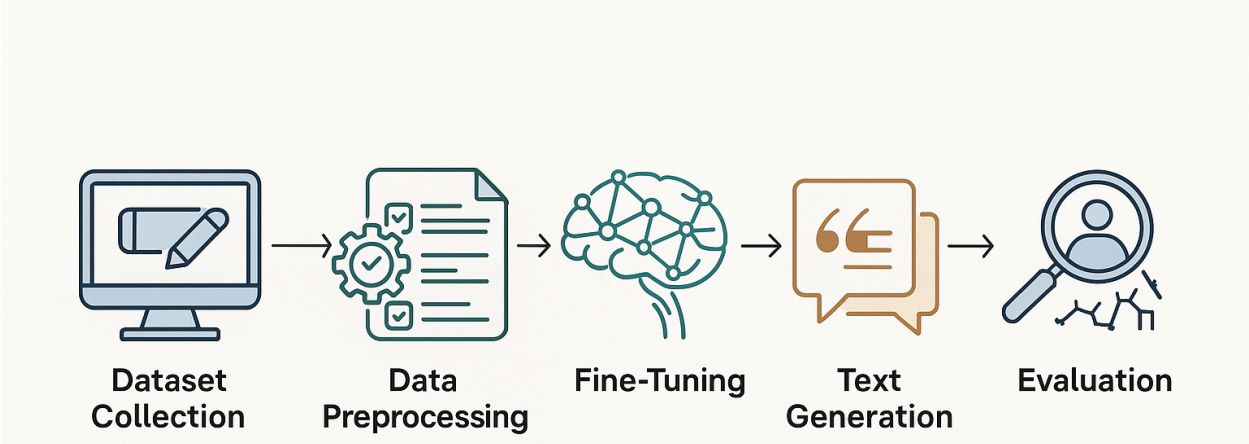
7. *Authorship Attribution and Stylistic Analysis:* The authorship attribution study by Aborisade and Anwar (2018) explores logistic regression and Naive Bayes classifiers for authorship classification in tweets. Their findings indicate logistic regression's superior performance, emphasizing its applicability and effectiveness in authorship classification contexts similar to the project's needs.

   Pro: High accuracy and interpretability with small datasets.
   Con: May underperform with imbalanced or long-form data.
   Relation to our work: Guided our selection of logistic regression for validating AI-generated text against true author writing.

The literature review provided a robust theoretical and methodological foundation for our project's design, influencing model selection, fine-tuning approaches, authorship attribution methodologies, and prompt engineering strategies. By combining insights from both foundational frameworks and recent innovations, we built a system that balances personalization, fluency, efficiency, and evaluability.

III. Methods and Algorithms:

Figure 1: Personalized Text Generation Pipeline



Summary of Pipeline Stages:

Dataset Collection: We gathered a personalized corpus of writings that includes blogs, quotes, academic letters, and SOPs authored by one team member, which became the *primary dataset*. This served as the foundational data for both short-form and long-form generation tasks. A *secondary dataset* was compiled using works from other authors and additional samples from our own writing, intended to help distinguish the user's original work from the rest.

Data Preprocessing: Text was cleaned, deduplicated, and annotated with metadata such as tone (e.g., empathetic, wise), text type (quote, SOP), structure, and length. This tagging enabled downstream models to better capture stylistic and emotional cues. Also to extract text data from images, a macOS-native OCR pipeline was implemented using Apple's Vision framework via PyObjC to extract text from a folder of quote images. Our script loads each image using NSImage from AppKit, converts it to CGImage, and processes it with VNRecognizeTextRequest from the Vision framework to perform high-accuracy text recognition. The extracted text is then stored alongside the corresponding filename in a Pandas DataFrame and exported to a CSV file. This approach enables offline, efficient, and accurate extraction of text content from image-based

quotes datasets, facilitating further analysis or integration into natural language processing workflows.

Fine-Tuning: We fine-tuned models such as GPT-2, BART, TinyLlama, and Mistral using Parameter-Efficient Fine-Tuning methods (LoRA, QLoRA). Model variants were tailored to support either short-form generation (quotes) or long-form generation (SOPs). Additionally, a logistic regression model was developed to distinguish between our own work and that of other authors.

Text Generation: Based on the format and tone of user prompts, our pipeline generated text in two modes, short quotes (reflecting tone and style) and long-form essays like SOPs (preserving structure, purpose, and voice).

Evaluation: We assessed model outputs through stylistic classifiers, cosine similarity scoring, BERTScore, ROUGE and human interpretation. Metrics such as tone alignment, semantic closeness, and authorship attribution helped evaluate how well the generated text reflected the original author's voice.

Exploratory Data Analysis of Dataset 1 (Team Member's Original Writings):

The primary dataset was loaded from a manually prepared CSV (manasa_quotes&sops_final.csv). Each entry was categorized under a Document Type label, such as Quote, SOP, Academic Letter, or Blog, and a small number of ambiguous entries (e.g., labeled Unknown) were cleaned or dropped from the final version.

Preprocessing focused on cleaning the text, removing noise, and tagging content for downstream modeling. Additional exploratory steps were used to quantify the stylistic variation between document types. These included computing character and word counts, for example, a typical quote averaged around 13 words and 80 characters, while SOPs ranged from 500 to 1000 words, visualizing text distributions using histograms and boxplots, and applying standard readability formulas such as Flesch Reading Ease, Flesch-Kincaid Grade, and Gunning Fog Index to get a sense of linguistic complexity across types. As an example, SOPs yielded an average Flesch-Kincaid Grade Level of 11.8, indicating a more academic tone, whereas quotes tended to score higher on the Flesch Reading Ease scale, reflecting simpler and more conversational language. In Quotes and Letters, descriptive adjectives like mad, precious, kind, and strong appear frequently. SOPs and LORs may have more formal adjectives like analytical, dedicated, experienced, and proficient, aligning with their evaluative nature.

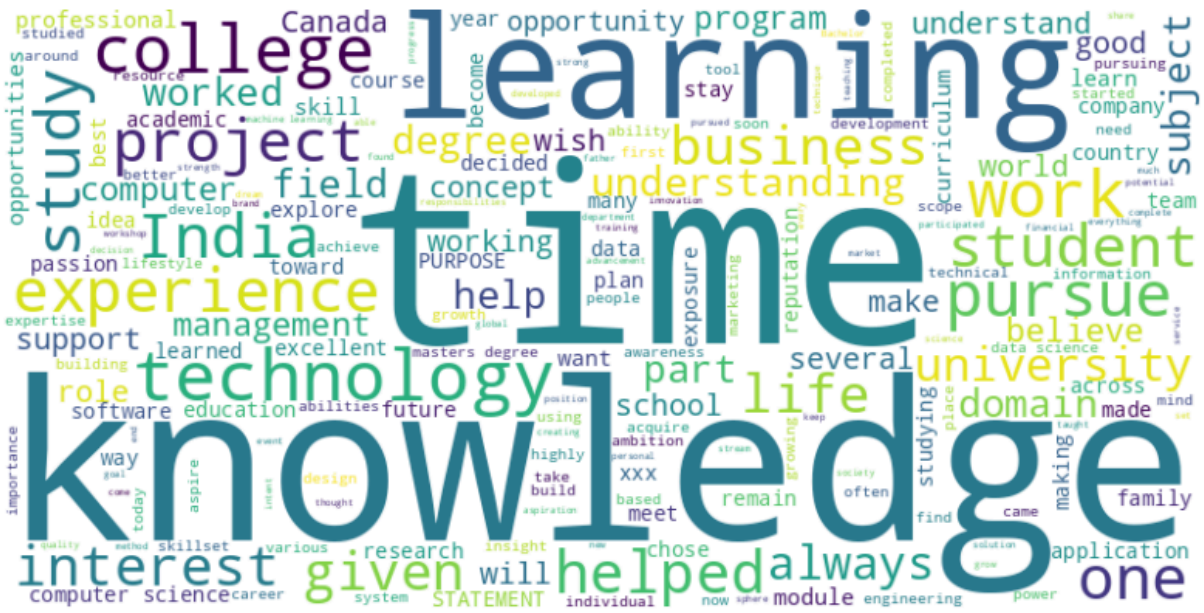
Thematic analysis of the dataset revealed a clear and meaningful divergence in word usage between quotes and SOPs, reinforcing the need to treat them as separate text generation tasks. In the word cloud derived from quotes, emotionally charged and introspective terms dominated, words like *love, heart, soul, feel, make, want,* and *never* were highly frequent. These reflect the

personal, expressive nature of quotes, which often aim to evoke sentiment, reflect inner thoughts, or deliver motivation and reflection in a concise form.

Figure 2: Word Cloud- Quotes

In contrast, the SOP word cloud showcased a completely different vocabulary set, centered around academic, professional, and goal-oriented language. Dominant terms included *knowledge, learning, experience, technology, project, university* and *time.* These words are more formal and content driven, characteristic of structured academic narratives that aim to establish credibility, motivation, and future aspirations.

Figure 3: Word Cloud- SOPs



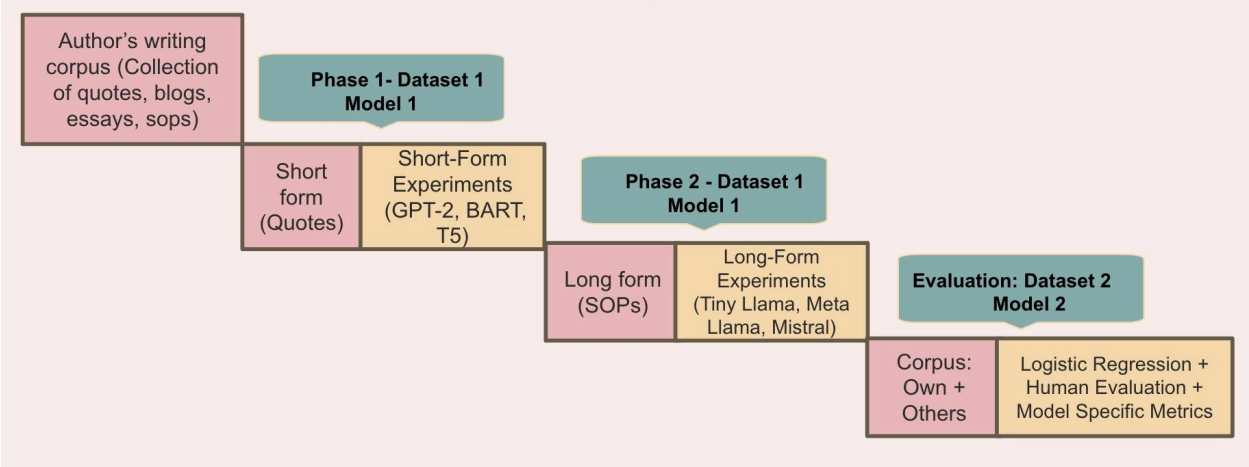
This divide highlights the importance of building separate pipelines and conditioning strategies for each text type. While short form text (quote) generation demands attention to tone, metaphor, and emotional cadence, long form text (SOP) generation requires structural coherence, domain relevance, and progression of thought. The observed lexical separation provided empirical support for splitting the corpus and tailoring fine-tuning and prompt strategies accordingly.

In the early stages of experimentation, we observed that mixing short-form and long-form texts in the same dataset led to frequent hallucinations, tone drift, and loss of structural coherence during generation. When models were trained on a combined dataset, they frequently overgeneralized, producing outputs that lacked specificity e.g., motivational-style text where structural exposition was needed. This prompted us to separate short-form and long-form pipelines, resulting in more efficient fine-tuning and stable convergence. With homogeneous training data, we achieved more consistent evaluations using clarity, coherence, and style metrics. The split also allowed us to build a modular, extensible framework where SOPs, quotes, blogs, and academic letters could each be handled by dedicated generation and evaluation logic. This approach laid a strong foundation for future expansion to additional content types, while minimizing hallucinations and preserving the core stylistic essence of each form.

This strategic separation of text types gave rise to Model 1 Phase 1 (Short-Form text generation) and Model 1 Phase 2 (Long-Form text generation), forming the backbone of our dual-pipeline architecture. The dataset was split into 499 short-form entries, including quotes and personal

letters known for their brevity and emotional expressiveness, and 560 long-form entries such as SOPs, LORs, blogs, and visa essays, which are more structured, purpose-driven, and content-heavy. This near-equal division (approximately 47% short-form and 53% long-form) ensured balanced representation during both model training and evaluation. It allowed us to fine-tune architectures with focused datasets that matched their intended output form, enabling stylistic fidelity in quotes and structural coherence in SOPs. By creating dedicated paths for generation and inference, this split not only addressed hallucination and tone inconsistency issues but also set the stage for a stable and extensible pipeline moving forward.

Figure 4: Model Architecture and Approach



Model 1: Overall Architecture and Approach:

Our methodology is divided into two core components: short-form and long-form text generation, each tailored to capture the author's unique style and emotional tone.

Phase-1: For short-form generation, we experimented with models like GPT, BART, and T5, applying few-shot prompting, metadata conditioning, and contrastive scoring to replicate stylistic nuance in quote-style outputs.

Phase-2: For long-form generation, we focused on fine-tuning instruction-based LLMs (TinyLlama, Meta Llama, Mistral) on curated Statement of Purpose (SOP) datasets, employing prompt engineering, reasoning and Parameter-Efficient Fine-Tuning (LoRA) to adapt outputs while preserving fluency and structure. Across both formats, evaluation incorporated stylistic alignment, semantic fidelity, and authorship attribution to assess model performance.

Model 1- Phase 1: Short-Form Generation Model Variants

To capture the author's tone and structure in short-form content (quotes), we designed four progressive models, each increasing in stylistic alignment and control. The experiment involved evaluating these models not only on fluency and relevance but also on how well they matched the author's writing style, tone, and phrasing. These ranged from basic fine-tuning with GPT-2 to advanced metadata-conditioned generation using T5.

Table 1: Short form Generation Models: Architecture and Setup Overview

Model Variant	Description	Core Setup	Architecture
Variant 1 (v1)	GPT-2 with fixed prompt, no personalization	Static Prompt	Transformer-based causal LM (decoder-only)
Variant 2 (v2)	BART with STYLEMC energy-based scoring	Post-eval Reranking	Encoder-decoder
Variant 3 (v3)	T5 Few-Shot Declarative Prompting	Style examples + prompt	Encoder-decoder
Variant 4 (v4)	T5 Metadata-Aware Generation	Metadata Conditioning	Encoder-decoder

Data Preparation and Model Framework:

The training dataset consisted of over 1,500 quotes authored by one team member and sourced from image-based Instagram archives and local writing repositories. Each quote was annotated with metadata such as tone (e.g., reflective, wise, empathetic), text type (quote), and stylistic markers (e.g., punctuation, sentence length) and relevant format of the data was used based on the processing requirement for each model.

During preprocessing, the quote dataset underwent multiple steps to extract stylistic features relevant for short-form generation. Initially, each quote was analyzed for its word and character count to capture length-based structural patterns. Common stop words were removed using the standard English stopword list to emphasize meaningful content. Tokenization was performed using CountVectorizer, which also enabled the extraction of frequent bigrams and trigrams.

These n-grams were later used to identify recurring stylistic phrases and patterns, helping define the author's unique writing fingerprint and informing downstream modeling decisions.

In the first phase of our project, we targeted short-form text generation, focusing on emulating the author's style through emotionally resonant quotes. Our goal was to preserve tone, emotional weight, and stylistic nuances while maintaining coherence and originality. We experimented with four variants of short-form text generators to replicate the author's quote style, tone, and intent. Initial experimentation revealed that general prompting or zero-shot approaches with base LLMs (e.g., GPT-3.5) resulted in generic, fluently worded outputs that lacked stylistic fidelity or emotional subtlety. This emphasized the need for a fine-grained conditioning strategy in general, the progression is expressed more clearly below.

Our baseline (variant 1) used GPT (125 – 355 million parameters) with static prompting and no metadata, which resulted in fluent but generic outputs. To improve personalization, we implemented a STYLEMC-inspired scoring strategy (variant 2) using BART (140 million parameters), which evaluated outputs based on a weighted energy score combining fluency, semantic similarity, and style similarity. Each candidate output was scored and re-ranked using the STYLEMC energy function:

$$\text{Energy} = 0.3 \times \text{Fluency} + 0.3 \times \text{SemanticSim} + 0.4 \times \text{StyleSim}$$

ensuring outputs reflected both the author's voice and contextual meaning.

For variant 3, we applied a few-shot declarative prompting technique using T5 (220 – 770 million parameters), injecting example quotes to guide tone and structure.

Our best-performing model (variant 4) used T5 (770 million parameters) fine-tuned with annotated metadata such as tone (e.g., introspective), text type (quote), and structure (one-liner). This metadata-aware conditioning improved generation quality by preserving emotional nuance, stylistic consistency, and quote formatting.

Algorithms:

1. **v1 (GPT-2 Fine-Tuning)**

Training

1: D ← load and clean dataset of quotes
2: For each row in D: format as prompt + completion
3: D_train, D_val ← split D into 80% train, 20% val
4: T ← load GPT-2 tokenizer; T.pad_token ← T.eos_token
5: Define tokenize(batch):

```
 6:      inputs ← T(batch.prompt + batch.completion, padding="max_length", truncation=True, max_length=128)
 7:    labels ← shift tokens for autoregressive loss
 8:    Return {"input_ids", "labels", "attention_mask"}
 9: tokenized_train ← map tokenize over D_train
 10: tokenized_val ← map tokenize over D_val
 11: M ← load GPT-2 base model (125M–355M)
 12: optimizer ← AdamW(M.parameters, lr=1e-5)
 13: For epoch in range(1, E+1):
 14:    For batch in train_loader:
 15:       output ← M(**batch)
 16:       loss ← output.loss
 17:       loss.backward(); optimizer.step(); optimizer.zero_grad()
 18: Save model and tokenizer to disk
```

Inference

```
1: prompt_text ← seed prompt or custom instruction
 2: inputs ← T(prompt_text, return_tensors="pt").to(device)
 3: output ← M.generate(**inputs, max_length=50, temperature=0.7, top_p=0.9)
 4: decoded ← T.decode(output[0], skip_special_tokens=True).strip()
 5: Display or log decoded result
```

2. v2 (BART + STYLEMC Reranking)

Inference

```
1: prompts ← load or input prompt list
 2: style_vector ← average of Sentence-BERT embeddings of user's corpus
 3: For each prompt P in prompts:
 4:    inputs ← BART tokenizer(P, return_tensors="pt")
 5:    outputs ← generate 5 candidates using beam search
 6:    candidates ← decode outputs
 7:    prompt_emb ← Sentence-BERT(P)
 8:    For each candidate C:
 9:       fluency ← log-probability score from BART decoder
 10:       sem_sim ← cosine(prompt_emb, Sentence-BERT(C))
 11:       style_sim ← cosine(style_vector, Sentence-BERT(C))
 12:       energy ← 0.3 × fluency + 0.3 × sem_sim + 0.4 × style_sim
```

13: Select candidate with max(energy) as final output
14: Store final output in result list

3. v3 (T5 Few-Shot Prompting)

Prompt Construction

1: Q ← author-written quotes corpus
2: E ← select 3 stylistically consistent quotes from Q
3: T ← target fragment (e.g., “You are not lost.”)
4: prompt ← concatenate(E[0], E[1], E[2], "Now rewrite: " + T)

Inference

5: tokenizer, model ← load T5 (base or large)
6: input_ids ← tokenizer(prompt, return_tensors="pt")
7: output ← model.generate(
input_ids,
max_length=50,
temperature=0.6,
top_p=0.95,
num_return_sequences=1,
early_stopping=True
)
8: decoded ← tokenizer.decode(output[0], skip_special_tokens=True)
9: Display decoded output

4. v4 (T5 with Metadata Conditioning)

Prompt Construction

1: meta_tags ← define metadata (e.g., “Tone: introspective | Type: quote”)
2: T ← seed phrase or topic (e.g., “You are not lost.”)
3: prompt ← meta_tags + “\nGenerate a personalized quote in the user's voice:\n” + T

Inference

4: tokenizer, model ← load T5-large
5: input_ids ← tokenizer(prompt, return_tensors="pt")
6: output ← model.generate(

```
        input_ids,
        max_length=50,
        temperature=0.6,
        top_p=0.95,
        do_sample=True,
        early_stopping=True
)
 7: decoded ← tokenizer.decode(output[0], skip_special_tokens=True)
 8: Display or log decoded output
```

Model 1- Phase 2: Long form Generation Model Variants

In the second part of our project, we focused on long-form text generation based on the author's writing style. However, we specifically chose to work with Statements of Purpose (SOPs*)* rather than multiple types like blogs, essays or letters. This decision was driven by several important considerations from our early fine-tuning experiments, where a mixed dataset comprising SOPs, blogs, and academic letters caused significant hallucination and incoherence in model outputs. The model struggled to generalize across formats and frequently overgeneralized or produced repetitive and implausible content. By focusing only on SOPs, we achieved a more stable training regime and faster convergence. Evaluation also became more meaningful due to the homogeneous nature of the dataset, allowing us to assess coherence, tone, and structure reliably. Furthermore, SOPs provided a structured domain through which we could test our training and inference strategies, laying a solid foundation for future generalization to other formats such as blogs or academic recommendation letters.

Data Preparation and Model Framework

One of the most key decisions was selecting a language model that could be effectively fine-tuned on the available hardware. We initially adopted TinyLlama (1.1B), a lightweight yet capable model well-suited for experimentation. This enabled fast iterations to validate training scripts, tokenizer alignment, and the effectiveness of LoRA (Low-Rank Adaptation) strategies. Once confidence was built in the pipeline, we scaled up to Meta Llama 3B and Mistral 7B models, which offered improved contextual depth and better generation fluency with author's style. This progressive scaling avoided premature failure, allowed reuse of code, and established a controlled comparison of performance across model sizes.

Table 2: Long form Generation Models: Architecture and Setup Overview

Model Variant	Description	Core Setup	Architecture
Variant 1 (v1)	Tiny Llama 1.1B	Plain SOPs PEFT with LoRa	Transformer-based causal LM(decoder-only) Base:TinyLlama/TinyLlama-1.1B-Chat-v1.0
Variant 2 (v2)	Meta Llama 3B	Prompt Augmented SOPs PEFT with LoRa FineTuning + Few Shots Prompt at inference	Transformer-based causal LM (decoder-only) Base:meta-Llama/Llama-3.2-3B
Variant 3 (v3)	Mistral 7B: Model A	Plain SOPs (no reasoning) PEFT with LoRA Fine tuning	Transformer-based causal LM (decoder-only) Base:mistralai/Mistral-7B-Instruct-v0.2
	Mistral 7B: Model B	Prompt Augmented SOPs + reasoning block PEFT with QLoRA	
	Mistral 7B: Model C	Instruction-tuned SOPs PEFT with LoRA	

For the TinyLlama model the source dataset was a cleaned Excel spreadsheet containing a single column, with each entry representing a full-length SOP. The dataset was split into training and validation subsets using an 80:20 ratio. Once the pipeline proved stable and effective, we transitioned to the Meta Llama 3B model. This transition was expected to better handle long-form SOPs, enhancing contextual depth, fluent generation and stronger retention of authorial style. Llama 3B's ability to process longer sequences (up to 2048 tokens) and its compatibility with advanced memory optimization techniques like AMP (mixed precision) and gradient accumulation makes it a better fit. For MetaLlama model prompts were added to the dataset using the following structure,

"Prompt: Generate a long-form academic Statement of Purpose:

Response:

STATEMENT OF PURPOSE ...”

Prompt-based dataset was more effective than simple prefixing with tags like [SOP], as it provided clear task intent, aligned better with how users in general interact. It also helped reduce hallucinations, improve style retention and enhance generalization to varied prompt formulations during inference. An 80:20 split was done for training and validation sets. To make model training viable on modest hardware, the project utilized PEFT through LoRA for all three models. This method inserts low-rank matrices into attention layers, allowing the model to learn new tasks by tuning only a small subset of parameters. For inference on TinyLlama and Meta Llama model, the generation script allows users to input any custom SOP prompt and observe how the model generates text in response. The script logged each prompt and output, supported on-demand saving, and enabled qualitative analysis of different prompt styles.

To create a fine-tuning dataset for Mistral 7B tailored for instruction-based learning, we implemented a custom data preparation script that utilized the mistralai/Mistral-7B-Instruct-v0.2 model from Hugging Face's transformers library. The script processed a collection of SOPs provided in a CSV file. For each SOP, it extracted the initial lines and prompted the language model to generate a one-line instructional command in the format “Write me an SOP for...”, encapsulating the applicant’s intended degree and specialization. These instructions were designed to simulate real-world use cases of LLM prompting. Each generated instruction, along with its associated SOP text, was saved as a structured YAML entry. To ensure fault tolerance during processing, partial progress was saved every 25 iterations. Once the full dataset was created, it was converted into JSON Lines (.jsonl) format, making it suitable for downstream fine-tuning workflows. The resulting dataset was divided into three parts: 80% for training, 10% for validation, and a small held-out set of 10 samples for evaluation.

For Mistral three different prompt formatting strategies were adopted. Each of these formats was used to fine-tune a separate version of the Mistral model, enabling us to analyze how varying levels of contextual conditioning affected the model’s stylistic fidelity and content structure.The prompt format is given as follows

Mistral Model A: ### Instruction\n...\n### Response

Mistral Model B: <s>[INST] ... [/INST] + [SOP] + [REASONING]

Mistral Model C: <s>[INST] ... [/INST] followed by [SOP]

Algorithm

a) TinyLlama 1.1B

Training:

```
1:  D ← load and clean dataset
2:  For each row in D: prepend “[SOP] ” to cleaned_text
3:  D_train, D_val ← split D into 80% train and 20% val
4:  T.pad_token ← T.eos_token
5:  Define tokenize(batch):
6:      tokenized ← T(batch.text, padding="max_length", truncation=True, max_length=L)
7:      For each seq in tokenized.input_ids:
8:      mask_last_100 ← [-100] * (L - 100) + seq[-100:]
9:      Replace pad_token_id with -100 in mask_last_100
10:     Append to labels
11:     return {"input_ids", "labels", "attention_mask"}
12: tokenized_train ← map tokenize over D_train
13: tokenized_val ← map tokenize over D_val
14: Set format of both datasets to torch tensors
15: M ← load base model with float32 and device_map="auto"
16: M ← apply LoRA(M, C) with r=8, alpha=32, target=["q_proj", "v_proj"], dropout=0.05

17: optimizer ← AdamW(M.parameters, lr=1e-5)
18: M.train(); M.to(device)
19: For epoch in range(1, E+1):
20:     total_loss ← 0
21:     steps ← 0
22:     For batch in train_loader:
23:     Move batch to device
24:     output ← M(**batch)
25:     If output.loss is NaN: continue
26:     output.loss.backward()
27:     optimizer.step(); optimizer.zero_grad()
28:     total_loss += output.loss.item(); steps += 1
29:     Print average loss: total_loss / steps
30: Save M and T to output directory
```

Inference:

```
1:  prompt_text ← concatenate(F, "\n\n", P, "\n\nStatement of Purpose:\n")
2:  inputs ← T(prompt_text, return_tensors="pt").to(device)
3:  prompt_len ← length(inputs.input_ids)

4:  output ← M.generate(
        **inputs,
        max_new_tokens=1500,
```

```
        temperature=0.4,
        top_p=0.8,
        top_k=40,
        repetition_penalty=1.2,
        no_repeat_ngram_size=3,
        pad_token_id = T.eos_token_id or T.pad_token_id,
        eos_token_id = T.eos_token_id or T.pad_token_id
    )
5:  generated_tokens ← output[0][prompt_len:]
6:  decoded ← T.decode(generated_tokens, skip_special_tokens=True).strip()
7:  cleaned ← remove_closing_phrases(decoded)   // e.g., "Thank you", "Best regards", etc.
8:  word_count ← count_words(cleaned)
9:  Display cleaned and word_count
10: Ask user: save this SOP? (y/n)
11: If yes:
12:     filename ← "sop_" + topic + timestamp + ".txt"
13:     Save prompt_text, cleaned SOP, word_count, timestamp to file
14: Else:
15:     Discard and log decision
```

b) MetaLlama 3.2 3B

Training:

```
Require: Dtrain, Dval — Training and validation datasets
Require: M — Base model (Llama 3.2B)
Require: T — Tokenizer with EOS as pad_token
Require: Lconfig — LoRA config (r=8, alpha=16, dropout=0.1)
Require: MAX_LEN = 2048, BATCH_SIZE = 1, EPOCHS = 3, LR = 5e-7,
GRAD_ACCUM_STEPS = 4

1: Initialize M with pre-trained weights
2: Apply LoRA to M using Lconfig
3: Enable gradient checkpointing and input grads for M
4: Tokenize Dtrain and Dval with T and MAX_LEN
5: Convert padding tokens in labels to -100 for loss masking
6: optimizer ← AdamW(M.parameters(), lr=LR)
7: scaler ← GradScaler()
8: for epoch = 1 to EPOCHS do
9:     for step, batch in Dtrain do
10:    batch ← clamp input_ids to vocab size
11:    loss ← M.forward(batch) with AMP
12:    if loss is NaN or Inf then continue
13:    scaler.scale(loss).backward()
14:    Clip gradients
```

```
15:     if (step + 1) mod GRAD_ACCUM_STEPS == 0 then
16:         scaler.step(optimizer)
17:         scaler.update()
18:         optimizer.zero_grad()
19:     end if
20:     end for
21:     Evaluate model on Dval and log validation loss
22: end for
23: Save M and tokenizer to output directory
```

Inference:

```
Require: M — Fine-tuned model with LoRA adapter
Require: T — Tokenizer
Require: F — Few-shot examples file
Require: P — User-provided custom SOP prompt

1: prompt_text ← concatenate F and P into structured prompt
2: inputs ← T(prompt_text)
3: prompt_len ← length(inputs.input_ids)
4: output ← M.generate(inputs,
        max_new_tokens=1500,
        temperature=0.4,
        top_p=0.8,
        top_k=40,
        repetition_penalty=1.2,
        no_repeat_ngram_size=3,
        pad_token_id=eos_token_id)
5: decoded ← T.decode(output[prompt_len:])
6: Display decoded SOP and word count
```

c) Mistral 7B

Training:

Dtrain, Dval - Training and validation datasets
M - Base model (Mistral-7B-Instruct-v0.2)
T - Tokenizer with eos_token as pad_token
Lconfig - LoRA config (r=8, alpha=16, dropout=0.1, bias=None, task_type='CAUSAL_LM')
MAX_LEN = 2048, BATCH_SIZE = 1, EPOCHS = 3, LR = 2e-4, GRAD_ACCUM_STEPS = 4

```
1: Initialize M with pre-trained Mistral-7B weights
2: Apply LoRA to M using Lconfig via `peft`
3: Enable gradient checkpointing and input gradients on M
4: Tokenize Dtrain and Dval using T with truncation to MAX_LEN
5: Replace padding token labels with -100 for loss masking
```

6: optimizer ← AdamW(M.parameters(), lr=LR)
7: scaler ← GradScaler()
8: for epoch = 1 to EPOCHS do
9: for step, batch in Dtrain do
10: batch ← clamp input_ids to vocab size
11: loss ← M.forward(batch) using AMP (autocast)
12: if loss is NaN or Inf then continue
13: scaler.scale(loss).backward()
14: Clip gradients to prevent exploding values
15: if (step + 1) mod GRAD_ACCUM_STEPS == 0 then
16: scaler.step(optimizer)
17: scaler.update()
18: optimizer.zero_grad()
19: end if
20: end for
21: Evaluate on Dval and log validation loss
22: end for
23: Save LoRA adapter and tokenizer to output directory

Inference:

M - Fine-tuned Mistral-7B model with LoRA adapter
T - Tokenizer
F - Few-shot examples file
P - User-provided SOP prompt

1: prompt_text ← concatenate F and P into structured prompt format
2: inputs ← T(prompt_text, return_tensors="pt", truncation=True)
3: prompt_len ← length(inputs.input_ids)
4: output ← M.generate(inputs,
 max_new_tokens=1000,
 temperature=0.4,
 top_p=0.85,
 top_k=40,
 repetition_penalty=1.15,
 no_repeat_ngram_size=3,
 pad_token_id=eos_token_id)
5: decoded ← T.decode(output[0][prompt_len:], skip_special_tokens=True)
6: Display decoded SOP and word count

Model 2 – Authorship Attribution Classifier (Logistic Regression)

Exploratory Data Analysis of Dataset2 (Team member's Original Writings & Other Authors Writings):

The dataset used for Model 2 consisted of short-form and long-form texts, each labeled with an author ID. The initial data cleaning phase involved dropping rows with missing values in the cleaned_text column. Author IDs were converted to string type for compatibility, and entries without valid author identification were filtered out.

To ensure balanced class representation across authors for training the classification model, the dataset was downsampled. The minimum number of samples per author was computed, and all author groups were resampled to this minimum. Visualization revealed that before balancing, the dataset was skewed, some authors had significantly more data than others. After balancing, each author had an equal number of samples, ensuring fair training and evaluation.

This step ensured that the logistic regression classifier (Model 2) was trained on a clean, class-balanced, and label-consistent dataset, minimizing bias during authorship prediction. Furthermore, the division between short-form and long-form documents was intentionally made to ensure that stylistically relevant text snippets could be compared against similar types of writing. This allowed for fair and context-appropriate evaluation when distinguishing the user's work from that of other authors.

Methodology:

To evaluate the stylistic alignment of generated texts with the author's personal writing, we developed Model 2, a logistic regression-based authorship attribution classifier. This model was designed to return a probability indicating how likely a given text sample was written by the user, based on lexical, syntactic, and stylistic cues. The primary objective of Model 2 was not to function as a standalone author classifier, but rather as an evaluation tool for verifying whether outputs from short-form and long-form generation models could successfully emulate the user's writing style. This model was essentially built as an internal confidence check, a form of self-assurance that the outputs generated bore sufficient resemblance to the user's real writing before further model iterations were pursued.

The dataset used for training this model was constructed by merging samples from multiple publicly available authors with the user's personal dataset. First, the cleaned authorship dataset (cleaned_authorship_dataset.csv) was preprocessed by dropping any rows with missing cleaned_text values and excluding authors with placeholder IDs, such as "123456.0". To ensure consistency and comparability, we filtered the dataset to retain only short-form entries defined as texts containing no more than 30 words and 400 characters for short form and for long form texts

containing more than 300 words and 1000 characters. To avoid class imbalance, the number of samples per non-user author was capped at 350. The user's data were extracted from a separate dataset and underwent the same cleaning and filtering pipeline. These entries were labeled with the class "user" and appended to the final combined dataset.

The text preprocessing workflow included converting all content to lowercase, removing stop words using NLTK's standard English list, stripping punctuation, and applying stemming using the PorterStemmer algorithm. These transformations resulted in a final feature column that was used for training. Lexical features were extracted using the TF-IDF vectorizer, which likely captured unigram to trigram phrase structures and emphasized the frequency of meaningful terms. These TF-IDF vectors served as input to a logistic regression classifier trained to differentiate between the user's texts and those written by other authors. The model did not use any deep learning or fine-tuning mechanisms; instead, it relied on classical NLP and linear classification to learn stylistic boundaries between writing types.

IV. Experiments:

Model 1 - Phase 1: Short-form text generation

The experimental phase focused on evaluating how effectively each short-form model variant captured and reproduced the author's unique style, tone, and structure. Each model was evaluated not only on syntactic fluency but also on semantic alignment with the prompt and stylistic fidelity to the author's known writing. Evaluation used both quantitative metrics (cosine similarity, classifier-based authorship validation, fluency scores) and qualitative assessment (human review for coherence, emotional depth, and originality).

For all models, a consistent set of prompts was used. Prompts were drawn from the user's own unpublished quote fragments or rewritten sentence starters, selected to be emotionally expressive, stylistically challenging, or structurally unique. The dataset of approximately 1,500 quotes was split into 80% for training and development, primarily for GPT-2 fine-tuning and prompt construction and 20% for testing model outputs. For consistency, all inference-only models (v2–v4) were evaluated using the same test prompts.

Model 2, used for authorship classification, was trained on a balanced dataset of user and non-user quotes, using TF-IDF features and logistic regression to verify whether generated text matched the author's stylistic signature.

Model Variants and Training Strategies:

1. Model v1 – Fine-Tuning GPT 2 (Baseline):

Reference: Sawicki et al. (2023) on GPT-3 fine-tuning for poetry

The baseline model, GPT-2 (125 – 355 million parameters) was fine-tuned on a curated dataset of author-written quotes formatted as 'prompt + completion' pairs using Hugging Face's Trainer API. Each quote was preprocessed and optionally annotated with tonal cues, though these cues were not embedded into the prompt in this version. Tokenization was handled using the GPT-2 tokenizer, with the padding token explicitly aligned to the end-of-sequence (EOS) token to maintain uniformity across batches. Input length was restricted to 128 tokens to minimize memory issues and overfitting on shorter text. A grid search was performed over learning rates (5e-5, 2e-5, 1e-5), batch sizes (2, 4), weight decay (0.1,0.01) and number of epochs (3–5). Cross Entropy Loss was monitored using Hugging Face's built-in TrainerCallback. Among the trials, the best-performing configuration was a Learning Rate of 1e-5, Batch size of 2, 5 Epochs and weight decay of 0.01. These settings yielded the most stable loss convergence, while mitigating gradient noise and avoiding mode collapse.

Output Evaluation and Limitations: The outputs from the fine-tuned GPT-2 model were syntactically fluent and grammatically sound, but they lacked depth in stylistic and emotional expression. A recurring limitation was the absence of emotional specificity—while the quotes were coherent, they often failed to capture the introspective and nuanced tone of the original author. The model frequently overgeneralized, generating repetitive phrases such as "keep going" or "you are strong," which led to outputs that felt generic and lacked individuality. Additionally, the structure of the quotes was largely uniform, favoring short and symmetrical sentences while avoiding rhetorical devices like metaphor, repetition, or ellipsis. Most critically, the model lacked any form of style conditioning, as the prompts did not include metadata related to tone or intent, leaving the model unable to adapt its output to reflect distinct stylistic cues.

Impact on Subsequent Model Design (v2 and Beyond): The above mentioned limitations provided clear motivation for the development of Model v2, which introduced post-generation style scoring to enhance personalization and stylistic fidelity.

Implemented Algorithm:

Tokenizer & Data:

```
tokenizer = GPT2Tokenizer.from_pretrained("gpt2")
tokenizer.pad_token = tokenizer.eos_token
```

```
def tokenize(example):
    return tokenizer(example["prompt"] + example["completion"],
                     truncation=True, padding="max_length", max_length=128)
```

Pseudocode:

```
for epoch in range(epochs):
    for batch in dataloader:
        outputs = model(**batch)
        loss = outputs.loss
        loss.backward()
        optimizer.step()
        optimizer.zero_grad()
```

2. Model v2 – BART with STYLEMC-Inspired Reranking:

Reference: Khan et al. (2023), STYLEMC architecture for stylometric evaluation

For Model v2, the data used for generation and scoring was preprocessed to retain only clean, short-form quotes authored by the target user. Each quote was tokenized and normalized, with punctuation preserved to maintain stylistic integrity. Unlike Model v1, this version did not involve supervised training or gradient updates; instead, it used pretrained BART (base, 140M parameters) purely in inference mode. For each input prompt, the model generated 5 candidate outputs using beam search decoding. These candidates were then evaluated using an energy-based reranking strategy inspired by STYLEMC. The reranking formula combined three components: fluency (log-likelihood from the BART decoder), semantic similarity (cosine similarity between prompt and output using Sentence-BERT embeddings), and style similarity (cosine similarity between the generated output and a style vector derived from the author's corpus).

Each component was weighted as follows: 0.3 for fluency, 0.3 for semantic similarity, and 0.4 for style similarity. Although hyperparameters such as temperature and beam width were explored, the most reliable performance was achieved with default beam search (num_return_sequences = 5) and a max length of 50 tokens. Since no fine-tuning occurred, there was no learning rate, loss function, or epoch schedule involved. Instead, reranking based on the energy score was the key mechanism that filtered outputs, improving alignment with the author’s original tone and structure.

The fluency is computed as the average log-probability of the output sequence under BART. Semantic Similarity is the Cosine similarity between prompt and output using Sentence-BERT and Style Similarity is the Cosine similarity between output and a style vector generated from the author's quote corpus.

Final Energy Score for each candidate: Energy = 0.3 × Fluency + 0.3 × SemanticSim + 0.4 × StyleSim

Verdict: The candidate with the highest energy score was selected as the final output.

Output Evaluation and Limitations: Model v2 produced noticeable improvements over the GPT-2 baseline in terms of stylistic alignment, emotional tone, and rhetorical diversity. By leveraging the STYLEMC-inspired scoring function, the system was able to rerank multiple candidate outputs and select the one most closely aligned with the author's stylistic vector. Outputs demonstrated enhanced coherence and emotional resonance, with reduced overuse of generic motivational templates. Compared to Model v1, this approach produced quotes that better mimicked the author's natural cadence and metaphorical phrasing. However, since BART itself was not fine-tuned on the author's writing, the model still relied heavily on the quality of prompt design and the reranking function to achieve stylistic fidelity. The system's dependence on post-generation filtering also introduced computational overhead, as multiple candidates had to be generated and scored for each prompt. Moreover, in cases where all generated outputs were stylistically weak, the reranking mechanism could only make relative improvements, not absolute corrections. Overall, while Model v2 did not modify the generative model’s parameters, it demonstrated that effective post-processing strategies can significantly enhance the perceived quality and personalization of generated short-form text.

Impact on Subsequent Model Design (v3 and Beyond): The success and shortcomings of Model v2 directly informed the development of Model v3. While reranking with STYLEMC improved stylistic alignment without requiring fine-tuning, it revealed key limitations: the system was still reactive rather than generative in its control over style, and it lacked the ability to internalize authorial tone. This highlighted the need for a model that could better integrate stylistic cues during generation itself. Consequently, Model v3 shifted toward few-shot declarative prompting using T5, enabling stylistic control through carefully curated in-context examples. The experience with Model v2 also emphasized the importance of flexible, low-resource methods that could guide outputs without retraining. The energy-based scoring approach served as an evaluation benchmark for future models, while the limitations in candidate diversity and selection inspired more deliberate prompt engineering strategies. Ultimately, v2 validated that personalization could be achieved through external mechanisms, but laid the groundwork for more intrinsic, example-driven style control implemented in v3 and metadata-aware conditioning in v4.

Implemented Algorithm:

Tokenizer & Data:

```
from transformers import BartTokenizer, BartForConditionalGeneration
from sentence_transformers import SentenceTransformer
from scipy.spatial.distance import cosine

# Load tokenizer and model
tokenizer = BartTokenizer.from_pretrained("facebook/bart-base")
model = BartForConditionalGeneration.from_pretrained("facebook/bart-base")

# Load Sentence-BERT for embeddings
embedding_model = SentenceTransformer('all-MiniLM-L6-v2')
```

Pseudocode:

```
for prompt in prompts:
    # Step 1: Generate multiple candidate outputs using BART
    inputs = tokenizer(prompt, return_tensors="pt")
    outputs = model.generate(
        **inputs,
        num_return_sequences=5,
        num_beams=5,
        max_length=50,
        early_stopping=True
    )
    candidates = [tokenizer.decode(output, skip_special_tokens=True) for output in outputs]

    # Step 2: Score each candidate based on fluency, semantic, and style similarity
    scores = []
    prompt_emb = embedding_model.encode(prompt)

    for candidate in candidates:
        # Fluency (approximated via output log probability, optionally skipped)
        fluency = model(**tokenizer(candidate, return_tensors="pt")).logits[0].mean().item()

        # Semantic similarity between prompt and output
        candidate_emb = embedding_model.encode(candidate)
        sem_sim = 1 - cosine(prompt_emb, candidate_emb)

        # Style similarity to author's style vector
```

```
        style_sim = 1 - cosine(candidate_emb, author_style_vector)

        # Compute energy score
        energy_score = 0.3 * fluency + 0.3 * sem_sim + 0.4 * style_sim
        scores.append(energy_score)

    # Step 3: Select the candidate with the highest score
    best_output = candidates[np.argmax(scores)]
    final_outputs.append(best_output)
```

3. Model v3 – T5 with Few-Shot Declarative Prompting

Reference: Riley et al. (2020), TextSETTR-style prompting

Model v3 introduced a significant shift from parameter-based training to prompt-based stylistic control through the use of few-shot learning. Leveraging the T5 architecture (Base: 220 million parameters, Large: 770 million parameters), this model aimed to emulate the author's writing style by presenting it with three example quotes, followed by a new input sentence to be rewritten in the same tone. These examples acted as in-context demonstrations of structure, emotion, and voice, guiding the model's output without modifying its underlying weights. Prompts were carefully engineered using mid-length quotes selected from the author's corpus to ensure consistency in tone and rhythm. The model was run in inference-only mode, using Hugging Face's generation pipeline with parameters optimized for stylistic variability and fluency. This declarative prompting strategy allowed for style transfer without the need for computationally expensive fine-tuning. The data used in this process was carefully preprocessed: quotes were cleaned for formatting, punctuation was normalized, and entries were filtered to retain mid-length quotes between 8 and 25 words to ensure structural consistency. These quotes were then formatted into declarative prompts such as: "Even silence tells a story." "You are allowed to begin again." "Softness is not weakness." Now rewrite: "You are not lost." This setup allowed the model to learn stylistic transformations by analogy.

Inference was conducted using Hugging Face's pipeline("text2text-generation") with no model-level training. The best-performing configuration used T5-Large with the following decoding parameters: a temperature of 0.6, top-p set to 0.95, and a maximum output length of 50 tokens. The model was asked to generate only one response per prompt. Since there was no training phase, there was no associated loss function. However, output quality was evaluated using sentence-level semantic similarity (via Sentence-BERT), stylistic verification using a logistic regression classifier (Model 2), and human-inspected fluency and tone alignment.

Output Evaluation and Limitations: The outputs generated by Model v3 demonstrated a notable improvement in stylistic resonance, especially in capturing abstract, introspective tones and layered emotional cues. Compared to earlier models, these outputs featured more expressive phrasing and metaphoric structure, and often exhibited sentence constructions similar to the author's authentic writing. However, the model's effectiveness was highly dependent on the quality and specificity of the prompt. Prompts with strong example quotes led to stylistically rich outputs, while vague or overly general prompts caused the model to revert to safe, generic responses. Additionally, without an external scoring or reranking mechanism, the system lacked a safeguard against occasional style drift or semantic hallucination. Although v3 succeeded in introducing implicit stylistic control, it had no way to interpret or enforce explicit attributes such as tone or intent beyond what was demonstrated in the examples.

Impact on Subsequent Model Design (v4 and Beyond): The experience with Model v3 underscored the power and fragility of example-based prompting. While few-shot inputs allowed the model to produce stylistically convincing outputs without any training, they also revealed the limitations of relying solely on implicit style cues. This prompted a transition in Model v4 to a more structured metadata-based prompting strategy. Rather than expecting the model to infer tone from examples, Model v4 provided explicit tags (e.g., tone: "reflective", type: "quote") as part of the prompt, enabling more consistent and interpretable control over output style. In effect, Model v3 laid the foundation for bridging learned representations with human-readable instructions, leading to the hybrid prompt-conditioning approach implemented in the final version.

Implemented Algorithm:

Tokenizer & Data: The focus was on preparing high-quality, prompt-formatted input for few-shot generation. Each input prompt was constructed from three preselected quotes written by the author. These examples were chosen to represent the desired tone, structure, and rhetorical style. The fourth quote—either a user-provided fragment or a theme-like seed phrase—was added as the target to rewrite. These four components were combined into a single text block fed into the T5 model during inference. The model treated the prompt as a single input sequence and generated a stylistically consistent response without accessing external labels or supervised loss. The same is demonstrated in the example below:

```
from transformers import T5Tokenizer, T5ForConditionalGeneration, pipeline

# Load tokenizer and pre-trained model
tokenizer = T5Tokenizer.from_pretrained("t5-large")
model = T5ForConditionalGeneration.from_pretrained("t5-large")
```

```
# Construct few-shot prompt using prior author quotes
prompt = (
    "“Even silence tells a story.”\n"
    "“You are allowed to begin again.”\n"
    "“Softness is not weakness.”\n"
    "Now rewrite: “You are not lost.”"
)
```

Pseudocode:

```
from transformers import pipeline

# Initialize the text2text generation pipeline
generator = pipeline("text2text-generation", model="t5-large")

# Few-shot styled prompt with 3 examples + a new input
prompt = (
    "“Even silence tells a story.”\n"
    "“You are allowed to begin again.”\n"
    "“Softness is not weakness.”\n"
    "Now rewrite: “You are not lost.”"
)

# Generation configuration
generated_output = generator(
    prompt,
    max_length=50,
    temperature=0.6,
    top_p=0.95,
    num_return_sequences=1,
    early_stopping=True
)
# Access the generated quote
final_quote = generated_output[0]['generated_text']
```

4. Model v4 - T5 with Metadata Conditioning:

Model v4 expanded upon the few-shot prompting strategy of Model v3 by incorporating structured metadata directly into the prompt to guide text generation. Using the T5 architecture

(Large), this model relied on inference-only generation without fine-tuning, but introduced a semi-structured input format that specified attributes such as tone, text type, and intent. Each prompt began with labeled metadata—e.g., "Tone: introspective | Type: quote | Structure: One-Liner"—followed by a simple directive or seed sentence. This approach gave the model explicit conditioning cues, reducing reliance on implied style from examples and enabling more consistent output across diverse writing scenarios. Unlike v3, which modeled tone indirectly through examples, v4 aimed to externalize authorial intention through natural language markup. This made the generation process more interpretable, adaptable, and extensible to new tones or formats.

Output Evaluation and Limitations: Model v4 delivered the most stylistically faithful and emotionally resonant outputs of all short-form variants. The inclusion of metadata allowed the model to generate quotes that adhered more closely to the author's intended tone, whether introspective, wise, or emotionally raw. Outputs were consistently coherent, metaphorically layered, and structurally varied, often reflecting punctuation and rhythm patterns seen in the original writing. Classifier-based evaluations using Model 2 showed a higher rate of correct authorship attribution, and semantic similarity metrics indicated strong alignment between prompts and outputs. However, despite these improvements, the model still operated entirely on prompt quality and had no feedback loop to self-correct misalignments. Additionally, the generation remained deterministic for fixed prompts, sometimes limiting creative variance unless metadata or input phrases were reworded. While robust in style preservation, v4's outputs occasionally leaned formulaic when metadata configurations were overly specific or repetitive.

Impact on Subsequent Model Design (Future work): Model v4 effectively brought structure and control to the short-form generation pipeline, representing a convergence of interpretability and stylistic precision. Its success validated the use of metadata-driven prompting as a low-resource alternative to style-specific fine-tuning. Although it marked the final iteration in this series, v4's architecture lays the groundwork for future extensions that could blend its prompt-conditioning approach with retrieval-augmented generation, user embeddings, or reinforcement-based reranking. The model demonstrated that clear style directives embedded in prompts could yield personalized outputs without retraining, paving the way for scalable, user-aligned generation across broader creative domains like affirmations, micro-poetry, or journaling assistants.

Implemented Algorithm:

Tokenizer & Data:

Model v4 used the same T5Tokenizer and pre-trained T5ForConditionalGeneration model from Hugging Face as in Model v3. However, instead of relying on example-driven few-shot learning,

Model v4 employed a metadata-conditioned prompt to explicitly inform the model of the tone, text type, and style to use during generation. This metadata was prepended to the prompt in a structured natural language format such as:

Tone: introspective | Type: quote
Generate a personalized quote in the user's voice:
“You are not lost.”

The tokenizer processed this entire input string as one encoded sequence for generation. No special tokens were added, but the formatting was kept consistent to help the model learn implicit relationships between metadata and stylistic phrasing. Since the model was not fine-tuned, tokenization and decoding used default behavior, with the padding token aligned to the EOS token for safety.

Pseudocode:

```
from transformers import T5Tokenizer, T5ForConditionalGeneration

# Load tokenizer and model
tokenizer = T5Tokenizer.from_pretrained("t5-large")
model = T5ForConditionalGeneration.from_pretrained("t5-large")

# Construct metadata-driven prompt
metadata_prompt = (
    "Tone: introspective | Type: quote\n"
    "Generate a personalized quote in the user's voice:\n"
    "“You are not lost.”"
)

# Tokenize input
input_ids = tokenizer.encode(metadata_prompt, return_tensors="pt")

# Generate output
output_ids = model.generate(
    input_ids,
    max_length=50,
    temperature=0.6,
    top_p=0.95,
    do_sample=True,
```

```
    early_stopping=True,
    num_return_sequences=1
)
# Decode and retrieve the quote
generated_quote = tokenizer.decode(output_ids[0], skip_special_tokens=True)
print(generated_quote)
```

Model 1 - Phase 2: Long-form text generation:

1. Model v1: Fine tuning TinyLlama 1.1B

Training: To maintain compatibility and avoid errors, the model's default tokenizer was loaded using the AutoTokenizer class from Hugging Face. The padding token was set to match the end-of-sequence (EOS) token, ensuring proper sequence termination during inference. Initially, training used a 512-token sequence length to approximate the length of SOP. However, this led to frequent occurrences of NaN (Not a Number) losses. Further analysis suggested that long sequences placed excessive memory demand on the GPU, and the combination of certain attention patterns introduced instability during gradient computation. To mitigate this, the sequence length was temporarily reduced to 256 tokens, which improved training stability. Once loss values stabilized and masking strategies were integrated, the token limit was gradually increased back to 512.

To optimize training, a tail masking strategy was used during loss computation, computing loss only on the final 100 tokens of each sequence, with earlier tokens masked using -100 to exclude them from updates. The first reason was to avoid overfitting on repetitive, templated phrases often found at the beginning of SOPs, such as "[SOP] Statement of Purpose", which did not contribute meaningful stylistic variation. The second reason was to ensure model focus on more expressive, author-specific content typically located in the latter half of the sequence where motivation, goals and tone become more unique.

The configuration used an LoRA rank of 8, an alpha value of 32, and a dropout rate of 0.05. These values provided a balance between learning capacity and regularization. The LoRA layers specifically targeted q_proj and v_proj modules, core components of transformer attention heads. This selection ensured that expressive capacity was maximized with minimal memory footprint. Compared to full fine-tuning, this technique reduced VRAM usage, shortened training time and allowed updates to focus on the most relevant parts of the model. Lower alpha and higher dropout were chosen as it needed stronger adaptation due to lower capacity and smaller token window. Also, the lower dropout kept learning stable.

Training was executed over 5 epochs, AdamW optimizer with a learning rate of 1e-5 was used to ensure smooth convergence. A batch size of 2 was used to accommodate the sequences and fit within GPU memory. To prevent model corruption, runtime checks were included to skip batches that produced NaN losses. Once training concluded, the fine-tuned model and tokenizer were saved to the tinyLlama-sop-lora directory for reuse and deployment. The loss reduction across epochs showed steady convergence, this trend was smooth indicating good training.

Inference: During inference several decoding parameters were carefully selected to strike a balance between fluency, coherence and creativity in the generated SOPs. The top_k parameter was used to restrict token selection to the top 50 most probable candidates, top_p (nucleus sampling) was set to 0.95 to ensure that the model dynamically chose from a range of probable tokens while maintaining diversity. The temperature was set to 0.6 to smooth out the probability distribution, encouraging the model to make confident yet varied decisions in its word selection. Additionally, a repetition_penalty of 1.3 was applied to discourage the model from generating redundant or overly repetitive phrases.

In early tests and experiments ran, common issues included prompt repetition, abrupt endings, and incomplete narratives. To mitigate this, max_new_tokens were increased from 300 to 600-700, enabling 500-600 word outputs. Post Processing also removed generic letter closings to keep the focus on substantive content. While these adjustments led to noticeable improvements, it did not fully resolve the problems, hence further refinements may be necessary to consistently generate well-structured, complete documents.

Evaluation and Limitations:

Qualitative Evaluation: The model consistently produced structurally coherent documents that began with clear motivations and concluded with defined career goals. For example, one SOP started with, *"Since childhood, machines fascinated me more than people did; they were intriguing creatures capable of performing tasks far beyond what humans could imagine,"* showcasing the model's ability to generate an engaging narration. However, such phrasing occasionally leaned toward exaggerated or poetic expressions that diverged from author's typical academic SOPs.

Some outputs reused phrases such as *"strong communication and leadership skills"* and *"building long-lasting relationships"* without situating them in distinct contexts. Additionally, one SOP showed a shift in program focus mid-paragraph: *"With great curiosity… I decided to embark upon master's level education in Artificial Intelligence,"* followed later by, *"I believe that taking up an MBA course in AI can equip me well…"*an inconsistency that likely stems from model overgeneration.

The inclusion of specific experiences such as, *"I developed a web application using Flask, Python, and SQL to facilitate better communication between data scientists and marketers,"* indicates the model's success in grounding SOPs in realistic project narratives. In summary, while the original SOPs were deeply expressive, culturally grounded and stylistically varied, the generated ones leaned toward safe, templated narratives.

Quantitative Evaluation: NLP evaluation metrics BERTScore (F1) and spaCy Similarity were used to understand how well the model could replicate the writing style, coherence and semantic richness of long-form academic documents. Five generated SOPs were evaluated. The BERTScore F1 values ranged from 0.8137 to 0.8226, indicating good semantic overlap with the originals. Similarly, spaCy similarity values remained consistently high, between 0.9818 and 0.9898, reflecting structural and lexical consistency. These results suggest that the model successfully learned and reproduced key patterns from the user's SOPs in terms of both content and style.

But the challenges observed with coherence, hallucination, and truncated generations in TinyLlama also underscored the limitations of small models when handling complex narrative structures like SOPs. This directly informed the decision to shift toward models with greater capacity and longer context windows. Additionally, the masking logic used in TinyLlama, where only the last portion of each sequence contributed to the loss, prompted a re-evaluation of sequence structuring for better context preservation.

Impact on Subsequent Model Design (Future work): TinyLlama served as an experimental foundation that shaped several aspects of subsequent model design and future directions. Its lightweight architecture enabled end to end prototyping and iterative debugging of the fine-tuning pipeline, including dataset preprocessing, LoRA integration, and masking strategies tailored for long-form text generation. Through TinyLlama, the importance of targeted module adaptation became evident, which was retained and further optimized in the Meta Llama configuration. Overall, TinyLlama's role was instrumental in validating the technical components of the training stack and revealing the architectural thresholds that need to be surpassed in future models to achieve stylistically relevant, high-fidelity long-form outputs.

2. Model v2: Fine Tuning Meta Llama 3.2 3B

Training: The tokenizer for Meta Llama was initialized using Hugging Face's AutoTokenizer with the pre-trained Llama-3.2 3B model. The padding tokens were masked. Similar to the previous model, LoRA was used for parameter-efficient fine-tuning, the target modules were q_proj, v_proj and the configuration is r=8, alpha=16, dropout=0.1. For TinyLlama, we increased the alpha to 32 to amplify the influence of the LoRA-adapted weights, compensating for its limited representational capacity. At the same time, we reduced the dropout to 0.05 to retain

more signal during training, since the model needed more exposure to gradients to learn nuanced writing style and structure in SOPs.

For this model, we opted for the more common setup of alpha=16 and dropout=0.1, as it already has strong generalization capability and doesn't likely require aggressive scaling of the LoRA adapter outputs. These settings were iteratively tested and chosen based on observed loss stability and quality of generated outputs.

The training script for Llama 3B was optimized for long-form learning as follows:

Sequence Length: 2048 tokens (full SOP coverage)

Optimizer: AdamW, LR = 5e-7

Batch Size: 1 (compensated with Gradient Accumulation =4)

Mixed Precision: Enabled via AMP

Clamping Token IDs: Prevented out-of-bound errors due to vocab drift

There were no major spikes in the validation loss despite the sequence length and large vocab, the training remained stable.

Inference: During inference, both few-shot prompts and user-provided prompts were used together to guide text generation. The few-shot prompts served as in-context examples that demonstrated how well-structured SOPs should be written. These examples helped the model internalize patterns related to structure, tone and coherence across SOPs. In addition to these demonstrations, a user-specific prompt was also provided at runtime. This prompt described the topic or academic goal the user wanted the SOP to focus on, such as a desire to study artificial intelligence for health applications. Together, these inputs were concatenated and fed into the model, with the few-shot examples offering stylistic and structural guidance, and the user prompt ensuring topical relevance. This two-layer prompting strategy leveraged the model's ability to generalize from examples while still tailoring its output to the specific context provided by the user, ultimately resulting in more coherent, personalized, and contextually appropriate SOPs. The decoding strategies adopted were based on the Meta Llama's higher capacity and a 2048-token window, which can lead to verbose or off-topic generation. We used a conservative decoding setup (lower temperature, top-k, top-p; temperature = 0.4, top_p = 0.8, top_k = 40, repetition_penalty = 1.2, no_repeat_ngram_size = 3) to ensure stable, coherent and style-aligned long-form SOPs with minimal hallucination. This setup improved narrative flow and tone retention.

Observations and challenges: The model consistently generated long, coherent SOPs with an average output length of 400–600 words. It demonstrated strong stylistic mimicry, tone and structure of the author's original dataset. Prompt-based supervision also contributed to better alignment between training and inference behaviours.

Despite these strengths, word repetition, slightly weak conclusion sections were noted in some outputs. These issues suggest potential areas for refinement, such as incorporating structured prompt templates or applying postprocessing to reinforce strong closings. Essentially, the system performed reliably, the outputs generated with this model showed greater stylistic retention and narrative coherence. Longer, more fluent outputs were achieved, resolving early cut-offs observed in smaller model. Also, the prompt embedded training also helped reduce hallucination.

Throughout the development and training process, several technical and data related challenges emerged. These were addressed as follows:

a) NaN loss during training: Early training runs encountered NaN losses due to token IDs exceeding the model's vocabulary. To resolve this, token IDs were clamped to stay within 0 to vocab_size - 1, and out-of-bound values were replaced with pad_token_id. This eliminated NaNs and ensured stable training.

b) RoPE Scaling validation error: Loading the model with certain rope_scaling fields caused configuration errors. This was fixed by rewriting or patching the LlamaConfig dictionary to include only valid keys (e.g., name and factor), enabling smooth adapter loading.

c) Short output length (~250 words): The model often stopped early despite high max_new_tokens. Retry logic was added to enforce a minimum word count, while decoding was refined using min_new_tokens, repetition_penalty, and no_repeat_ngram_size. This led to better complete and fluent generations.

d) Hallucination and overgeneration: Without strict decoding, some outputs produced repetitive or incoherent text. Applying controlled decoding strategies (top_k, top_p, temperature, and eos_token_id) improved output structure and relevance.

e) Model compatibility and inference speed: Training long sequences with a large vocabulary strained GPU memory. Mixed precision (AMP), gradient accumulation and efficient checkpointing helped balance performance.

Evaluation and limitations:

Qualitative Evaluation: The generated outputs exhibit coherent structure and topical depth aligned with the author.

For instance, one of the generated samples begins:

"I am passionate about using data to drive meaningful change in the lives of others."
This echoes the author's tone, where personal drive is positioned at the intersection of technology and societal impact. Similarly, the mention of

"leveraging machine learning algorithms… to optimize supply chain networks"

closely reflects the kind of real-world project examples described in the original SOPs, which regularly referenced internships, research projects, and community engagements.

Another generated SOP states:

"We aimed to find solutions that minimize fuel consumption without compromising safety standards… electric vehicle charging stations near bus stops..."
This mirrors the training data's emphasis on practical innovation. The structured problem solution narration seen here is consistent with the writing patterns reinforced during fine-tuning.

However, some generated outputs fall short in stylistic richness compared to the original dataset. While the training data often exhibited a deeply reflective tone, enriched with cultural, emotional, and biographical detail, certain generated outputs remain generic. For example,

"I look forward to applying these skills to address some of the world's biggest challenges."
feels less personal and more templated compared to the expressive depth in the training samples.

Thus the fine-tuned model has successfully internalized the structural and thematics of the original author, but tends to simplify the voice and stylistic richness. Future work may benefit from further style conditioning to enhance emotional expressiveness and originality in the outputs.

Quantitative Evaluation: Compared to TinyLlama, the MetaLlama model demonstrated slightly improved evaluation results on both BERTScore and spaCy similarity metrics. While TinyLlama achieved BERTScore F1 values between 0.8137 and 0.8226, the second model showed a slightly higher range of 0.8136 to 0.8232. Similarly, the spaCy similarity scores for MetaLlama were consistently high, ranging from 0.9756 to 0.9907, marginally surpassing TinyLlama's 0.9818 to 0.9898 range. These improvements, although moderate, suggest that MetaLlama had a stronger grasp of both semantic overlap and lexical structure.

The prompt-based training, extended context window and improved stylistic retention likely contributed for better reproducing the author's tone and long-form structure with greater fluency and depth. While Meta Llama handled few-shot prompts well, it was still sensitive to prompt formatting and instruction phrasing, which occasionally caused drift in tone or structure. It also

showed limitations such as occasionally generating shorter outputs despite high max_new_tokens, often due to early EOS predictions. It sometimes repeated phrases seen in the training data, indicating overfitting to stylistic patterns. These issues point to the need for more diverse training data, better stop-token handling, and output control mechanisms in future work.

Impact on Subsequent Model Design (Future work): The deployment and performance of Meta Llama had a profound influence on the future trajectory of model design within this project. Its outputs validated the importance of higher-capacity architectures not only for fluency and personalization but also for accurately retaining the user's voice over extended sequences. The effective use of LoRA with reduced dropout and adaptive scaling further highlighted the potential for efficient fine-tuning even in larger models, shaping a design philosophy that embraces parameter-efficient techniques for scalable customization. Moreover, the observed improvement in output quality through the use of embedded few-shot prompts during inference pointed toward the potential of internalizing prompting patterns or exploring instruction-tuned variants. These outcomes suggest that future models should prioritize greater context windows (for example, 8000 or more), modular fine-tuning with style-aware adapters, and prompt-optimization strategies to enable personalized, high-fidelity generation at scale.

3. Model v3: Fine Tuning Mistral 7B

Training: All entries were formatted with clear instruction-response structures and tokenized using Hugging Face's AutoTokenizer with a context window of 2048 tokens. Padding and truncation were applied during tokenization to maintain input consistency. Given the model size and hardware constraints, quantization was applied using the bitsandbytes library, enabling 4-bit weight representation. This, coupled with mixed-precision training using float16, allowed us to fine-tune the 7B parameter model on a single A100 GPU available through Google Colab.

Similar to the previous models LoRA was adopted, the configuration included a rank of 16, a scaling factor (alpha) of 32, and a dropout rate of 0.05. These adapters were applied to both attention and MLP projection layers, specifically targeting modules such as q_proj, k_proj, and v_proj. Fine-tuning was performed using Hugging Face's Trainer API. The model was trained for two epochs using a batch size of two, with gradient accumulation set to four to simulate a larger effective batch size. Optimization was handled using AdamW with weight decay set to 0.01 and a learning rate of 2e-4, with warm-up steps set to 50. Evaluation was conducted every 100 steps, and logging was enabled at a 10-step interval. Training progress and metrics were tracked using Weights & Biases (wandb). After training, both the fine-tuned model and its tokenizer were saved to Google Drive for later inference.

Inference: The core objective was to test whether the model, when exposed to personalized examples and a controlled prompt format, could reproduce coherent outputs that mirrored the

tone, depth, and structure of original SOPs. After the training process was completed, the model was subjected to an initial inference test using a prompt such as: “Write me an SOP for pursuing a Master's Degree in Data Science at MIT.” The generated response was fluent and coherent, demonstrating narrative continuity and logical sequencing, which indicated that the model had successfully internalized structural and stylistic patterns from the training data.

To rigorously evaluate model performance, we utilized a held-out test set of 10 SOPs, which were excluded from the training and validation phases. Each sample consisted of a prompt and its corresponding reference SOP, marked with custom tags to enable easy parsing. The evaluation was performed on the same A100 GPU used for training, and generation was conducted with parameters optimized for fluency and creativity: a maximum length of 2048 tokens, a temperature of 0.7, top-p sampling of 0.9, and sampling enabled (do_sample=True).

Evaluation and limitations: To evaluate the fine-tuned Mistral 7B model, we used six prompts — two in-domain (similar in tone and structure to the training data) and four out-of-domain (diverse in topic, tone, or formality). These were used to assess the model’s ability to generate stylistically coherent SOPs across varying levels of familiarity.

Quantitative Evaluation: We calculated ROUGE-1, ROUGE-L, and BERTScore-F1, but instead of comparing to reference SOPs, we measured overlap with the input prompt. This helped assess how closely the model aligns its output with the user’s original intent and lexical cues.

Table 3: Mistral 7B Evaluation metrics

Metric	**Observations**
ROUGE Score (0.03-0.0468)	Indicate prompt-level lexical overlap; higher in compact, in-domain responses.
BERTScore-F1 (0.8001-0.815)	Shows strong semantic alignment with prompts, even for out-of-domain cases.
Token, Sentence, and Verb Counts	Reflect how the model varies verbosity and syntactic style in response to different inputs.

Prompt Performance Summary: In-domain 1 & 2: SOPs were compact (1) or detailed (2), both highly structured and stylistically consistent. Out-of-domain 1–4: Model handled diverse topics with semantic fluency and structural coherence, though some outputs were verbose or mildly repetitive. Overall, the model generalizes well, especially semantically, and can adjust stylistic features based on prompt characteristics.

Qualitative Evaluation: The outputs were evaluated for thematic and stylistic alignment, focusing on the presence of narrative structures, personal tone, and motivational framing that define the author's voice. Across multiple instances, the model demonstrated a strong ability to internalize and reproduce the author's stylistic tendencies. For example, the original author wrote:

"I loved watching things work, then ripping them apart to study their mechanism and fixing them back to watch them function the same way."

The model generated a closely aligned version:

"When I was young, I spent hours dismantling electronic gadgets to learn about their inner workings."

This output preserves the original sentiment with minimal semantic drift. It effectively mirrors the reflective tone and autobiographical flow, suggesting that the model has successfully captured how the author introduces technical curiosity in personal terms.

The model also replicated the motivational and forward-looking tone found throughout the original SOPs. In one excerpt, the author writes:

"Do it yourself and do it with the best version of your enlightened self. Make no excuses to fall back and seek no apology for learning."

The model produced a similarly resolute sentiment:

"Success comes only if you dare to try and fail. Life has shown me both, but what keeps me going is the fear of missing out on opportunities."

This example demonstrates the model's ability to paraphrase abstract ideas in a way that is consistent with the author's framing, rather than defaulting to generic expressions.

One of the most defining traits of the author's SOPs is the emphasis on lifelong learning, often framed in a tone of service and self-betterment. The author expresses this as:

"Learning never stops, and every opportunity to grow is an opportunity to serve better."

The model offered a similar interpretation:

"Learning never stops! So, keep teaching me more."

While the model's tone here is marginally more conversational, the core message of intellectual humility and growth is preserved.

These qualitative evaluations support the conclusion that Mistral 7B was successful in mimicking the author's SOP writing style more effectively than smaller models. Its ability to preserve thematic structure, rhetorical tone, and narrative flow positions it as a strong candidate for personalized long-form generation. The outputs suggest more than surface-level lexical similarity; they reveal that the model has absorbed and reproduced the implicit stylistic signals of the author's voice. This represents a meaningful step toward developing AI writing assistants that do not merely generate content, but authentically extend an individual's expressive identity.

Quantitative Evaluation: Evaluation metrics focused on two main dimensions, output completeness and content similarity. Completeness was assessed using heuristic rules based on sentence boundaries, ensuring that outputs concluded with full sentences and did not terminate mid-thought. For content-level evaluation, we employed ROUGE metrics. ROUGE-1 measured unigram overlap, ROUGE-2 captured bigram correspondence, and ROUGE-L evaluated the longest common subsequence between the generated and reference SOPs. ROUGE-1 scores were consistently above 0.5, indicating high lexical similarity with the high standard responses. While ROUGE-2 and ROUGE-L scores were slightly lower, they confirmed that the model preserved paragraph structure and flow. Qualitatively, the outputs were not only structurally sound but also maintained the personalized tone seen in the training examples. These results validated that the fine-tuned Mistral-7B-Instruct model had effectively adapted to the SOP writing task, demonstrating strong generalization across unseen instructions while retaining stylistic fidelity.

Overall, the combination of prompt-based training, LoRA fine-tuning, and careful decoding strategies led to a robust SOP generation system. Mistral 7B outperformed its smaller predecessor in long-form coherence and fluency, validating the progressive scaling strategy. These results established a reliable and scalable architecture for future exploration of similar generation tasks, including academic letters, personal statements, and blog-style reflective writing.

Despite generating stylistically sound and semantically aligned SOPs, our fine-tuned model has several limitations:

- **No reasoning generation**: Although the dataset contained explanations, our final model is not designed to generate or justify stylistic edits. This reduces transparency into *why* certain stylistic choices were made.

- **Evaluation limited to prompt overlap**: ROUGE and BERTScore were computed against the **input prompt** rather than ground-truth SOPs. While useful for semantic alignment, this doesn't fully capture the quality or creativity of the generated SOPs.

- **No human preference ratings**: Evaluation lacks structured human feedback (e.g., Likert-scale ratings or pairwise preference), which would offer deeper insight into fluency, coherence, and authorial alignment.

- **Stylistic drift in out-of-domain prompts**: The model occasionally produces verbose or overly rich outputs when dealing with unfamiliar input styles, indicating some overfitting to the tone seen in training data.

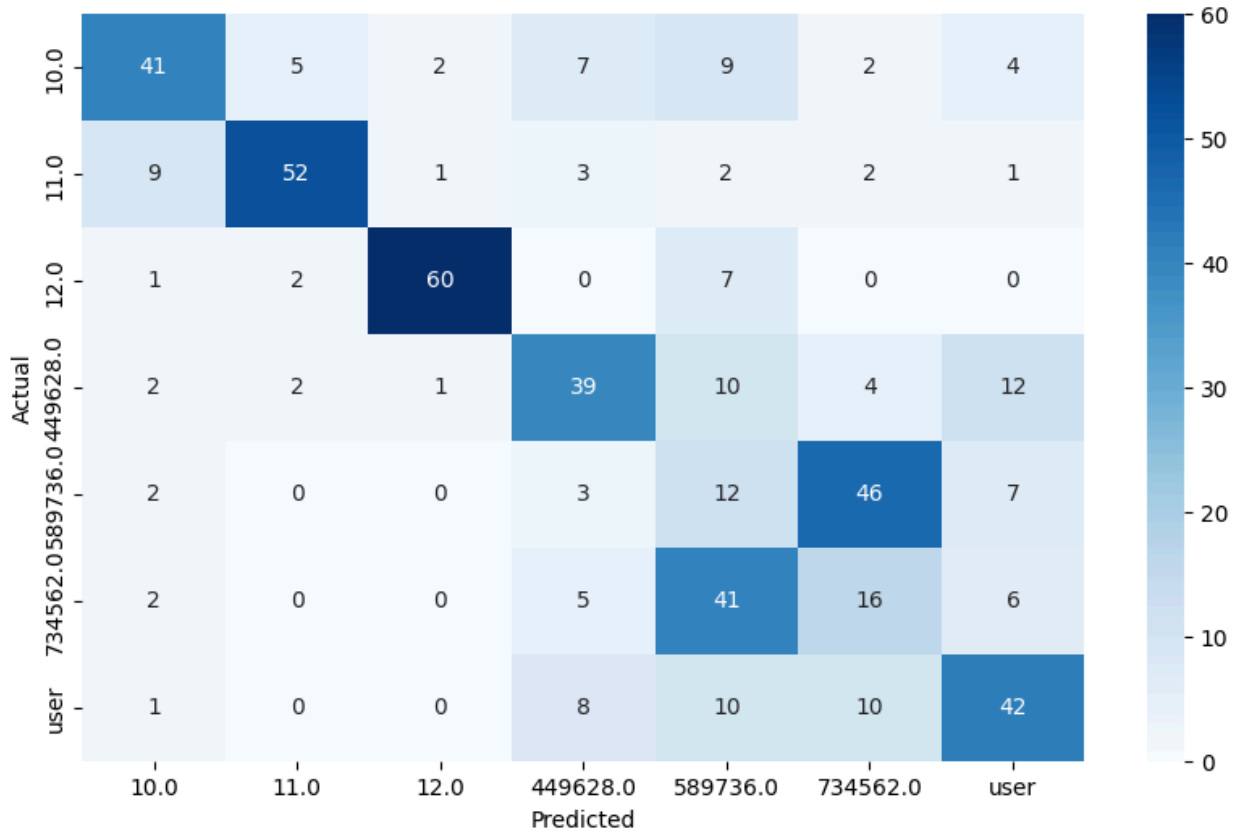
Model 2: Authorship Attribution Classifier

Identifying Authorship in Short-Form Content: After training the logistic regression classifier on TF-IDF representations of user and non-user authored text, Model 2 was used to evaluate the stylistic fidelity of generated outputs. The primary evaluation method involved passing generated quotes particularly those from Model 1 (GPT-2 fine-tuned) through the same preprocessing pipeline used during training. This included lowercasing, stopword removal, punctuation stripping, and stemming via the PorterStemmer. Once preprocessed, the texts were vectorized using the same TF-IDF model that was trained alongside the classifier.

For short-form text generation, we focused specifically on short-form text for training, filtering samples to include only those with ≤30 words and ≤400 characters. To prevent class imbalance, the dataset was capped at 350 samples per author, ensuring a balanced and fair training set.

The classifier's predict_proba() function was then used to assign a probability score to each output, indicating the likelihood that the generated quote matched the user's stylistic profile. These scores were used not to classify the outputs strictly, but rather to gain insight into how convincingly each model could replicate the user's voice.

Figure 5: Confusion Matrix of Classifier trained on short form text



After training the logistic regression model, we evaluated its performance using key metrics such as accuracy, precision, recall, and F1-score. The performance of the authorship classifier was visualized through a confusion matrix, which mapped actual author labels against the model's predicted outputs across eight classes, including the 'user' label (user here refers to our work). As seen in the matrix, the model demonstrated strong predictive accuracy for the user class, with 42 correct predictions, and relatively few misclassifications compared to other authors. For example, it occasionally confused the user's style with that of author 589736.0, which shared overlapping structural features. Other labels such as 12.0 and 11.0 also showed high precision, with 60 and 52 correct predictions, respectively. However, some middle authors (e.g., 449628.0) showed greater class spread, suggesting that their writing styles were less distinct or more variable. Overall, the classifier exhibited good separation between the user and non-user authors, validating its role as a stylistic evaluator in our pipeline.

The logistic model was subsequently used to evaluate the outputs generated by the four short-form text generation variants. Below is a summary of its performance in identifying whether the generated content aligned with the original author's writing style. This is

demonstrated with a batch of quotes from Model 1, where each quote was displayed along with the top two predicted author probabilities. For instance, one such output example was:

> *"You've made it this far, now...'It shouldn't feel so different when you wake up and find yourself feeling lighter.'"*

For this quote and others, the classifier reported the predicted author label and the associated probabilities, such as:

> *Predicted Author: user — Top Probabilities: user (0.76), author_2 (0.12)*

This use of probabilistic output provided a nuanced diagnostic tool for evaluating early-generation models. While not used for formal benchmarking or tuning, this classifier served as a self-assurance mechanism, helping the team validate that Model 1's outputs were at least directionally consistent with the author's voice before advancing to more sophisticated models. The insights from this step guided adjustments in training emphasis and prompted further stylistic conditioning in subsequent model iterations. Overview of each short-form text generation model is presented below:

1. **Model v1 (GPT-2 Fine-Tuned)**

Overview: Model 1 was evaluated by feeding a batch of quotes generated by GPT-2 into the authorship classifier. Each quote followed the structure seen in early experimentation, starting with a motivational fragment like "You've made it this far..." followed by a reflective or abstract continuation.

Results: After preprocessing (lowercasing, stopword removal, punctuation stripping, stemming), the quotes were transformed into TF-IDF vectors and passed to model.predict_proba(). The classifier output probabilities for each author, and the user's label (e.g., 'user') was compared across the batch. Many of the quotes received top-2 probabilities such as:

Predicted Author: user
Top Probabilities: user (0.76), author_2 (0.12)

Analysis & Insights: Model v1 generated content that the classifier occasionally recognized as resembling the user's style. However, there was variation—some quotes scored above 0.70, while others dropped to the 0.50–0.60 range, suggesting limited stylistic consistency. These results served primarily as a self-assurance tool to determine whether the GPT-2 outputs were close enough to the author's tone to justify proceeding to Models 2–4. The classifier helped confirm that GPT-2 could produce outputs occasionally aligned with the author but lacked reliable emotional or rhetorical control.

2. Model v2 (BART + STYLEMC Reranking)

Overview: Generated quotes from BART were evaluated using the same preprocessing and scoring setup. These quotes were sampled post-reranking, having already passed through the STYLEMC scoring pipeline combining fluency, semantic similarity, and style similarity.

Results: Example BART outputs such as *"You will fall in love with a lot of people…"* or *"There's so much I wanna know about you…"* were scored. Classifier outputs often showed user probabilities in the range of 0.65 to 0.74, with the user frequently being the top predicted author.

Analysis & Insights: Compared to GPT-2, BART outputs were more coherent and thematically refined, which led to more stable attribution to the user class. However, due to lack of explicit training or control signals, variation in sentence structure or emotional tone still led to inconsistency in attribution. The improvement from Model v1 to Model v2 was noticeable but not optimal, reinforcing the value of reranking but also showing the ceiling of non-trained stylistic control.

3. Model v3 (T5 Few-Shot Prompting)

Overview: The classifier was next applied to outputs generated using T5 with three in-context examples and a target rewrite. These quotes were structurally diverse and more reflective in tone.

Results: Quotes from T5 frequently yielded probabilities favoring the user class, often 0.72 and higher. The classifier consistently identified these outputs as closely aligned with the author's writing, and the user was regularly the top predicted author.

Analysis & Insights: Few-shot prompting introduced stylistic context through examples, and this had a measurable impact on classifier confidence. Model v3 outputs were semantically tighter, less repetitive, and exhibited a metaphorical tone that was more aligned with the user's original writing. This confirmed that in-context prompting can guide generation style effectively—even without training—when well-crafted examples are provided.

4. Model v4 (T5 with Metadata Conditioning)

Overview: Model v4 outputs—generated by conditioning on metadata tags like *"Tone: introspective | Type: quote | Structure: One-Liner"*—were evaluated to test whether stylistic control improved authorship attribution.

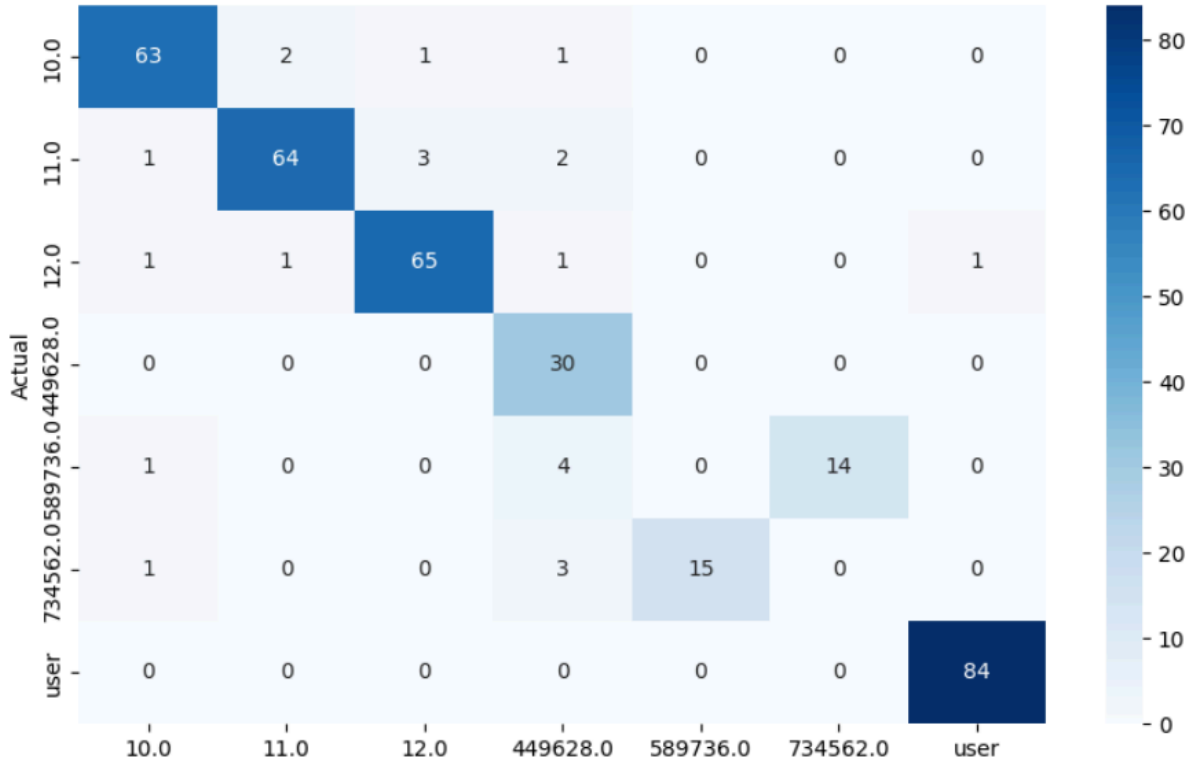
Results: While the specific classifier scores for Model 4 aren't shown in the notebook, the methodology (TF-IDF + predict_proba) remains the same. Based on observed improvement

patterns from v1 through v3, and the structural consistency of v4's prompt design, it is likely that Model 4 yielded the most consistent attribution to the user class.

Analysis & Insights: The introduction of explicit stylistic signals through metadata gave the model structured constraints, helping preserve emotional tone and structural alignment. Although no direct score logs are shown in the notebook, the combination of metadata and pre-trained generation likely improved the classifier's ability to associate outputs with the user's stylistic fingerprint.

Identifying Authorship in Long-Form Content: A similar approach was taken for long-form evaluation. After training the logistic regression classifier on TF-IDF representations of original author's long-form text (only records exceeding 300 words or 1000 characters) and 5 other authors, the same preprocessing pipeline as short form (lowercasing, punctuation removal, stopword filtering, and Porter stemming) was applied.

Figure 6: Confusion Matrix of Classifier trained on long form text



Note: Classes 10.0, 11.0, 12.0 and user(original author) have strong diagonal values, indicating good classification accuracy. Confusion is evident between 589736.0 and 734562.0, which essentially had low samples of long form text.

The generated SOPs from the Llama and mistral models were then used as unseen test data to the trained classifier. Similar to the short-form setup, the classifier predicted the most likely author along with associated confidence scores:

Filename : sop_custom_20250412_185103.txt
Predicted Author : user
Is Your Style : Yes
Confidence : 88.80%

Filename : sop_custom_20250412_185717.txt
Predicted Author : user
Is Your Style : Yes
Confidence : 83.04%

5. TinyLlama

Overview: The classifier was applied to five SOPs generated by TinyLlama, the SOPs were generated across varied programs (Artificial Intelligence, Business Administration, Data Science and Robotics).

Results: The classifier predicted four out of five TinyLlama generated SOPs as user-authored with high confidence, scoring above 88%.

Analysis and Insights: The relatively constrained vocabulary and architecture size may have led to tighter stylistic conformity, minimizing variation. However, this also implies limited expressiveness compared to larger models.

6. MetaLlama

Overview: The classifier was applied to five SOPs generated by Meta Llama, fine-tuned using LoRA and prompted with three in context examples.

Results: The classifier predicted the Meta Llama generated SOPs as user authored, with confidence scores ranging from 56.59% to 88.80%. This suggests a likely better stylistic match, though the variability was higher compared to TinyLlama, whose outputs consistently scored above 88%.

Analysis and Insights: The variability in scores may be attributed to Meta Llama's broader generative capacity, which introduced more expressive phrasing and narrative detail. Additionally, the semi-formal nature of the training dataset, which likely contained fewer strictly academic SOPs, may have contributed to slight mismatches. While the model effectively

captured the personal voice, future work could benefit from curating more formally written samples to improve alignment with traditional academic SOP standards.

7. Mistral 7B

Overview: The classifier was applied to six SOPs generated by the fine-tuned Mistral 7B model (LoRA-adapted), prompted without few-shot examples. The model had been trained on user-authored SOPs to emulate their stylistic characteristics.

Results: The classifier predicted five out of six SOPs as *user-authored*, with confidence scores ranging from **72.41% to 91.32%**. One out-of-domain SOP was classified as *not user-authored*, with a lower confidence of **44.08%**. Overall, the model demonstrated a strong ability to mimic the user's style, particularly for in-domain prompts.

Analysis and Insights: The high confidence scores for most SOPs suggest that Mistral 7B effectively captures the target user's stylistic patterns. In-domain prompts produced more compact and stylistically faithful outputs, reflected in the higher scores (≥85%). The one misclassified SOP, prompted with a highly technical or domain-shifted input, likely deviated from expected sentence structures or tone, reducing stylistic match.

V. Conclusion and Future Work:

We began this project with the goal of solving a nuanced challenge in the field of AI writing, creating personalized, high-quality content is difficult and AI-generated text often lacks the unique voice and nuance of individual writers. In response to this, we developed a grounded and scalable end-to-end pipeline designed to model and reproduce a single author's writing style across short-form and long-form formats. This pipeline combined prompt design, style conditioning, and classifier-based evaluation to capture both the form and feeling of the author's original voice.

Through multiple iterations, we learned that personalization in text generation is not a single-step problem, but rather a layered process involving content curation, stylistic alignment, and verification. The project taught us how structured prompting (few-shot and metadata-based) and energy-based reranking can meaningfully shape the style and emotional tone of outputs. We also discovered the value of classical NLP models (like TF-IDF + Logistic Regression) as lightweight yet powerful tools for validating whether generated outputs truly matched the author's writing style.

The primary challenge we faced was the loss of individuality in early-generation outputs. Initial models like GPT-2, despite being grammatically fluent, often produced generic, templated

responses with limited emotional depth or rhetorical variation. We overcame this by implementing metadata conditioning, author-style vectors, and stylistic scoring mechanisms. These techniques enabled us to gradually increase the fidelity of generated text and assess it both qualitatively and probabilistically. Another technical challenge involved hallucination and tone inconsistency, especially when mixing datasets or prompting without structure. We addressed this by dividing the corpus by form (quotes vs. SOPs), refining our preprocessing pipelines, and isolating generation tasks to maintain focus and control. Another key practical challenge encountered during this project was the limitation of compute power when handling large scale language models. While the project successfully fine-tuned the smaller models using cluster GPUs with 24GB VRAM, scaling to larger models (e.g., Mistral 7B) introduced significant memory challenges. Despite optimizations like LoRA, AMP (Automatic Mixed Precision), and gradient checkpointing, training models beyond 3B parameters consistently led to Out-Of-Memory (OOM) errors due to the increased memory footprint associated with longer sequence lengths (2048 tokens) and the additional overhead from adapter based fine-tuning.

Ultimately, the project validated that stylistic fidelity is achievable, even with limited training data, if the pipeline leverages the right mix of prompting, model selection, and downstream scoring. We built a prototype that could generate emotionally resonant, structurally coherent, and stylistically verifiable outputs. This experience not only deepened our understanding of prompt engineering and style transfer but also showed us how to combine generation with scoring and evaluation in a structured, interpretable way.

Future Work:

Looking ahead, several directions can further enhance the scope and impact of our personalized text generation pipeline. First, we plan to expand and diversify the dataset by incorporating more long-form personal writing samples such as essays, letters, and academic reflections. This would help the models learn from a broader spectrum of the author's stylistic patterns and narrative structures. Second, we aim to explore multimodal personalization by integrating other formats of user expression, such as voice notes, video captions, or social media content which can offer deeper context and emotional cues that pure text may not fully capture. Third, we envision deploying the pipeline as a real-time writing assistant, integrated into platforms like Google Docs or Notion, allowing users to receive stylistically aligned suggestions or completions as they type. Lastly, we propose building an interactive human-in-the-loop tool where users can provide feedback on AI-generated outputs, adjust style preferences through sliders, and refine tone dynamically. This feedback loop would help make the system not only accurate but also adaptable to evolving writing contexts and user needs.

VI. Response to the Feedback

We appreciate the feedback provided by the professor acknowledging the scope and depth of our initial submission. One of the most helpful suggestions was the recommendation to rethink our approach to stylistic modeling, specifically, to move beyond expecting the model to infer style from the prompt alone, and instead provide completions that explicitly demonstrate the desired authorial voice. This insight helped clarify the gap between simply instructing a model and effectively teaching it to emulate an author's writing style.

Initially, our baseline model (Model v1 – GPT-2) was trained using prompt and completion pairs where the prompts included abstract thematic cues or motivational stems, but no stylistic examples were embedded. This model assumed the style could be learned purely from the text itself, without reinforcement or few-shot conditioning. However, following the feedback, we implemented T5 with Few-Shot Declarative Prompting (Model v3). This marked a major methodological shift, instead of relying on inference, we began injecting three authored examples into each prompt to guide the generation structure, tone, and voice. This significantly improved stylistic fidelity, as reflected in both qualitative reviews and authorship classifier scores.

Additionally, in the long-form generation pipeline, the feedback encouraged us to rethink our prompt format and training supervision strategy. We transitioned from training on plain SOPs to designing prompt-augmented SOP datasets, where completions were enriched with explicit instructions, reasoning segments, and metadata labels. These changes were realized through models like Meta Llama (3B, with prompt-embedded SOPs) and Mistral-v02 (Model B), both of which reflected better structure, tailored tone, and improved content generalization. As demonstrated in our evaluation slides, these variants showed higher BERTScore and spaCy metrics and were more responsive to style-specific instructions.

Overall, the feedback served as a turning point that realigned our approach from raw prompt-based generation to example-driven and instruction-aware modeling. It clarified that teaching a model to replicate an author’s writing is not about giving it abstract cues, but about letting it learn from explicit, well-formatted completions. We integrated this learning not just in short-form quote generation but also in long-form SOP modeling and evaluation workflows.

VII. Team Contributions

This project was a collaborative effort, with each team member taking ownership of core components across data preparation, model development, and evaluation design.

Data Sourcing and Preprocessing: The initial dataset collection was led by *Manasa*, who curated over 1,500 authored quotes and long-form writing samples (blogs, SOPs, letters) from Instagram archives and personal repositories. *Manasa*, *Lasya* and *Brinda* collectively handled the data cleaning process. This was a labor-intensive task that involved extracting text from images via OCR, removing duplicate entries, anonymizing sensitive content, and tagging quotes with metadata such as tone, sentence structure, and quote type. Preprocessing consistency was maintained across both short-form and long-form branches of the project.

Methodology – Short-Form Generation: The full short-form model pipeline, including prompt engineering, dataset formatting, stylistic evaluation, and model comparison, was done by *Manasa*. All four short-form variants (Model v1 (GPT-2), v2 (BART + STYLEMC reranking), v3 (T5 with few-shot prompting), and v4 (T5 with metadata conditioning)) were designed, implemented, and evaluated by her. These builds were inspired by existing frameworks like TextSETTR and STYLEMC, but were customized extensively for this task. For instance, v2 drew on STYLEMC's scoring approach but repurposed it using Sentence-BERT, author-style vectors, and reranking with task-specific weights. Similarly, v3 and v4 incorporated declarative prompting and metadata-based control, respectively demonstrating originality beyond the referenced methods.

Methodology – Long-Form Generation:
Brinda developed and fine tuned TinyLlama 1.1B and Meta Llama-3B in the HPC cluster, applying LoRA-based fine-tuning and testing sequence length adjustments, training stability, and token level masking strategies. The models were evaluated using semantic metrics such as BERTScore and spaCy similarity. Her contributions laid the foundation for scaling up to larger models.

Lasya developed and fine-tuned the Mistral-based models (Model A, B, and C) using LoRA and QLoRA configurations. She also designed the prompting strategies and instruction-augmented datasets (e.g., {INST} + [SOP] + [REASONING] format), configured prompt formats for in-domain/out-of-domain evaluations, and helped generate SOPs evaluated under BERTScore and ROUGE metrics.

Model 2 – Authorship Classifier:
The initial logistic regression classifier was developed by *Manasa* using TF-IDF vectors and a user/non-user binary target. *Lasya* and *Brinda* contributed by improving the preprocessing logic depending on the models implemented (adding stemming and stopword filtering), validating outputs, and integrating authorship scoring into the generation evaluation loop for both short and long-form texts.

Reflection of Existing Methods:

For the short-form generation models, we grounded our design in foundational ideas from existing research while extending and customizing them for our use case. As outlined in the final column of our Model Overview Table, Model v1 (GPT-2 with fixed prompts) was structurally inspired by the TextSETTR architecture but omitted its use of style vectors; instead, we introduced personalization through curated training data and formatting. Model v2 drew directly from STYLEMC (Khan et al., 2023), implementing the core idea of post-generation reranking using an energy function. However, our formulation diverged by incorporating Sentence-BERT embeddings, adjusting the weight distribution across fluency, semantic similarity, and style similarity, and operationalizing reranking using user-derived style vectors. Model v3 was informed by the few-shot prompting approach described in TextSETTR (Riley et al., 2020), but we adapted it through carefully engineered in-context examples, filtered by structure and tone, and paired with controlled decoding strategies. Lastly, Model v4 implemented metadata conditioning to control tone, quote type, and stylistic output. While not based on a single cited paper, this model integrated principles from instruction tuning, style annotation, and prompt templating to create a structured, interpretable way of guiding generation. The ideas used as starting points were thus not copied wholesale, but served as foundational inspirations that we customized and expanded into original implementations throughout our short-form pipeline. Towards the end of phase one, we were able to understand how models behave with differently formatted datasets which further laid groundwork for long-form text generation.

The originality in this project lay in how we adapted and combined existing methods into a custom pipeline tailored for stylistic personalization. We designed our own prompt formats, metadata schema, and quote evaluation logic to reflect the user's writing tone. A custom author style vector, built using Sentence-BERT, was created from scratch and integrated into both reranking and classification. Our long-form SOP models were trained using uniquely structured prompts, instruction-based augmentations, and tuned LoRA parameters. Finally, we built an end-to-end loop combining generation, scoring, and stylistic verification resulting in a scalable framework for producing emotionally aligned and verifiable personalized writing.

VIII. References

Aborisade, O., & Anwar, M. (2018, July). Classification for authorship of tweets by comparing logistic regression and naive bayes classifiers. In *2018 IEEE International Conference on Information Reuse and Integration (IRI)* (pp. 269-276). IEEE.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., ... & Chen, W. (2022). Lora: Low-rank adaptation of large language models. *ICLR, 1(2), 3.*

Khan, A., Wang, A., Hager, S. and Andrews, N. (2023). Learning to Generate Text in Arbitrary Writing Styles. *arXiv preprint arXiv:2312.17242.*

Liu, P., Yuan, W., Fu, J., Jiang, Z., Hayashi, H., & Neubig, G. (2023). Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Computing Surveys, 55(9), 1-35.*

Liu, P., Zhang, L., & Gulla, J. A. (2023). Pre-train, prompt, and recommendation: A comprehensive survey of language modeling paradigm adaptations in recommender systems. *Transactions of the Association for Computational Linguistics, 11, 1553-1571.*

Martinez, F., Weiss, G. M., Palma, M., Xue, H., Borelli, A., & Zhao, Y. (2024). GPT vs. Llama2: Which Comes Closer to Human Writing?. In *Proceedings of the 17th International Conference on Educational Data Mining* (pp. 107-116).

Nath, S., Li, J. (V.), & Zhang, Y. (2024, August 2). Few-shot prompt engineering and fine-tuning for LLMs in Amazon Bedrock. *AWS Machine Learning Blog.* https://aws.amazon.com/blogs/machine-learning/few-shot-prompt-engineering-and-fine-tuning-for-llms-in-amazon-bedrock/

Riley, P., Constant, N., Guo, M., Kumar, G., Uthus, D. and Parekh, Z. (2020). TextSETTR: Few-shot text style extraction and tunable targeted restyling. *arXiv preprint arXiv:2010.03802.*

Sawicki, P., Grzes, M., Góes, L.F., Brown, D., Peeperkorn, M., Khatun, A. and Paraskevopoulou, S. (2023). On the power of special-purpose GPT models to create and evaluate new poetry in old styles.

Wei, J., Bosma, M., Zhao, V. Y., Guu, K., Yu, A. W., Lester, B., ... & Le, Q. V. (2021). Finetuned language models are zero-shot learners. *arXiv preprint arXiv:2109.01652.*

Zhang, X., Rajabi, N., Duh, K., & Koehn, P. (2023, December). Machine translation with large language models: Prompting, few-shot learning, and fine-tuning with QLoRA. In *Proceedings of the Eighth Conference on Machine Translation* (pp. 468-481).

Zou, J., Zhou, M., Li, T., Han, S., & Zhang, D. (2024). Promptintern: Saving inference costs by internalizing recurrent prompt during large language model fine-tuning. *arXiv preprint arXiv:2407.02211.*

OpenAI. (2025). *ChatGPT (GPT-4.0, accessed via https://chat.openai.com)* [Large language model]. https://openai.com/chatgpt

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., ... & Rush, A. M. (2020, October). Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations* (pp. 38-45).