

**Stylistic Models for  
Authorial Recreation:  
A Comparison of Transformer-based  
Language Models for Author-Styled  
Latin Text Generation**

*Rufus Behr*

A dissertation submitted in partial fulfilment  
of the requirements for the degree of  
**Bachelor of Science**  
of the  
**University of Aberdeen.**



Department of Computing Science

2022

# Declaration

No portion of the work contained in this document has been submitted in support of an application for a degree or qualification of this or any other university or other institution of learning. All verbatim extracts have been distinguished by quotation marks, and all sources of information have been specifically acknowledged.

Signed: Rufus Behr

Date: 2022

# **Abstract**

It is nearly impossible for text to be devoid of its author's style. Whether we like it or not, our fingerprints are left on the sentences that we construct. And, whilst stylistic text transfer (the process of adjusting text for stylistic elements like politeness, casualness, etc.) is a fairly well-trodden research field, author-styled generation is not. This dissertation, as far as the author is aware, is the first research project that aims to generate author-styled Latin text. It therefore contributes to the research fields of author-styled text generation and low-resource languages.

# **Acknowledgements**

Firstly, I would like to thank my advisor Dr. Ehud Reiter for allowing me to undertake this project and for all the support and insight he provided throughout it. I would also like to thank Dr. Sam Newington for putting me in contact with Latin experts that were able to evaluate my project. Lastly, I would like to thank my family for their unwavering support throughout my degree.

# Contents

|          |  |           |
|----------|--|-----------|
| <b>1</b> | <b>Introduction</b>                    | <b>9</b>  |
| 1.1      | Motivation . . . . .                   | 9         |
| 1.2      | Objectives . . . . .                   | 10        |
| <b>2</b> | <b>Background and Related Work</b>     | <b>12</b> |
| 2.1      | Problem Domain . . . . .               | 12        |
| 2.1.1    | Lingua Latina . . . . .                | 12        |
| 2.1.2    | Latin Grammar . . . . .                | 13        |
| 2.1.3    | Style in Latin . . . . .               | 13        |
| 2.1.4    | Latin Composition . . . . .            | 13        |
| 2.2      | Related Work . . . . .                 | 14        |
| 2.3      | Background . . . . .                   | 15        |
| 2.3.1    | Transformers . . . . .                 | 15        |
| 2.3.2    | BERT . . . . .                         | 15        |
| 2.3.2.1  | Latin BERT . . . . .                   | 16        |
| 2.3.3    | Large Language Models . . . . .        | 16        |
| 2.3.3.1  | GPT-3 . . . . .                        | 17        |
| 2.3.3.2  | LLaMA . . . . .                        | 18        |
| 2.4      | Prior Work: Latin Authorship . . . . . | 18        |
| 2.4.1    | The Dataset . . . . .                  | 18        |
| 2.4.2    | Results . . . . .                      | 19        |
| <b>3</b> | <b>Design &amp; Methodology</b>        | <b>20</b> |
| 3.1      | Requirements . . . . .                 | 20        |
| 3.1.1    | Functional Requirements . . . . .      | 20        |
| 3.1.2    | Nonfunctional Requirements . . . . .   | 20        |
| 3.1.2.1  | Code Quality . . . . .                 | 20        |
| 3.2      | Design Prerequisites . . . . .         | 21        |
| 3.2.1    | Dataset . . . . .                      | 21        |
| 3.2.2    | Author Selection . . . . .             | 22        |
| 3.3      | Latin BERT . . . . .                   | 22        |
| 3.4      | Transformer . . . . .                  | 25        |
| 3.5      | ChatGPT . . . . .                      | 26        |
| 3.6      | LLaMA . . . . .                        | 27        |

---

|          |  |           |
|----------|--|-----------|
| 3.6.1    | Alpaca   | 28        |
| 3.6.2    | LoRA   | 28        |
| 3.6.3    | Alpaca with LoRA                               | 29        |
| 3.6.4    | Approach for Latin                             | 29        |
| <b>4</b> | <b>Implementation</b>                          | <b>31</b> |
| 4.1      | Development                                    | 31        |
| 4.1.1    | Language & Libraries                           | 31        |
| 4.1.2    | Cloud Computing                                | 31        |
| 4.2      | Data   | 32        |
| 4.3      | Models   | 32        |
| 4.3.1    | LatinBERT                                      | 33        |
| 4.3.2    | Transformers                                   | 34        |
| 4.3.3    | ChatGPT  | 35        |
| 4.3.4    | LLaMA  | 35        |
| 4.3.4.1  | Weights  | 35        |
| 4.3.4.2  | Latin Approach Inspired by Alpaca LoRA         | 35        |
| <b>5</b> | <b>Evaluation</b>                              | <b>36</b> |
| 5.1      | Experimental Design                            | 36        |
| 5.2      | Survey Results                                 | 37        |
| <b>6</b> | <b>Discussion, Conclusion, and Future Work</b> | <b>39</b> |
| 6.1      | Discussion                                     | 39        |
| 6.2      | Future Work                                    | 40        |
| 6.3      | Conclusion                                     | 40        |
| <b>A</b> | <b>User Manual</b>                             | <b>45</b> |
| A.1      | Requirements                                   | 45        |
| A.2      | Create Environment and Install Dependencies    | 45        |
| A.3      | Running the Project                            | 46        |
| <b>B</b> | <b>Maintenance Manual</b>                      | <b>48</b> |
| B.1      | Setting-up the Development Environment         | 48        |
| B.1.1    | Requirements                                   | 48        |
| B.1.2    | Running the Project                            | 49        |
| B.2      | Summary of Files                               | 49        |
| B.3      | Direction for Improvement                      | 50        |
| <b>C</b> | <b>Generated Output</b>                        | <b>52</b> |
| C.1      | Survey Text Generated                          | 52        |
| C.2      | LatinBERT Initial Analysis                     | 53        |

# List of Tables

|     |  |    |
|-----|--|----|
| 1.1 | Generated text in the style of Caesar . . . . .  | 11 |
| 3.1 | Breakdown of Sources for the Unified Dataset . . . . .   | 21 |
| 3.2 | LatinBERT Looping in generation with the last fourteen words from the context provided . . . . .                         | 24 |
| 3.3 | Models after being prompted "Instruction: Tell me about alpacas."<br>- Table Source: Alpaca-LoRA's github (52) . . . . . | 29 |
| 4.1 | Data available per author . . . . .  | 32 |
| 5.1 | Results from the Evaluation Survey . . . . .   | 37 |
| B.1 | A Summary of all the Files . . . . .   | 50 |
| C.1 | The generated text included in the Survey . . . . .  | 53 |
| C.2 | Totality of LatinBERT initial generation with the last fourteen context words provided . . . . .                         | 57 |

# List of Figures

|     |   |    |
|-----|---|----|
| 3.1 | Top 25 extant authors in the Dataset . . . . .                              | 23 |
| 3.2 | The Transformer Architecture - Source: Vaswani et al. (49) . . . . .        | 25 |
| 3.3 | First Prompt for ChatGPT . . . . .  | 26 |
| 3.4 | Prompt for ChatGPT to pretend it's a prospective Classics student . . . . . | 27 |
| 3.5 | Final Prompt Template for ChatGPT . . . . .                                 | 27 |
| 3.6 | LoRA in action - Source: Hu et al. (24) . . . . .                           | 28 |
| 4.1 | The code used to prepare the data for LatinBERT . . . . .                   | 33 |
| 4.2 | The code used to generate new Latin text using LatinBERT . . . . .          | 33 |
| 4.3 | The code used to generate new Latin text using the Transformer . . . . .    | 34 |

## Chapter 1

# Introduction

Style is implicit in writing and can reveal a lot of information about its author. Humans are generally capable of discerning style in the writing of one another; this is particularly true for teachers, who know their students' writing intimately and can detect if a student has presented plagiarised work as their own. The task proves to be more difficult for computers, however, since style is subjective and not well-defined. To address this difficulty there are metrics — the number of dependent clauses, vocabulary usage, use of punctuation, etc. — that can be used by computers to try to statistically determine the style of a given author.

This statistical analysis of style in text is referred to as Stylometry (14) and determining the author based on the style of given text is known as authorship attribution. Stylometry and authorship attribution are an active area of research — particularly as it pertains to detecting contract cheating or more recently plagiarism through Large Language Models like ChatGPT<sup>1</sup> (40; 25; 29). In addition to identifying fraudulent work, this research has also been used to try to determine authorship of Shakespeare (26), to help improve spam filters (43), and to estimate emotional intelligence (17).

Whilst identifying style can prove tricky for computers, it is even harder to create new text in a desired style of an author. There has been plenty of work on text style transfer, but this usually refers to the idea of shifting the tone of the text (27).

This paper investigates the effectiveness of transformer-based language models to generate author-styled text in Latin, a low-resource language<sup>2</sup>. It compares the generated output of specifically fine-tuned models with the output of Large Language Models (for the purposes of this paper a Large Language Model is one with billions of parameters)<sup>3</sup>.

## 1.1 Motivation

Despite the recent rapid acceleration and accompanying excitement around Large Language Models and text generation, the research field of author-styled text generation is still surprisingly sparse. Moreover, Latin and other low-resource languages as a whole are more difficult to work with for Natural Language tasks and are consequently less widely studied (34). This work, therefore, contributes both to the research of low-resource languages as well as author-styled text.

---

<sup>1</sup><https://help.openai.com/en/articles/6783457-what-is-chatgpt>

<sup>2</sup>Low-resource languages refer to languages that have less data available and are therefore often studied less than their high-resource counterparts.

<sup>3</sup>It is worth noting that other definitions of Large Language Models will be far below this parameter number, but, given that ChatGPT, one of the models used, has 175 billion parameters (20), a model consisting of  $\approx$  110 million, such as BERT's base model (22), is comparatively not so large.

But why Latin? In the early 20<sup>th</sup> century, prospective Classics students for Oxford University were required to compose text in the style of particular authors (Cicero, Catullus, etc.) as part of their entrance examination<sup>4</sup>. Taking inspiration from the Oxford Latin Composition test, this project aims to similarly generate Latin text in the style of given authors but doing so computationally. An example of generated text in the style of Caesar can be seen in Table 1.1.

## 1.2 Objectives

This project has two objectives. The primary objective is to generate author-styled Latin text using three types of models: a pre-existing Latin model, a newly built and trained Latin model, and Large Language Models. The secondary objective is then to compare the effectiveness of each of these models for this task. Consequently, four supporting objectives naturally arise:

### 1. Obtain a relevant corpus

Whilst this might be the easiest objective listed, given that a trained model can only be as good as the data it was trained on, it is also perhaps the most crucial — particularly, with regard to the decisions on how to preprocess the data.

### 2. Train the models

After obtaining the relevant corpus, the next objective becomes using the dataset to fine-tune a pre-existing model and train a new model on the text of particular authors.

### 3. Generate Text

After the models have been trained, a seed text — i.e. the text that will start all of the generated author-styled texts — must be carefully selected. At which point, the models can generate the continuation of the texts in their attempted authorial style.

### 4. Evaluate the outputs of the different models

After the different models have created text, it remains to evaluate their attempted author-styled outputs, both individually and comparatively.

If these supporting objectives are achieved, then the second objective will have been achieved, yet the primary objective of generating author-styled Latin text will be dependent on the results of the evaluation.

---

<sup>4</sup>This was conveyed to me by Dr. John Behr

<sup>5</sup>Approximate translation provided by Google Translate: <https://translate.google.co.uk/>

| Generated Text Examples           |  |   |
|-----------------------------------|--|---|
| Seed Text                         | Model Generated Caesar   | Translation <sup>5</sup>  |
| Olim sapiens vir nomine Rufo erat | Olim sapiens vir nomine Rufo erat, qui multos annos in Gallia commoratus est. Ille tempus suum in studiis et litteris consumebat, sed postea ad militiam se contulit et ad Caesarem venit. | Once upon a time there was a wise man named Rufus, who lived many years in Gaul. He spent his time in studies and literature, but afterwards he turned to the military and came to Caesar.                            |
| Puer calceos habet.               | Puer calceos habet, non utilem solum sed etiam formosum. Quotidie in via ambulat, gaudens et alacriter procedens. Non pauci obvia mirantur, clamantque "Ecce puer, puer calceatus!"        | The boy has shoes, not only useful but also beautiful. Every day he walks on the road, rejoicing and proceeding with vigor. Not a few are astonished at the sight, and cry out, "Behold a child, a child with shoes!" |

**Table 1.1:** Generated text in the style of Caesar

## Chapter 2

# Background and Related Work

This chapter explains the domain of this particular problem, details some of the literature that will be pertinent for this project, and explains prior work of mine that this project builds upon.

## 2.1 Problem Domain

Before approaching the problem of author-styled generation, given that this project focuses on Latin generation, it is important to first have an understanding of the language and the relevant background information for this problem.

### 2.1.1 Lingua Latina

Historically, Latin was a language spoken and written within the Roman Empire and had primary usage from the 7th century BC through the 18th century AD, where its later usage was primarily for academic texts (7) <sup>1</sup>. Due to the long-running nature of the language, it has naturally evolved over time resulting in eight main versions: Old Latin, that is Latin predating  $\approx 75$  BC; Classical Latin, which spans from the end of Old Latin until the 3rd century AD, was what the educated Romans would have learnt and is mostly the Latin taught within schools today; Vulgar Latin or "common Latin" lasted until the 7th century AD, and refers to Latin as it was usually spoken within the Roman Empire (in contrast to the formality of written Latin); Late Latin is the written counterpart to vulgar Latin after the Classical Latin period; Medieval Latin then evolved from Late Latin during the Medieval period at which point spoken Latin was nonexistent for everyday life (outside of the Church); Renaissance Latin, much like Medieval Latin before it, was the Latin of the time it existed, whose study was emphasised by the humanist movement, and attempted to revert some of the evolutions of Vulgar and Medieval Latin back to Classical Latin; Ecclesiastical Latin began with the introduction of Latin in the Church and in some form or another persists to today, appearing within Church-related documents or services; and Contemporary Latin refers to Latin after the 19th century, for, whilst Latin is a 'dead' language, new written Latin and translations of popular contemporary literature, although not very common, exist (7; 41; 11; 3; 15; 6; 10; 5; 4). The Latin that this project primarily seeks to investigate is Classical Latin because the pre-eminent authors that the Oxford Composition Test — and therefore this project — focus on are from that age of Latin.

---

<sup>1</sup>This ignores Latin within the Catholic Church, which effectively persisted until Vatican II (20th century).

### 2.1.2 Latin Grammar

I will now provide a brief overview of the grammar of Latin, particularly within the Classical Latin time period.

Because Latin is an inflected language (7), the forms of words convey their meaning within a sentence. For instance, the ending of a noun will indicate its case and its number, where the case can be one of six cases that determine its usage (e.g. subject, direct object, direct address, etc.) and the number can be one of two (singular or plural). Nouns also have genders that are intrinsic to the noun itself. Adjectives must then follow the same functional form of the noun that it agrees with (i.e. an adjective agreeing with a singular neuter subject will also be singular neuter and in the same case as the noun) (16).

Verbs similarly have conjugated forms that explicitly make clear their information (e.g. plurality, tense, mood, etc.) (16).

Although Latin often follows subject-object-verb word order, as a consequence of its inflected nature, the word order is not as strict as other languages, but a difference in word order can be used to place emphasis (16).

### 2.1.3 Style in Latin

Since the project aims to simulate the style of particular authors, it makes sense to briefly overview some stylistic features in Latin texts.

In Latin as in English, there are a variety of techniques employable by the omission or repetition of words whether that is through asyndeton and polysyndeton (omitting or repeated use of conjunctions to affect the speed of the text), through repetition of words or phrases in sentences (or lines in poetry) such as anaphora, or through repetition of letters for alliteration (53; P). Unlike English, however, because Latin has declinable words, there are also stylistic flairs that rely on syntactical patterns — whether through repetition of syntax, as in parallelism, or through intentional inversion of it, as in chiasmus (53).

Moreover, counting syntactic features for authorship attribution is as effective for Latin authors as it is for English authors. In Clarence Linton Meader's 1905 work *Types of Sentence Structure in Latin Prose Writers*, he explains that mechanical analysis of the formation of sentences in terms of their clauses and other syntactic features can prove very effective at determining Latin writers of rhetoric — he mentions Cicero, in particular (35). The style of writing in prose will also often be determined by the subject matter at hand; Nutting (37) reviewing J. Arbuthnot Nairn's *Latin Prose Composition*, specifies that the "chief divisions of Latin prose style ... are (a) Oratory, (b) Narrative, (c) Letters, (d) Philosophy, [and] (e) Criticism of Arts and Letters".

Conversely, whilst subject matter might play a role in determining authorship of poetry, the rhythmic meter, determined by the order of words in a line and the line's resulting syllabic patterns, plays a larger role in shaping its style (12).

### 2.1.4 Latin Composition

When beginning to speak of Latin Composition, it is important to state precisely what form of composition is intended. Nutting (37) states that J. Arbuthnot Nairn considers three main types of composition: translating Latin into another language and then back into Latin, translating English into Latin, and creating original text. As the Oxford Latin Composition test required composing

in the style of particular authors, for this project we will be focusing on creating original author-styled text rather than performing style transfer (the act of maintaining the original meaning behind text but changing its style) (47).

## 2.2 Related Work

There has been research in the area of generating text that accounts for style and authorial style specifically, which this project seeks to build upon specifically for Latin authors.

**Stylish Text Generation** Generating poetry computationally is not a novel idea. In fact, according to Gonçalo Oliveira (23), it can date back to the 1960s, yet, given the challenging nature of poetry specifically, the challenge remains popular within the Natural Language Research community. Generating stylistic poetry, however, is an even more complicated task as it can require expert labeling (54). In Yang et al. (54), the authors specifically aim to generate poetry in specific styles in an unsupervised manner, eliminating the need for expert labeling or development of rules or templates, and they believe they were the first to do so. They accomplished their generation through a sequence-to-sequence model with an attention mechanism and then the model attempts to maximise mutual information, a concept from information theory to measure the dependency of random variables (54).

In Lin et al. (32), the authors found it was possible to imitate the style of text by treating an example of text, whose style you wish to imitate, as a "soft" template, and then they were able to work new content into this "soft" template to maintain both the original style whilst incorporating new content. Although the authors did not make any mention of or investigate capturing authorial style explicitly, it may be possible to perform this content transfer with authorial style by using the author's texts, although, currently, this approach would limit the length of text generated.

**Author-Stylized Text** There has, however, been work specifically investigating the generation of author-styled text. In Liu et al. (33), the authors investigate the problem of article headline generation and propose a new approach for headline generation that takes into account an author's historical headlines to create author-styled headlines. They create a transformer based sequence-to-sequence model that uses contrastive learning, which contrasts articles from the same author against articles from a different author to learn (31) what discriminates the two, and they also takes into account a style vector, which is an averaged vector of all the historical headlines of an author, using vector fusion methods.

In Tikhonov and Yamshchikov (46), the authors employ a long short-term memory (LSTM) network that utilise phonetic and semantic embeddings to generate Russian and English poetry in the style of particular authors. They also propose a new cross-entropy metric to evaluate the stylistic similarity of the generated poems and the author.

The approach of fine-tuning pre-existing language models on the desired author's texts with the aim of capturing authorial style, as outlined in 1.2, resembles work by Syed et al. and thereby also Singh et al., as they build on the former's work. In Syed et al. (44), the authors propose StyleLM, a language model built by pre-training a transformer on a corpus of text in the author's language (in their case English) and then fine-tuning with denoising autoencoder loss, to generate author-styled text without parallel data for that author. In Singh et al. (42), which shares an author with the former paper, they build upon the former paper's work by introducing a new framework:

"DRAG: Directing a Generator for Stylized Rewriting", and this framework has the same stages as the paper by Syed et al. but with the added final stage of using a novel framework, inspired by reinforcement learning, that they introduce to further fine-tune the language model to yield better results.

## 2.3 Background

In this section, I review the relevant groundwork upon which the rest of the research project is built. In particular, this section details different language models that will be used, namely: Transformers, the most foundational model whence all the subsequent models are based upon (49); Bidirectional Encoder Representations from Transformers or BERT (22); and Large Language Models — specifically, ChatGPT<sup>2</sup> and LLAMA (48).

### 2.3.1 Transformers

It is far from an overstatement to claim that the introduction of transformers in Vaswani et al. (49) is one of the most significant developments in the recent history of natural language processing. Like most state-of-the-art model architectures prior to transformers, transformers utilise an encoder-decoder network. The primary difference, however, with a transformer is that it only employs attention, rather than the historic approach of using recurrences or convolutions, and, in particular, the use of self-attention, where each encoded token knows its position and emits both a key and a query. The query emitted dictates what prior keys the token is looking for and the relevant attention is then determined through a scaled dot-product of all the prior keys and this query, where the prior keys and tokens of a high compatibility will yield a larger value, thereby contributing more information into this token's position (49; 30). Additionally, because of the lack of recurrences as utilised by prior sequence-based language models, the transformer was more parallelisable, resulting in faster training times and out-performed other models with new state-of-the-art scores for translation.

### 2.3.2 BERT

Researchers at Google, about a year after the advent of the transformer, released another landmark natural language processing paper in (22), where they introduced Bidirectional Encoder Representations from Transformers (BERT). If the most noteworthy feature of the transformer was its self-attention, then for BERT it would be its bidirectionality. BERT being bidirectional means that, unlike prior language models, the information from the text was understood in both directions (left-to-right and right-to-left), and it does this through its model architecture, a multi-layer bidirectional transformer based on transformers as above in 2.3.1. In order to pre-train BERT, the researchers utilised masked language modelling, wherein the model randomly masks out words in its input and the goal then becomes to predict the masked words. Once BERT is pre-trained, it can then be fine-tuned for various natural language processing tasks. For instance, in the paper, they showed that fine-tuning BERT could outperform prior systems for the Stanford Question Answering Dataset, where the objective is to answer a question based on a passage provided that contains the answer (22).

---

<sup>2</sup><https://openai.com/blog/chatgpt>

### 2.3.2.1 Latin BERT

BERT was first released as two models (one solely for English and a general multilingual model) (22; 18), but, in following years, researchers have trained BERT models for a variety of languages, eventually including Latin. In Bamman and Burns (18), the authors presented Latin BERT, a contextual language model for Latin. The model was trained on a dataset comprised of 6 sources, spanning from Classical Latin into Contemporary Latin with translations of contemporary work. They show 4 main natural language processing case studies for Latin BERT: part-of-speech tagging, text infilling, word disambiguation, and contextual nearest neighbors.

For the part-of-speech tagging case study no further fine-tuning was actually required. Instead, since ‘cum’ can be in different parts of speech according to context, they encoded 100 sentences with the word Latin word ‘cum’ as a preposition and another 100 as a conjunction and then withdrew the representation of ‘cum’ from each sentence. Afterwards, they found that, if graphed, the vector representations of ‘cum’ were nearly perfectly separated dependent on its usage (18).

Given the pre-training method of BERT was through masked language modelling as described in 2.3.2, Latin BERT is well-suited to the second case study, text infilling, which involves predicting a word that has been removed from context (18). By taking cases where text is missing and humans have determined the most likely word missing and removing them from Latin BERT’s training, the authors were able to test Latin BERT’s ability to predict the human-determined word given context. Retrieving Latin BERT’s top 50 predictions for this test data, the authors discovered that its top guess was the human-determined word 33.1% of the time, the human-determined word was in the top 10 predicted words 62.2% of the time, and in the top 50 predictions 74.04% of the time (18).

The authors also attempted the more challenging task of word disambiguation, which involves determining the appropriate relevant meaning of a word in context (18). The authors mention two examples to make the notion clearer: ‘est’ in Latin may either mean ‘he/she/it eats’ or ‘he/she/it is’, and another example, which does not rely on the form of the word, is the Latin word ‘vir’ can mean ‘man’ but it can also contextually mean ‘husband’. For this problem, they create a new dataset for evaluating a model’s performance on word disambiguation in Latin and they find that Latin BERT, fine-tuned for this task, results with an accuracy of 75.4%, where a random choice would have resulted with 50%.

The authors’ last case study provides insight into finding similar texts to a provided one, particularly finding similarity of specific instances of words in context (18). This would enable further work to be done for intertextuality, discovering relationships between similar but distinct texts. The authors proposed way of accomplishing this task is by encoding representations of a Latin text dataset — in their example, they used a subset of the Classical Latin texts — and then finding the nearest neighbours of a word’s representation within a particular sentence.

### 2.3.3 Large Language Models

The base BERT language model had 110 million parameters and the large BERT model had 304 million, which, whilst quite sizable — particularly for the time of its release — is orders of magnitude smaller than the models of this section. In particular, the models reviewed are Generative pre-trained transformers (GPT) 3 (20), whose series of models serves as the basis for ChatGPT — the model used in this paper, and LLaMA (48).

### 2.3.3.1 GPT-3

The methodology behind OpenAI's GPT framework, based, as evidenced by its name, on transformers as seen in 2.3.1, is rather straightforward and effective. The approach is to train a model with both a large parameter size and a large amount of data, as the authors show that increasing the size of the model results with better in-context learning (20). Thus, the researchers put forward GPT-3, a 175 billion parameter language model that was trained on 410 billion tokens of text data from the CommonCrawl, 19 billion from WebText2, 12 billion from Books1, 55 billion from Books2, and 3 billion from Wikipedia (20). One of the fascinating things about Large Language Models as Brown et al. (20) shows are its varying levels of effectiveness without the need for fine-tuning under three different approaches: few-shot, one-shot, and zero-shot.

Before explaining how the other methodologies work, it is worth noting that fine-tuning for large language models like GPT is in no practical way different than for standard models, where a provided model has its weights updated by further training on a specific task-related dataset (20). As always with fine-tuning, the usual drawbacks and advantages stand. Most notably, the drawbacks are the increased likelihood of overfitting resulting in generalisation and the need for a dataset for every new particular task (20), and the main advantage is, of course, potentially improved performance for the task at hand. The authors did not choose to fine-tune in Brown et al. (20), but they noted it might be worth investigating in the future.

Instead of updating the weights through fine-tuning, however, the few-shot approach allows the model to see a few examples of what is expected during inference (20). That is the model is provided with examples (typically between 10 and 100) of context and completion for a desired task — for instance, if the task is translating English to French, the examples are an English word, its context, followed by its French translation, its completion — where the last example referred to as the prompt only has context, allowing the model to complete what it believes the result should be contextually (20). This approach has a clear advantage over the fine-tuning one discussed above in that, although some examples are requisite, there is no longer a need for an entire task-specific dataset for a given task, but the model under performs relative to state-of-the-art fine-tuned models for the task(20).

The one-shot approach works in a similar manner as the few-shot, but, where the few-shot uses multiple examples, the one-shot is provided a description of the task at hand followed by one example of the task (20). The zero-shot approach analogously follows, where no examples are provided but the model is given the description of the task with a prompt (20).

As the title of the paper, "Language Models are Few-Shot Learners", suggests the authors evaluate GPT-3 using few-shot, one-shot, and zero-shot settings, showing that GPT-3 performs particularly well in natural language processing tasks in a few-shot setting and they conjecture that very large language models may continue to be the key forward for general natural language systems (20).

**ChatGPT** Although OpenAI did not originally investigate fine-tuning GPT-3, as mentioned above, in late 2022, the company released ChatGPT<sup>3</sup>, which is a website that allows interaction with their fine-tuned model of GPT-3.5, the successor to GPT-3. ChatGPT is a natural language based assistant, which was optimised through Reinforcement Learning with Human Feedback. An important

---

<sup>3</sup><https://help.openai.com/en/articles/6783457-what-is-chatgpt>

note is that, although the website is accessible to the public, the data and model for GPT-3.5 is entirely private and proprietary, which makes external research validation and assessment of the model more difficult.

### 2.3.3.2 LLaMA

In stark contrast to the recent private and proprietary GPT models, Meta AI released a series of state-of-the-art Language Models called LLaMA, ranging from 7 billion parameters to 65 billion parameters, which were trained on public data (48). Additionally, all the model weights were also accessible to the research community (48). The authors also found that the LLaMA model with 13 billion parameters outperform GPT-3 in zero-shot and few-shot tasks, as detailed above, which is indicative of its improved efficiency since it's over 10 times smaller than GPT-3 (48).

## 2.4 Prior Work: Latin Authorship

Before proceeding now with the rest of the project, it is important to acknowledge the prior work in this domain that I have undertaken. As part of completion of CS4040 Research Methods, I investigated the subject of authorship attribution in Latin. The motivating factor behind the research was as a precursor to this project. Before generating text in a particular authorial style, it seemed worth verifying that the different author-styled text could be separated, as it implies that the styles implicit within the texts were unique (19).

### 2.4.1 The Dataset

**Creation** The primary difficulty was creating a unified dataset similar in design to the dataset in Latin BERT, 2.3.2.1. The dataset was created using the Classical Languages Toolkit (28), a Python library, and consisted of texts from The Latin Library, Italian Poets in Latin, and the CLTK Tesserae Latin Corpus. The two main difficulties in developing this dataset was that each of these corpora had different formats and that each distinct text should only be included once. For the time being, the texts were made distinct by only including texts of an author if texts from that author had not been seen from a prior source, but, in the future, the Levenshtein distance can be used to determine the similarity between texts to only save distinct texts (19; 9).

**Preprocessing** The preprocessing of text for the dataset in Behr (19) included:

- Making all the characters unicode. This procedure was included because there were Greek letters in some of the text, which were transliterated, and the models were meant to only use Latin characters.
- Making all words lower-case. Across the texts from the different sources, there was an inconsistent use of capitalisation, so the texts were standardised by making all the text lower-case, which does not remove any of the meaning of the texts.
- Removing all numbers. There were instances of numbers for both line and verse numbers, which were unnecessary.
- Removing all the special characters. The removal of special characters like tabs and newline allowed for the all the text to be in one cohesive plain-text block.

### **2.4.2 Results**

The main results of this research relevant to this project were that the texts were indeed separable by author, which gave credence to the idea that there was implicit style in the text, and the development of the dataset, which is used in this project.

## Chapter 3

# Design & Methodology

As this project revolves around experimentation rather than developing a system, this chapter introduces and provides justification for the design choices and approaches for the different models and the requirements for this project.

### 3.1 Requirements

The requirements are split into functional and non-functional requirements of the project's finished code.

#### 3.1.1 Functional Requirements

The functional requirements for the models in the project follow from the primary objective outlined in 1.2. Namely, the primary functional requirement is for the model to be able to generate text in the style of a provided Latin author, but, just as the objective had sub-objectives, the functional sub-requirements for this primary requirement follow equivalently for all the models, excluding ChatGPT.

**Extracting specific author subsets** : It will be necessary to interact with the dataset to retrieve text from the specific authors so the models can be trained.

**Training** : It is naturally crucial to be able to define and train the models (where applicable), but, even further, it is important that the code should be easily adaptable for other authors in the dataset.

**New Inference** : After training the models, another requirement is being able to load the models again and generate new text using these models. Moreover, the models must be able to generate new text as a continuation of some seed text — i.e. a starting point or basis — provided.

#### 3.1.2 Nonfunctional Requirements

##### 3.1.2.1 Code Quality

As with any project that involves software development, the quality of the code is paramount. There are many ways to determine the quality of the code, many different standards one could go by, different development methodologies, but, in all cases, the outcome should result in code that is readable and understandable to other developers. To ensure that the code maintains a high standard, everything should be well-documented, the code should include necessary comments, and the git will be regularly used to maintain a version history of the project.

Moreover, the research community has largely adopted the use of Python Jupyter notebooks<sup>1</sup>, which will also be employed. These notebooks allow for a combination of documentation and

---

<sup>1</sup><https://jupyter.org/>

| Data available             |                      |
|----------------------------|----------------------|
| Name of Dataset            | Size of Dataset (MB) |
| The Latin Library          | 96.3                 |
| Italian Poets in Latin     | 1.1                  |
| CLTK Tesserae Latin Corpus | 53.9                 |
| Raw Total                  | 151.3                |
| Final Preprocessed Total   | 60.7                 |

**Table 3.1:** Breakdown of Sources for the Unified Dataset

code snippets, so the justification and explanation of chunks of code along with references can be shown. Using these notebooks makes it easier for subsequent researchers to replicate the results of the project.

## 3.2 Design Prerequisites

### 3.2.1 Dataset

As mentioned previously, this project contributes to the research of low-resource languages. It is therefore important to be explicit in the resources (the data) available for this project, since it is both potentially one of the limiting factors but also functions as one of the interesting aspects of the project and provides insight to the models' effectiveness. Clearly, the dataset is a prerequisite to train the models, which, outside of ChatGPT, is done for all of them. Another prerequisite for the data is a single corpus interface. It would be useful to have an interface that has access to all the data from across the different sources in the preprocessed format described in 2.4.1, as it would enable easier use when developing and training the models.

**Data Available** As stated in 2.4.1, the dataset used in this project pooled data from three primary sources: The Latin Library, Italian Poets in Latin, and CLTK Tesserae Latin Corpus. The amount of space in megabytes of these resources and their resultant combination into one unified dataset (with the preprocessing as described above) can be seen in Table 3.1. Additionally, it is worth noting that, since this project looks at Classical Latin text, the Italian Poets in Latin will be of less use as it does not contribute to our final author subsets. To put the size of resources available into perspective, in Tikhonov and Yamshchikov (46), the authors were focusing on author-styled poetry generation in English and Russian, and those respective datasets were 150 and 140 megabytes. Thus, this project's dataset of both prose and poetry in the Classical Latin time period is therefore approximately 40% the size of their solely poetry-focused datasets.

**Corpus Interface** The guiding motivation behind the corpus interface is both to abstract the access to the project's dataset with parameters dictating how it should be accessed, which allows the interface to be referenced with the parameters in one line of code rather than repetition of many lines throughout the project, and to be able to have auxiliary functionalities on the data that are useful for other components (e.g., a function to associate every author to a unique colour so that plotting is easier). Consequently, the corpus interface is a class that has a dictionary of all the authors to their works and then has associated functions that make accessing particular works and information easier.

### 3.2.2 Author Selection

Although the choice of authors for this project does not change the function of the models, it is clearly a prerequisite before proceeding with generation. A number of considerations were taken when determining the most suitable authors for this task.

Firstly, the amount of text available for the author must be a consideration, as, if there is very little text available for a given author, then fine-tuning or training the models will likely overfit to the point of not generalising at all. Figure 3.1 visualises the authors with the 25 most extant texts within the dataset. The author with the most text written is Cicero, one of the most famous Latin authors, but, interestingly, the second most prolific author after Cicero is Jerome, who lived in the 4<sup>th</sup> century AD (outside the the time period this project is concerned with) and translated the New Testament into Latin. Similarly, Bede, who appears towards the end of Figure 3.1, was not a Classical Latin writer, so he too is removed from consideration.

Secondly, as this project concerns replicating style, it makes sense to try to pick authors that have a clearly identifiable style within the community of Latinists<sup>2</sup>. On the r/Latin subreddit<sup>3</sup>, someone asked "Which classical author has the best style in your opinion? In what way?" and the consensus seemed to be that for poetry Ovid and Vergil have the best style and for prose Cicero, Seneca, and Plautus (13). This view is corroborated by the recurring names of these authors across the internet and in Clarence Linton Meader's work, as mentioned in 2.1.3 (35). On the Latin literature wikipedia page, when mentioning examples of Roman authors, they reference Cicero, Vergil, Ovid, and Horace (8), and, on a wikipedia page devoted to study of Latin poetry, the poets, whose work is used as examples throughout the page, are Catullus, Horace, Virgil and Ovid (12). Whilst none of these references are claiming these authors to have the best style, their frequent appearance shows the prominence of these authors, which is something to take into consideration when selecting the authors.

Lastly, given the initial inspiration behind this project was the Oxford Latin Composition entrance exam, it seems fitting to take into account the authors that are still relevant for student who study Classics and are preparing for university. In the United States, there is a Latin exam for high school students called the Advanced Placement (AP) Latin Exam, which gives students college credit (enabling them to have credit equivalent to Latin 1 once they enter university) if they earn a high enough grade on the exam. This exam focuses on Caesar and Vergil.

Taking into account how much text is available, the style of the author, and the initial inspiration for this project, the project will focus on replicating the style of Cicero, Caesar, Vergil, and Ovid. It also enables us to investigate the replication of style in prose (Cicero and Caesar) versus poetry (Vergil and Ovid).

## 3.3 Latin BERT

When determining the design or the approach of how to utilise LatinBERT for this project, it seemed reasonable to investigate what LatinBERT was capable of generating by default and whether it would continue writing in the style of an author if it was provided starting text in that author's style. A concern was that, since it was trained on this data, it may just continue the

---

<sup>2</sup>A Latin specialist

<sup>3</sup>A subreddit being a particular community on the online discussion website reddit

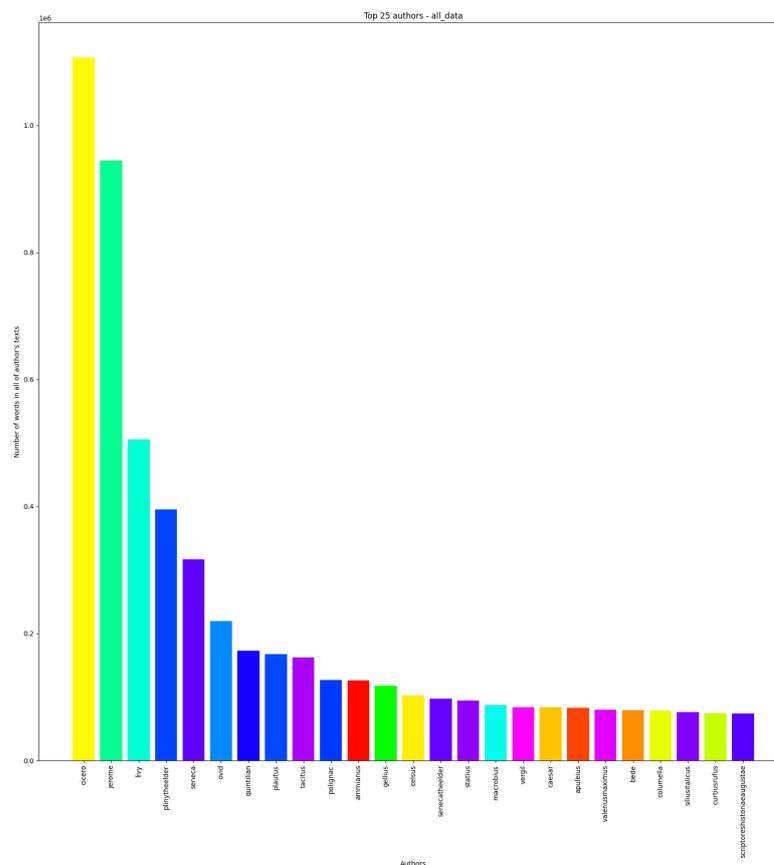


Figure 3.1: Top 25 extant authors in the Dataset

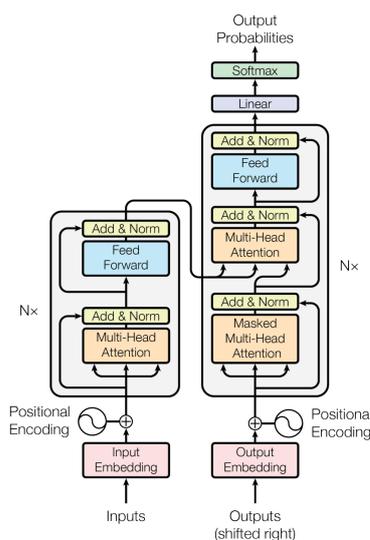
original text verbatim.

Before evaluating this question however, it should be noted that, whilst Latin text is what LatinBERT is designed to work with, it naturally can not directly interact with text, so the input text must be represented or rather encoded before given to the model. LatinBERT comes with its own tokenizer, however, to address this issue, which is used in all the LatinBERT approaches. Additionally, there are constraints on the length of text that any model can encode and generate within one context or chunk of text, and LatinBERT allows for a maximum of 512 encoded tokens (18).

Knowing how LatinBERT encodes the text and its maximum length of encodings, allows us to examine its default generation ability. I randomly sampled 49 contiguous word chunks of size 200 from different Latin texts spanning a few different authors and then allowed LatinBERT to attempt to generate the continuation for each of these chunks. It's worth noting that the generation performed was functionally the same as text-infilling, as described in 2.3.2.1, but, instead of predicting a word within context, it is predicting a word at the end of context. Afterwards, I compared the generated continuation and the actual continuation of the text. In table 3.2, I show a few of the results from this sampling approach where the first fourteen words are provided;<sup>4</sup> it is clear that

<sup>4</sup>The other results are similarly quite poor. I include all 49 of these generated texts in C.2.





**Figure 3.2:** The Transformer Architecture - Source: Vaswani et al. (49)

### 3.4 Transformer

In addition to the pre-built models employed, I decided to include a newly trained transformer on the Latin author subsets. Although the architecture of the transformer does not deviate from the original paper by Vaswani et al. (49), which can be seen in Figure 3.2, the input to the transformer needs to be determined. In LatinBERT, there was a tokenizer that would encode the text to be operated on by the model, but, when defining our own transformer, there are various ways to encode the text and it may affect the performance of the model.

Although Brown et al. (20) use byte-level byte-pair encodings that use less space than most character-based approaches before it (51), for the sake of simplicity a character-level is employed here. A character-level encoding works by having each unique character correspond to a unique number and then all the input text is encoding by replacing each character with their respective numbers. Similarly to decode the numbers, every number in the encoded text is then replaced with its associated character. For instance, if one used American Standard Code for Information Interchange (ASCII) character encodings, the text "hello" would be "104 101 108 108 111", but, due to the preprocessing in this project, the number of characters required is significantly smaller than the number of available ASCII encodings. The transformer then has an internal table that maps every possible character to a vector of a customisable size, which is how the input embeddings in Figure 3.2 are created.

The transformers add these input embeddings to their positional encodings, which comes from a table that maps the location of the text — out of a custom possible maximum length, which for this project was somewhat arbitrarily chosen to be 256 — to an embedding vector of the same size as the input embeddings. Afterwards, the model passes these vectors through the attention mechanisms that characterise transformers as described in 2.3.1.

Having determined the design of the transformer, the design and methodology for training it to perform author-styled text generation is the logical next step. In Singh et al. (42) and Syed et al. (44), the authors first either acquire pre-trained transformers or pre-train them themselves, so

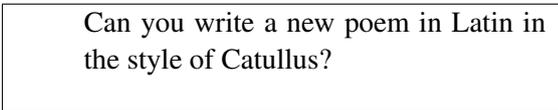
the transformers learn about the language, and then the authors fine-tune these pre-trained transformers on particular author subsets with the aim of teaching the transformers their authorial style. For this project, this approach would mean training our designed transformer on the Latin dataset and then fine-tuning on our specified authors. Given that this is how author-styled generation has historically been approached, this project will follow this methodology, but we will also observe what happens when the pre-training stage is eliminated from this process – i.e. only training on a transformer on particular authors.

In Singh et al. (42) and Syed et al. (44), it is unclear whether, during the pre-training phase, the authors removed the subset of the author whose style they wish to mimic, so that it would be unseen in the fine-tuning phase. At first I opted not to remove the particular author from the dataset during the pre-training phase for this project because I was uncertain how much data the transformer needed to learn about Latin generally, as is the goal during the pre-training. Moreover, the already somewhat limited data is split into 80% for training and 20% for validation, and by removing large authorial subsets may cause the data to become too limited. Furthermore, as LatinBERT was fine-tuned on particular author subsets despite having been pre-trained with these subsets, removing these author subsets from the dataset for my Latin transformer's pre-training phase would be potentially handicapping it needlessly.

### 3.5 ChatGPT

Although this project undertakes no fine-tuning or development of ChatGPT, it is still critical to explain the design and methodology for how ChatGPT is used. As stated in 2.3.3.1, ChatGPT is a website that provides an easy interface for the end-user to leverage a version of the language model GPT 3.5 (and for some users GPT-4) fine-tuned to interact with the user in a helpful manner akin to an assistant.

**Initial Approach** Just as with LatinBERT, trying the simplest approach first is logical as it establishes a precursory baseline for subsequent iterations to try and improve upon and because, if it works as intended, no further work is required. Thus, the first prompt tried for ChatGPT can be seen in Figure 3.3. ChatGPT replied helpfully by asserting that it could and then generated a poem, which it claimed was a new poem in the style of Catullus as asked, but, after examining the text generated, it became apparent that the text was actually a poem by Horace, a poet of the same era.



Can you write a new poem in Latin in the style of Catullus?

**Figure 3.3:** First Prompt for ChatGPT

**Further Prompt Engineering** It became clear after the initial experimentation that a different approach would be required with more intricate prompt engineering. Prompt engineering refers to the act of writing the task description, as seen described in the different "shot" approaches in 2.3.3.1, for language models to understand what is required and perform the task correctly (1).

I then pedantically explained the concept of the Oxford University Latin Composition exam, the inspiration for this project, and ask ChatGPT to pretend they are a prospective Classics student,

who is tasked with writing new Latin text in the style of the author I specify. The full prompt can be seen in Figure 3.4. ChatGPT again helpfully asserts that it can accomplish this task, but, after prompting it write in the style of Caesar, it responded with the start of Caesar's work "Commentarii de Bello Gallico". It was successful in identifying the correct author, but it failed in generating new text.

Let's play a game where you're pretending to be a prospective Classics student at Oxford University and, as such, you are instructed with writing new Latin text in the style of particular authors determined by a prompt following the form:  
 Author: X, where X is the author I want you to replicate.  
 You then respond with a proposed title and the body of the text you are writing in the form:  
 Proposed Title: X, where X is the proposed Title  
 Text: Y, where Y is the text for the piece.  
 If I ask any other questions not in the form of that prompt, you can answer me accordingly.

**Figure 3.4:** Prompt for ChatGPT to pretend it's a prospective Classics student

After I informed ChatGPT that it had provided me with existing Latin text, it attempted to correct this error by summarising Caesar's "Commentarii de Bello Gallico" in Latin instead of creating new text. After correcting this error, however, ChatGPT was able to create unseen Latin text, which, according to ChatGPT, was ostensibly in the style of Caesar.

I want to modify the game where after I say the author. I also say the starting phrase you must begin the new work with.  
 E.g.  
 Author: X  
 Prompt: Y  
  
 and then you respond with  
 Text: Y Z, where Y is the prompt I provided and Z is the continuation

**Figure 3.5:** Final Prompt Template for ChatGPT

Establishing this context for ChatGPT allowed for a new prompt, which can be seen in Figure 3.5, that enabled me to provide starting text that ChatGPT use as a basis to generate from in the style of the provided author, and this template is what was used for the final text generation in the other authors' styles.

## 3.6 LLaMA

It's great for the research community that Meta has trained and released LLaMA, but using LLaMA poses a couple of design and methodology challenges for this project. Firstly, on a conceptual level how can LLaMA be used to generate Latin author-styled text? Use of the "shot" learning approaches is difficult because it might require example author-styled composition texts for generation, and, similarly, how can LLaMA be fine-tuned to generate text that does not exist yet? Secondly, even if a dataset is found or created that allows LLaMA to be fine-tuned for this task, fine-tuning a large language model is very computationally intensive.

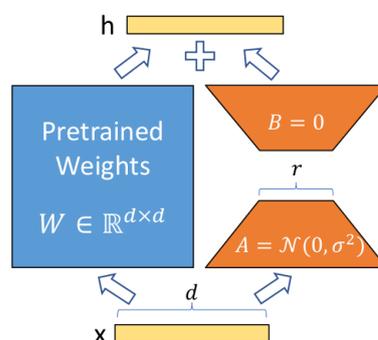
### 3.6.1 Alpaca

Researchers at Stanford University fine-tuned LLaMA (the 7 billion parameter version) to create an instruction-following model and dataset that can be studied in academia (45). In order to create a relevant dataset for LLaMA, the researchers employed a technique called the "self-instruction method", where they first used a dataset of 175 human-created instruction-output pairs and used OpenAI's text-davinci-003<sup>5</sup> to create more instructions with the original human-created ones as examples (45). They created 52 thousand instruction-following examples through this methodology, which is accessible to everyone (45).

After they created their dataset, all that remained was to fine-tune the LLaMA model on these weights and evaluate the results. They were able to fine-tune LLaMA (7 billion parameter version) within 3 hours, using eight 80 Gigabyte A100 GPUs, and the authors estimate that the fine-tuning would cost "less than \$100 on most cloud computing providers" (45). To evaluate the model's performance, they performed a blind pairwise test comparing Alpaca and text-davinci-003, and the results were shockingly slightly in Alpaca's favour (45) despite having fewer model parameters, less training time, and a rather straightforward approach to the generating the data for fine-tuning. They noted that more broadly Alpaca behaved quite similarly to text-davinci-003 over a range of inputs (45), but they also mention the limitations that Alpaca faces. Most notably, Alpaca like other language models is prone to hallucinating information and thereby capable also of spreading misinformation (45).

### 3.6.2 LoRA

Although fine-tuning LLaMA for less than \$100 to the point that the model functions similarly to text-davinci-003 is impressive, it required capital or resources that are not necessarily obtainable (at least for the scope of this project), as mentioned at the start of this section. Thankfully, there is a way to overcome this hardware requirement barrier. Instead of performing backpropagation across all the trainable parameters when fine-tuning a large language model, what if it was instead possible to only train on select specific layers? This is part of the idea behind Low-Rank Adaption (LoRA), introduced in Hu et al. (24). In fact, LoRA can accomplish this whilst retaining the original weights of the models.



**Figure 3.6:** LoRA in action - Source: Hu et al. (24)

<sup>5</sup>This model uses GPT-3.5 turbo, according to <https://platform.openai.com/docs/models/gpt-3-5>

LoRA freezes the model's weights during training and instead adds adaptable trainable layers, so that the model processes an input both in its regular layer and its trainable layer summing the result before passing it to the next layer. Then, during optimisation only the adapter module weights are updated (24). When one wants to use the original model, nothing needs to be done, and, when one wants to use the fine-tuned model, the adapter weights are loaded with the model. Figure 3.6 shows the process of LoRA initialisation and how the adapter modules work in conjunction with the original hidden weights of the model that remain unchanged afterwards.

### 3.6.3 Alpaca with LoRA

Shortly after the release of Alpaca (45), a technology enthusiast implemented LoRA (24) for the process of recreating Alpaca Wang (52). However, with this implementation, the fine-tuning was accomplished within an hour of training using one RTX 4090, a commercial GPU, and the model qualitatively performs on a comparable level to Alpaca (52). A comparison of outputs can be seen in Table 3.3, which is a verbatim extract from the Alpaca-LoRA github page (52), and, as seen in the table, each model produces comprehensible text despite the vastly different hardware requirements for each of them to run.

| Comparison of Instruction-following Models |  |
|--|--|
| Model                                      | Output   |
| Alpaca-LoRA                                | Alpacas are members of the camelid family and are native to the Andes Mountains of South America. They are known for their soft, luxurious fleece, which is used to make clothing, blankets, and other items. Alpacas are herbivores and graze on grasses and other plants. They are social animals and live in herds of up to 20 individuals.   |
| Stanford Alpaca                            | Alpacas are small, fluffy animals related to camels and llamas. They are native to Peru and Bolivia, and were first domesticated around 5,000 years ago. They are kept mainly for their fine, soft fleece, which is used to make knitwear and other garments. Alpacas are herd animals and live in small family groups, led by an older male. They are highly social animals and can form strong bonds with their owners.  |
| text-davinci-003                           | Alpacas are a domesticated species of South American camelid, similar to llamas. They are native to the Andes Mountains in South America and are kept as livestock for their soft, luxurious wool. Alpacas are social animals and live in herds of up to 20 individuals. They are typically kept in small herds of two to five animals, and are relatively easy to care for. Alpacas are herbivores and feed on grass, hay, and other vegetation. They are also known for their gentle and friendly nature, making them popular as pets. |

**Table 3.3:** Models after being prompted "Instruction: Tell me about alpacas."  
- Table Source: Alpaca-LoRA's github (52)

### 3.6.4 Approach for Latin

The second question posed at the start of this section can be addressed by using LoRA, but the first, which drew into question the correct way to utilise LLaMA for author-styled text generation, remains. Taking inspiration from Stanford's Alpaca, however, I developed a methodology to fine-tune LLaMA for author-styled text generation.

I created a dataset that would allow LLaMA to be fine-tuned on particular authors. Firstly, recall the functional requirement that states the project not only needs to be able to write text in the style of particular authors, but it must be able to continue text from a given starting point. The

---

design of the dataset logically follows: first, sample texts of an arbitrary length (at least 50 words long) from our chosen authors; secondly, take the first fourteen words from the sampled text and add them to the end of the prompt "Write text in the style of <insert author> starting with "; thirdly, pair this instruction with the full sampled the text; and, lastly, after repeating the first three steps many times, save all of these prompt pairs in a JavaScript Object Notation (JSON) file of the same format as the Alpaca dataset, which allows for ease of training through the same code as seen in (52). These instructions capture both the importance of the seed text and the text that continues in the same style of the author.

## Chapter 4

# Implementation

As the last section proposed and justified the designs and methodologies for each of the models, this section will discuss the practicality of implementing these designs and methodologies. The code and data for this project are entirely available on GitHub<sup>1</sup>.

## 4.1 Development

### 4.1.1 Language & Libraries

The programming language used throughout this project is Python, specifically Python version 3.9.16. Python's ease of development and deployment (combined with the many machine learning libraries made for it) make it the clear favourite for this project. Whilst many standard libraries for this type of project were used, the two used most throughout the project are PyTorch (39) and Hugging Face's Transformers Library (2).

**PyTorch** (39) PyTorch is an open-source deep learning library developed by Meta, formerly Facebook, and the developers of PyTorch in its paper explain that PyTorch was designed with both usability and speed in mind, which was uncommon for prior deep learning libraries (39). PyTorch accomplishes usability by maintaining a Pythonic interface and making "the complexity inherent to machine learning ... handled internally by the PyTorch library and hidden behind intuitive APIs" (39). These APIs are mainly made efficient through being based on a highly optimised C++ base, its use of multiprocessing, and its ability to utilise a GPU through CUDA (39).

**Hugging Face's Transformers Library** (2) The Transformers Library, as the name would indicate, works well with transformer models. Particularly, it works well for a variety of pre-defined transformer architectures, including BERT and LLaMA, which are clearly relevant to this project (2). In fact, the library provides a straightforward way to create a dataset from a text file, a data loader for the model to learn, and then a seamless way to train one of these transformer models. Another reason to use the Transformers Library over other libraries is that it integrates well with PyTorch (2), which, as already stated, is one of the main libraries used for this project.

### 4.1.2 Cloud Computing

Although a laptop computer may be sufficient for small scale machine learning experiments, when trying to perform inference — let alone fine-tune — a large language model, a stronger computer will likely be required as described in 3.6. Graphical processing units (GPUs) are useful for larger machine learning projects due to their ability to run parallelised computations and so in this

project there is some minimum hardware requirement for training and running my transformer model, LatinBERT, and LLaMA (the 7 Billion parameter version).

Because I do not have personal access to powerful GPUs, I used a cloud computer provider, specifically for developing and deploying machine learning and artificial intelligence models, called Paperspace<sup>3</sup>. Through a pro plan with Paperspace, I had access to the following GPUS<sup>4</sup> for 6 hour periods: M4000, IPU-POD4, P4000, RTX4000, P5000, RTX5000, and A4000. These GPUS in the timeframe described were sufficient for all of the work in this project.

## 4.2 Data

The Corpus Interface implementation follow exactly as its design in 3.2.1. It was a Python class that can either load in an already preprocessed data file or it will generate a corpus file by loading in all the relevant text, performing its preprocessing operations, and saving the dictionary of authors to their works. There was, however, an unresolved problem with the preprocessing phase for the Corpus Interface. Within the preprocessing phase there was initially a word, sentence, and lemma tokenizer employed from the CLTK library (28), but the inclusion of the lemma tokenizer seems to have resulted with an unwanted token that replaces punctuation with the string ‘punc’.

This error was noticed when the transformer model was generating the ‘punc’ amidst actual Latin words, even though ‘punc’ is not a Latin word. As the transformer works on the character-level, it was possible that the transformer was generating this series of letters intentionally or randomly, but, after intently comparing the original text files with the resultant dataset, I realised that the Corpus Interface erroneously added ‘punc’. To combat this issue, I removed the use of this particular lemmatization tokenizer from the preprocessing process.

| Author | Size of Text Subset (MB) |
|--------|--------------------------|
| Cicero | 1.06                     |
| Caesar | .616                     |
| Vergil | .579                     |
| Ovid   | 1.47                     |

**Table 4.1:** Data available per author

Having established the desired authors in 3.2.2, it is also worth noting the size of the authorial subsets that were used to fine-tune LatinBERT and to train the transformer from scratch, as this is relevant to the notion of training within a low-resource context. The size of the authorial subsets can be seen in Table 4.1.

## 4.3 Models

Although the implementation for the models follows almost directly from their descriptions in the design section, it is worth explaining how the implementation was done so that other researchers are able to recreate the results, and it is important to evaluate whether each model meets the functional requirements outlined in the previous section.

```
dataset = TextDataset(file_path=f"{author}_data.txt", tokenizer=tokenizer, block_size=256)
data_collator = DataCollatorForLanguageModeling(tokenizer=tokenizer, mlm=True,
mlm_probability=0.35)
train_dataloader = DataLoader(dataset, batch_size=batch_size, collate_fn=data_collator)
```

**Figure 4.1:** The code used to prepare the data for LatinBERT

### 4.3.1 LatinBERT

Although LatinBERT is not explicitly defined to work with the Hugging Face Transformers Library as some models are, the library allows LatinBERT and its tokenizer's weights to be loaded into the module `AutoModelForMaskedLM` and `AutoTokenizer` respectively. After the weights have been loaded and the desired author subset has been stored, in order to fine-tune the masked language model (in this case, LatinBERT) we want to a data loader of our desired text. This operation has three sequential components to it: creating the requisite dataset object through the `TextDataset` class where we establish the maximum context size, which will divide the text into corresponding chunks, and, as 256 was the size chosen for the custom Transformer (see above 3.4), 256 was the chosen maximum context size for the chunks for fine-tuning as well<sup>5</sup>: creating a data collator object by passing the dataset as a parameter into the `DataCollatorForLanguageModeling`, which batches the data for masked language modelling using the LatinBERT tokenizer; and, finally, creating the data loader by passing in the dataset and the collator object into the `DataLoader` class from PyTorch, which allows for ease of loading the data samples to fine-tune the model. These steps can be seen concretely in code in Figure 4.1; it's remarkable how these three steps can all be accomplished in three lines of code using the HuggingFace and PyTorch libraries as discussed above.

```
seed_text = "<seed text>".lower()
input_ids = tokenizer.encode(seed_text, return_tensors='pt')
outputs = model.generate(input_ids=input_ids.to(device), max_length=50, do_sample=True)
predicted_text = tokenizer.decode(outputs[0], skip_special_tokens=True)
```

**Figure 4.2:** The code used to generate new Latin text using LatinBERT

The simplicity seen in preparing the data above, as showcasing the powerful combination of PyTorch and Hugging Face Transformers Library, can be seen again with LatinBERT's training process, which follows a standard training loop formulation where for a certain number of epochs the model trains on the batches prepared in the data loader. Generation is also surprisingly simple using these libraries, as seen in 4.2. After either loading a specific model or training, the desired seed text is encoded using the LatinBERT tokenizer and then the HuggingFace library lets the developer immediately generate with the input as a starting point. Afterwards, the text can be decoded back into Latin using the same tokenizer.

After training on each desired author subset independently of each other (i.e. LatinBERT is

<sup>3</sup><https://www.paperspace.com/>

<sup>4</sup><https://docs.paperspace.com/gradient/machines/#free-machines-tier-list>

<sup>5</sup>It should be noted that the two contexts are not equivalent in terms of the amount of Latin they can contain because they encode the text in different ways.

freshly fine-tuned for each author), these newly updated model weights are saved, allowing new generation to be performed by this process described above and seen in Figure 4.2.

Consequently, the LatinBERT model achieves all of the desired functional requirements from 3.1.1.

### 4.3.2 Transformers

The architecture of the character-level transformer design was implemented using PyTorch, following a tutorial by Andrej Karpathy (30). The specified architectural parameters of this transformer implementation include: the number of distinct characters in the author subset, which is the input size for the embeddings look up table mentioned in 3.4; a maximum context size of 256; the size of the embedding layer and the number of heads for self attention in each transformer block are 384 and 6 respectively, since the size of the embedding divided by the number of heads must be an integer (in this case  $384/6 = 64$ ); and, the number of layers, that is the number of transformer blocks, was chosen to be 6. I had experimented with using different parameters including an embedding size of 512 and 8 heads in the transformer blocks, but with the increased size came an increased time for training without clear increased benefit.

The training methodology to capture author-style was followed as described in the design section in 3.4. I first trained this transformer on the entire Latin dataset, and then I fine-tuned this now pre-trained transformer on specific author subsets. However, as mentioned in 4.2, there was an issue with the lemma tokenizer that caused ‘punc’ to be commonplace in the dataset before this tokenizer was removed. Unfortunately, the pre-trained general Latin transformer model trained on the entirety of the dataset was tainted by the presence of ‘punc’, which remained present in further fine-tunings. Thankfully, however, this dataset error was caught before the second methodology (removing the general pre-training stage and training the transformer solely on a particular author’s subset of text) to capture an author’s style was completed.

```
seed_text = "<seed text>".lower()
context_values = encode(seed_text)
context = torch.tensor(context_values, dtype=torch.long, device=device).reshape((
len(context_values),1))
gen =model.generate(context, max_new_tokens=220)[0].tolist()
print(prompt+ decode(gen))
```

**Figure 4.3:** The code used to generate new Latin text using the Transformer

Although this transformer does not make use of the Hugging Face Transformers Library, the dataset preparation, training procedure, and generation stages are all still straightforward. For the dataset preparation, the encoder and decoder are defined as lambda functions that map the unique characters to unique numbers and unique numbers to unique characters respectively. The totality of the author’s subset is encoded accordingly and split into a training set (80% of the subset) and validation set (20% of the subset). During the training phase, the model again follows a fairly standard training loop, where batches from the training set are used for the model to learn. Once the model has been trained on a particular author, the weights are then saved and can be loaded again for creating future text generation or further fine-tuning. The code for generating can be seen in Figure 4.3. In principle, this generation code follows the same format as the LatinBERT

generation code. The first step is to encode the seed text, then pass the encoded text into the model to generate the continuation (which is a function written in the class), and lastly decodes the generated output.

Thus, following the second training methodology (not through any fault of the design of the first), all the functional requirements are met for the Transformer model.

### 4.3.3 ChatGPT

ChatGPT required no further training or implementation beyond the scope of the prompts engineering described in 3.5. The only further consideration is whether to use OpenAI's API to interact with ChatGPT or whether to interact with the website developed by the company. Both options were explored independently, but the results are not dependent on the methodology chosen, so for the sake of usability and ease of replication the project opted to use the website.

ChatGPT passes the functional requirements trivially, as only the primary functional requirement is applicable to it and it meets the requirement by being able to generate new Latin text in what it deems to be the style of a given author.

### 4.3.4 LLaMA

#### 4.3.4.1 Weights

Before being able to do any experimentation with LLaMA, LLaMA's weights are required. I filled out the necessary form requesting access to the LLaMA's model weights. If a researcher wanted to recreate the experiments with LLaMA taken in this project, then one would have to request the weights from Meta themselves<sup>6</sup>. Thankfully, Meta obliged in providing me LLaMA's weights, which allowed me to carry out the proposed methodology from 3.6.4.

#### 4.3.4.2 Latin Approach Inspired by Alpaca LoRA

Recall that for LLaMA the proposed methodology involved mimicking the Alpaca experiment by the Stanford researchers by creating a custom self-instruct dataset, based upon the different author subsets, except, due to hardware constraints, this project would use LoRA to significantly reduce the number of trainable parameters and therefore also the requirements to perform this training. Because the custom-designed self-instruct Latin dataset follows the exact same format as the dataset for Alpaca, the same code for Alpaca-Lora (52) can be run to fine-tune LLaMA (the 7 billion parameter version) but with the datasets swapped.

Unfortunately, the output from this LLaMA model equipped with adapter modules could not generate new Latin text, but rather just repeated the provided seed text back to the user typically without any further text. Thus, the LLaMA model following the methodology designed in 3.6.4 does not meet the minimum functional requirements for this project, and therefore will not contribute text to be included as part of the evaluation.

---

<sup>6</sup>The form can be found here:

[https://docs.google.com/forms/d/e/1FAIpQLSfqNECQnMkycAp2jP4Z9TFX0cGR4uf7b\\_fBxjY\\_OjhJILIKGA/viewform](https://docs.google.com/forms/d/e/1FAIpQLSfqNECQnMkycAp2jP4Z9TFX0cGR4uf7b_fBxjY_OjhJILIKGA/viewform)

## Chapter 5

# Evaluation

With the design and implementation completed, the only objective that remains is to evaluate and compare the output of the selected models that met their functional requirements.

### 5.1 Experimental Design

In authorship attribution, even though it implicitly deals with style by discriminating between texts, the evaluation is entirely objective. The author attributed to the text either was or was not the original author, which leaves no room for ambiguous results. The same can not be said for generating new author-styled text, as there is no defined ground truth to compare the output with. Tikhonov and Yamshchikov evaluated their proposed model's generated author-styled text by creating a survey with two generated pieces of text by each of their four authors as well as two randomly sampled pieces of text by these authors and giving this survey to fluent English speakers to complete (46). The participants of the survey for each of the sixteen generated instances of text could select one of five options (whether it was written by one of the four authors or by a neural network) (46).

Applying their evaluation approach to this project would be challenging due to the text being in Latin. Firstly, the authors that were selected in Tikhonov and Yamshchikov (46) were both well-known to English speakers and stylistically distinct enough that, even if the author were unknown to the participant, they could likely identify the intended author (the authors had selected William Shakespeare, Lewis Carol, Bob Marley, and the band MUSE) (46). Furthermore, the population of English speakers is obviously much larger than the population that can read and write in Latin, let alone the number of people that can identify the style of particular Latin authors. Another issue that arises from this population problem is that a random English speaker may be able to identify the style of an English author without knowing the original work (e.g., an English speaker might identify text that could be by Shakespeare without being able to cite it directly), but the people that have specialised in learning Latin to the point of identifying the stylistic nuances between Latin authors of the same time period will likely know the sampled original author works and therefore can also identify if the text is new, which would mean all the generated text would be correctly identified as being written by a neural network.

Although their evaluation methodology cannot be copied directly due to the reasons above, inspiration can be taken from it to derive a similar evaluation approach. The focus in the methodology should shift from trying to see if the general populace can determine which texts are generated by an author or by a neural network to whether Latinists can identify the intended style in

the model-generated texts, the results of which should also provide comparative insight about how the models perform relative to one another (satisfying the last supporting objective). In order to evaluate the models comparatively and shift the focus accordingly, I created an anonymous survey for Latinists, comprised of shuffled texts generated by the working models. Each working model generates one piece of text per author, resulting in twelve pieces of text overall (since there are four authors and three models), and for each text there are five options available to the participants of the survey: the four authors and an unrecognisable option. The inclusion of the unrecognisable option allows for the surveyed Latinists to select authors only if the style of the text aligns with a particular author rather than, if it were not included, selecting the best option when none of them are very good or applicable. Also, after each text in the survey, there is a non-compulsory writing box for the Latinists to communicate their thoughts on the text above it, as well as a box at the end for any concluding thoughts across the entirety of the survey.

For this project I was able to enlist the help of Latinists to complete this survey. At the time of writing, only two Latinists have completed the survey, but there are several other Latinists that have expressed interest in completing this survey, from whom I am waiting to receive feedback from.

Another important consideration when generating the text for the survey is the role that the seed text plays. The seed text, providing a starting point for the rest of the generation, should be the same for each of the models and for each of the author-styles lest one of the models received text that biases the model favourably to generating in the style of one of the authors. Consequently, the seed text should be nondescript and unlikely to have been seen in any of the training data. I composed the following Latin seed text that falls under both categories: "Olim sapiens vir nomine Rufo erat", which translates to "Once upon a time there was a wise man named Rufus".

All of the text for the survey in the same shuffled order with the text's intended author, model that generated it, and approximate translations, according to Google Translate<sup>1</sup>, can be found in the Appendix C.1.

## 5.2 Survey Results

In this section, the results of the survey described above are tabulated and the comments from the Latinists in the survey are reviewed. In Table 5.1, accuracy has a slightly different connotation than in other settings. As explained, since measurement of captured style is subjective, there is no ground truth per se, so here the accuracy percentage refers to the percentage of times the model's intended author-styled generated text was identified by the Latinists.

| Survey Results |                     |                     |                     |                   |                |
|----------------|---------------------|---------------------|---------------------|-------------------|----------------|
| Model          | Accuracy for Cicero | Accuracy for Caesar | Accuracy for Vergil | Accuracy for Ovid | Total Accuracy |
| Transformer    | 50 %                | 100 %               | 0 %                 | 0 %               | 37.5 %         |
| LatinBERT      | 100 %               | 100 %               | 0 %                 | 0 %               | 50 %           |
| ChatGPT        | 50 %                | 100 %               | 0 %                 | 0%                | 37.5 %         |

**Table 5.1:** Results from the Evaluation Survey

<sup>1</sup><https://translate.google.co.uk>

**Comments from the Survey :** In addition to the above table, there were a few comments that provided further insight into the determinations made by the Latinists. For instance, for ChatGPT's attempted Caesar, the Latinists remarked on the use of the superlative being indicative of Caesar's style, and another remarking that it was "like Caesar's style but its syntax is too simplistic". As will be expanded upon in the next paragraph, it is clear from the table that the models did a poor job of recreating the style of the poets (Vergil and Ovid), but, interestingly, for the transformer's generation of text in the style Vergil the Latinists left somewhat opposing comments. One commented that the vocabulary reminds them of Ovid and there are "dactylic runs" (a term referring to a metrical pattern), whereas the other comment remarked about the similarity to Cicero due to the 'ut' clause. There was also one final concluding comment, remarking that some of these passages were basic and devoid of markers.

**Interpretation of Results :** There are a few conclusions that can be drawn from the survey results. Immediately, it is apparent that the models perform better on the prose (Cicero and Caesar) than on the poetry (Vergil and Ovid), and there are potentially many reasons for this. For LatinBERT and for the Transformer, the models were trained on data that had the newline character removed, and so the generated text, even if it might resemble one of the poet's style, will appear as prose, which was reflected in the comment above — after remarking on the "dactylic runs", the Latinist pointed out that it seemed to be prose rather than poetry. Another explanation for these results are that the Caesar and Cicero simply have more unique and identifiable styles than the poets or from each other. What is interesting, however, is that the struggle of generating new text in the style of poets persists in ChatGPT's generation.

Furthermore, it is interesting that ChatGPT had the same overall accuracy as the newly trained transformer with a very limited dataset and worse accuracy than the fine-tuned LatinBERT model. But given that a stronger divide appears to be along prose versus poetry rather than model against any other particular model, the effectiveness of the models' ability to generate comparatively is somewhat ambiguous. Moreover, since there were only two Latinists so far to evaluate and only one text sample per author was generated, these numbers are not definitive even if they still give some initial credence to the divide of prose versus poetry.

Interestingly, the results do also show that the methodology of training solely on these author subsets — rather than pretraining and then fine-tuning on the authorial subsets — was sufficient to produce text recognisably in the style of prose author using a character-level transformer.

## Chapter 6

# Discussion, Conclusion, and Future Work

### 6.1 Discussion

With the continued emergence and prominence of Large Language Models, natural language generation is a very exciting and active area of research. Despite this surge of interest in language generation, however, the study of generating author-styled text is still fairly barren. This project sought to add to the field of author-styled text generation and to the study of low-resource languages by generating author-styled text in Latin using three different types of models: a newly trained transformer, a fine-tuned pre-trained model (LatinBERT), and Large Language Models (ChatGPT and LLaMA). Many challenges were faced along the way, some more successfully met than others.

**Limitations** : One of the first limitations encountered was in the preparation of the dataset. It is true that Latin is a low-resource language, but the dataset was missing text. In order not to have any duplicate texts, only one author from one source of data was permitted into the unified dataset. A better approach would have been to accept all the works from every author and remove duplicates by finding similar works of the same author using the Levenshtein distance (9).

The dataset not only limited the amount of text available for the project to utilise, but it also adversely affected the transformer directly in two ways. Firstly, as noted in 4.3.2, the inclusion of the tokenizer that converted some of the punctuation to ‘punc’ resulted in generation that included ‘punc’, and, therefore, the first proposed methodology of 3.4 was not pursued. The second way the dataset directly affected the transformer was by removing the new-line character from all the text, which made all the poems effectively prose in the dataset and in generation. Another possible limitation for the transformer was the choice of encoding used. Instead of defaulting to the simplest to understand (each character being a unique number), it may have been worth exploring more alternatives.

An obvious limitation found was the methodology for LLaMA as explained in 4.3.4.2, which effectively meant that LLaMA was unable to generate any Latin text after the provided seed text. Since it was unable to sufficiently generate Latin text, the only Large Language Model that was included for the comparison of the generated text was ChatGPT.

ChatGPT may also have been limited by my prompts. Given that the quality of the text it generates can differ greatly based on the quality of the prompt written, it is possible that my prompts were hindering it from producing better quality author-styled text.

The largest limiting factor in this research project, however, was undoubtedly the number of

competent Latinists available to evaluate the generated text. Even though the first survey responses are indicative of the divide between poetry and prose, the results, particularly with regards to the varying effectiveness of the different models, are less conclusive than desirable because there were only two responders.

**Successes** Whilst there are clear areas for improvement as outlined, the project was successful. Most notably, this project is the first approach to computationally generate author-styled Latin text, and, more generally, it is one of the few research endeavours focused specifically on author-styled research.

The most surprising success was the effectiveness of the sole transformer-based approach. Although the transformer encoder was listed as a potential limitation above, the transformer was remarkably successful at learning an author’s style, given the simplicity of its design and the amount of data it had available for training. Being able to quickly fine-tune a model that, according to Table 5.1, is able to generate author-styled text with the same effectiveness as ChatGPT, even if not ChatGPT’s design, and that can be run on consumer hardware is an achievement.

## 6.2 Future Work

Encountering both limitations and some exciting ideas, which were not able to come to fruition within the scope of the project, there is a lot of groundwork for future research to build upon.

The most critical way to improve this research is by increasing the number of Latinists to fill out the survey — this will happen in the immediate future, as the remaining ones from my project fill it out, but it would be good to generate more text for evaluation and have more evaluators.

Future work should also address the faults of the methodology with LLaMA and devise new better-suited ways to either fine-tune the model or perform few-shot learning approaches. Yes, this future work is primarily to address the limitations of the methodology employed here, but the use of LLaMA for interesting research encourages Meta and other companies to continue facilitating open-source modelling, which can only be a net positive for the research community.

The same experiments should also be run again, but this time with the inclusion of the newline character to see if that is what was holding the transformer and LatinBERT back from producing author-styled poetry.

Lastly, it would be interesting to apply the methods discussed in LatinBERT (18) about intertextuality as seen in 2.3.2.1 to model’s generations. This approach would enable a researcher to determine which of the author’s texts, if any, the hallucinated generations are loosely or directly based upon.

## 6.3 Conclusion

By setting specific objectives at the start of this project in 1.2, it is now possible to assess how successful the project has been by examining if these goals were met. The primary objective, which was to effectively generate author-styled Latin text, is difficult to assess because the models were seemingly able to successfully generate prose in identifiable authorial styles but struggled with author-styled poetry. The secondary objective of comparing the effectiveness of the three types of models for generation was achieved, as the experiments were run across different specified types of language models with the results in Table 5.1. This, of course, was only possible because the supporting objectives listed in 1.2 were also achieved.

---

Author-styled text generation is a niche discipline in natural language generation, and, as far as I know, this is the first work on author-styled Latin text generation. Despite the setbacks and limitations, the project and its results were exciting and laid the foundation for interesting future work on the subject. It is said that Vergil once addressed criticism about imitating Homer by saying "it is easier to steal Hercules' club from him than a line from Homer" (36); I can now confidently say it is also not so easy to steal a line from Vergil.

# Bibliography

- [1] (2023). [https://en.wikipedia.org/wiki/Prompt\\_engineering](https://en.wikipedia.org/wiki/Prompt_engineering).
- [2] (2023). <https://huggingface.co/docs/transformers/index>.
- [3] (2023a). Classical latin. [https://en.wikipedia.org/wiki/Classical\\_Latin](https://en.wikipedia.org/wiki/Classical_Latin).
- [4] (2023b). Contemporary latin. [https://en.wikipedia.org/wiki/Contemporary\\_Latin](https://en.wikipedia.org/wiki/Contemporary_Latin).
- [5] (2023c). Ecclesiastical latin. [https://en.wikipedia.org/wiki/Ecclesiastical\\_Latin](https://en.wikipedia.org/wiki/Ecclesiastical_Latin).
- [6] (2023d). Late latin. [https://en.wikipedia.org/wiki/Late\\_Latin](https://en.wikipedia.org/wiki/Late_Latin).
- [7] (2023e). Latin. <https://en.wikipedia.org/wiki/Latin>.
- [8] (2023f). Latin literature. [https://en.wikipedia.org/wiki/Latin\\_literature](https://en.wikipedia.org/wiki/Latin_literature).
- [9] (2023). Levenshtein distance. [https://en.wikipedia.org/wiki/Levenshtein\\_distance](https://en.wikipedia.org/wiki/Levenshtein_distance).
- [10] (2023g). Medieval latin. [https://en.wikipedia.org/wiki/Medieval\\_Latin](https://en.wikipedia.org/wiki/Medieval_Latin).
- [11] (2023h). Old latin. [https://en.wikipedia.org/wiki/Old\\_Latin](https://en.wikipedia.org/wiki/Old_Latin).
- [12] (2023i). Prosody (latin). [https://en.wikipedia.org/wiki/Prosody\\_\(Latin\)](https://en.wikipedia.org/wiki/Prosody_(Latin)).
- [13] (2023). Reddit post. <https://www.reddit.com/r/latin/comments/1122szcl/>.
- [14] (2023). Stylometry. <https://en.wikipedia.org/wiki/Stylometry>.
- [15] (2023j). Vulgar latin. [https://en.wikipedia.org/wiki/Vulgar\\_Latin](https://en.wikipedia.org/wiki/Vulgar_Latin).
- [16] (2023k). Wikipedia's latin grammar. [https://en.wikipedia.org/wiki/Latin\\_grammar](https://en.wikipedia.org/wiki/Latin_grammar).
- [17] Adebayo, G. and Yampolskiy, R. (2022). Estimating intelligence quotient using stylometry and machine learning techniques: A review. *Big Data Mining and Analytics*, 5:163–191.
- [18] Bamman, D. and Burns, P. J. (2020). Latin BERT: A contextual language model for classical philology. *CoRR*, abs/2009.10053.
- [19] Behr, R. (2022). Latin authorship attribution. CS4040 Mini-Research Project.
- [20] Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020). Language models are few-shot learners. *CoRR*, abs/2005.14165.
- [21] Cho, K. (2019). Bert has a mouth and must speak, but it is not an mrf. <https://kyunghyuncho.me/bert-has-a-mouth-and-must-speak-but-it-is-not-an-mrf/>.
- [22] Devlin, J., Chang, M., Lee, K., and Toutanova, K. (2018). BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.
- [23] Gonalo Oliveira, H. (2017). A survey on intelligent poetry generation: Languages, features, techniques, reutilisation and evaluation. In *Proceedings of the 10th International Conference on Natural Language Generation*, pages 11–20, Santiago de Compostela, Spain. Association for Computational Linguistics.

- [24] Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., and Chen, W. (2021). Lora: Low-rank adaptation of large language models. *CoRR*, abs/2106.09685.
- [25] Ison, D. (2020). Detection of online contract cheating through stylometry: A pilot study. *Online Learning*, 24.
- [26] Jiménez, R. (2021). Shakespeare by the numbers: What stylometrics can and cannot tell us. <https://shakespeareoxfordfellowship.org/shakespeare-by-the-numbers-what-stylometrics-can-and-cannot-tell-us/>.
- [27] Jin, D., Jin, Z., Hu, Z., Vechtomova, O., and Mihalcea, R. (2020). Deep learning for text style transfer: A survey. *CoRR*, abs/2011.00416.
- [28] Johnson, K. P., Burns, P. J., Stewart, J., Cook, T., Besnier, C., and Mattingly, W. J. B. (2021). The Classical Language Toolkit: An NLP framework for pre-modern languages. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing: System Demonstrations*, pages 20–29, Online. Association for Computational Linguistics.
- [29] Juola, P. (2017). Detecting contract cheating via stylometric methods. pages 187–198.
- [30] Karpathy, A. (2023). Let’s build gpt: From scratch, in code, spelled out. <https://www.youtube.com/watch?v=kCc8FmEb1nY>.
- [31] Kundu, R. (2022). The beginner’s guide to contrastive learning. <https://www.v7labs.com/blog/contrastive-learning-guide>.
- [32] Lin, S., Wang, W., Yang, Z., Liang, X., Xu, F. F., Xing, E., and Hu, Z. (2020). Data-to-text generation with style imitation.
- [33] Liu, H., Guo, W., Chen, Y., and Li, X. (2022). Contrastive learning enhanced author-style headline generation. <https://arxiv.org/abs/2211.03305>.
- [34] Magueresse, A., Carles, V., and Heetderks, E. (2020). Low-resource languages: A review of past work and future challenges. *CoRR*, abs/2006.07264.
- [35] Meader, C. L. (1905). Types of sentence structure in latin prose writers. *Transactions and Proceedings of the American Philological Association*, 36:32–51.
- [36] Morgan, L. (2015). Virgil, hardly trying. <https://llewelynmorgan.com/2015/02/23/virgil-hardly-trying/>.
- [37] Nutting, H. C. (1926). Reviewed work: Latin prose composition. *The Classical Journal*, 22(1):66–69.
- [P] P, B. What stylistic features should you be looking out for when analysing latin poetry? <https://www.mytutor.co.uk/answers/5416/A-Level/Latin/What-stylistic-features-should-you-be-looking-out-for-when-analysing-Latin-poetry/>.
- [39] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2023). Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.
- [40] Ramnial, H., Panchoo, S., and Pudaruth, S. (2016). Authorship attribution using stylometry and machine learning techniques. In Berretti, S., Thampi, S. M., and Srivastava, P. R., editors, *Intelligent Systems Technologies and Applications*, pages 113–125, Cham. Springer

- International Publishing.
- [41] Richards, O. (2022). A simple guide to the 7 types of latin. <https://storylearning.com/learn/latin/latin-tips/types-of-latin>.
- [42] Singh, H., Verma, G., Garimella, A., and Srinivasan, B. V. (2021). Drag: Director-generator language modelling framework for non-parallel author stylized rewriting.
- [43] Sohn, D.-N., Lee, J.-T., and Rim, H.-C. (2009). The contribution of stylistic information to content-based mobile spam filtering. pages 321–324.
- [44] Syed, B., Verma, G., Srinivasan, B. V., Natarajan, A., and Varma, V. (2019). Adapting language models for non-parallel author-stylized rewriting. *CoRR*, abs/1909.09962.
- [45] Taori, R., Gulrajani, I., Zhang, T., Dubois, Y., Li, X., Guestrin, C., Liang, P., and Hashimoto, T. B. (2023). Alpaca: A strong, replicable instruction-following model. <https://crfm.stanford.edu/2023/03/13/alpaca.html>.
- [46] Tikhonov, A. and Yamshchikov, I. P. (2018). Guess who? multilingual approach for the automated generation of author-stylized poetry. *CoRR*, abs/1807.07147.
- [47] Toshevskaa, M. and Gievska, S. (2022). A review of text style transfer using deep learning. *IEEE Transactions on Artificial Intelligence*, 3(5):669–684.
- [48] Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., Rodriguez, A., Joulin, A., Grave, E., and Lample, G. (2023). Llama: Open and efficient foundation language models.
- [49] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. *CoRR*, abs/1706.03762.
- [50] Wang, A. and Cho, K. (2019). BERT has a mouth, and it must speak: BERT as a markov random field language model. *CoRR*, abs/1902.04094.
- [51] Wang, C., Cho, K., and Gu, J. (2019). Neural machine translation with byte-level subwords. *CoRR*, abs/1909.03341.
- [52] Wang, E. J. (2023). Tloen/alpaca-lora: Instruct-tune llama on consumer hardware. <https://github.com/tloen/alpaca-lora>.
- [53] Wikibooks (2022). Latin/stylistic features of latin verse and prose. [https://en.wikibooks.org/w/index.php?title=Latin/Stylistic\\_Features\\_of\\_Latin\\_Verse\\_and\\_Prose](https://en.wikibooks.org/w/index.php?title=Latin/Stylistic_Features_of_Latin_Verse_and_Prose).
- [54] Yang, C., Sun, M., Yi, X., and Li, W. (2018). Stylistic Chinese poetry generation via unsupervised style disentanglement. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3960–3969, Brussels, Belgium. Association for Computational Linguistics.

## Appendix A

# User Manual

The User Manual will walk through the process of running the code used in the project. This project also made use of ChatGPT, which the User Manual does not walk through in great detail. In short, to use ChatGPT as it was done in this project, go to `chat.openai.com/`, create or sign into an existing account, and then follow the prompting methodology as described explicitly in 3.5.

The rest of the manual walks through the process of running the models and experiments done for the Transformer and LatinBERT.

### A.1 Requirements

After obtaining a copy of the source code for this project, which is best done through its GitHub Repository: <https://github.com/SufurElite/LatinNLG>. There are two software prerequisites for this project:

- **Python 3.9** : Although it other versions of Python 3 may be able to run the code, the version of Python throughout this project has specifically been 3.9.16.
- **Anaconda** : Anaconda is a useful package manager, which can also be used to install Python. Anaconda can be installed through its download page here: <https://www.anaconda.com/download>. Moreover, the reason why we wish to install Anaconda in particular is because some of the project's code uses Jupyter Notebooks, which are runnable with Anaconda.

Unfortunately, due to the nature of fine-tuning larger models, you may find that your computer does not meet a minimum hardware requirement to fine-tune LatinBERT. It should still be possible, however, to fine-tune and run the Transformer's models.

### A.2 Create Environment and Install Dependencies

Once the software has been installed, we can immediately make use of Anaconda to create a self-contained environment for running the project.

1. Open a terminal, if using a unix based machine, or open the Anaconda Prompt on windows.
2. We will want to navigate to wherever you cloned the LatinNLG GitHub repository, which can be done by entering

```
cd "<File Path to LatinNLG>"
```

3. Now, we create a new Anaconda environment for the project by entering the command

```
conda create -n python=3.9 LatinNLG pip
```

which will create an environment using Python 3.9 with pip, Python's package manager, called LatinNLG. This should automatically activate the environment, but, if LatinNLG is not written on the left-hand side of the terminal, you can run the command

```
conda activate LatinNLG
```

, which will deactivate whatever the current environment is and activate LatinNLG.

4. Once you are using the LatinNLG environment, we can install all of the library dependencies used in LatinNLG by entering

```
pip install -r requirements.txt
```

5. After the libraries are installed, the only other thing we are missing are the LatinBERT weights, which can be obtained by entering

```
cd LatinBERT
./download.sh
cd ..
```

, which will retrieve the requisite LatinBERT weights and put them in a subdirectory called latin\_bert.

After following those steps, you will have everything needed to begin running the project.

### A.3 Running the Project

With the dependencies in order, all that remains is to start the jupyter notebooks.

1. In the same terminal, run the command

```
jupyter notebook
```

, which will start a local server and open a webpage that has the files from the directory available.

2. At this point you can click on either `LatinBERT-FineTune.ipynb` or `LatinTransformerFineTune.ipynb`, depending on what you would like to run.

Each notebook is modifiable and its code is accompanied by instructions and explanation. If you wish to fine-tune LatinBERT, then you can do so through the notebook by changing the author to your preferred author and running the cells within the fine-tune section. Likewise, if you then want to generate new text, at the bottom of that notebook you can load in the weights of a fine-tuned LatinBERT to generate new text with the prompt of your choosing.

If you want to run one of the pre-trained Transformer models from the paper (Caesar, Cicero, Ovid, or Vergil), you can select the `LatinTransformerFineTune` notebook, choose your preferred author and starting text, and similarly generate new text.

If you want to fine-tune a transformer to a Latin author, instead of running the jupyter notebook, you would open `transformer.py` in the `LatinNLG` directory with a text editor and change the list of authors to your preferred author list. Afterwards, through the same terminal as before you run

```
python transformer.py
```

, which will train and save the weights of the different authors you requested. At this point, you can now run the jupyter notebook and select the `LatinTransformerFineTune` notebook to generate new text with your newly trained weights.

## Appendix B

# Maintenance Manual

The Maintenance Manual covers how to execute the project's code, the dependencies and requirements for the project, an overview of the different files and their purpose, and directions for future improvement.

## B.1 Setting-up the Development Environment

As this project uses Python, an interpreted rather than compiled language, setting-up the an environment to modify the code is same process as wanting to run the code, as seen in the User Manual. It is first required to obtain the code base, which can be done through cloning its GitHub repository: <https://github.com/SufurElite/LatinNLG>.

### B.1.1 Requirements

The requirements can be divided into hardware and software.

- **Hardware:** From a hardware perspective, a GPU, whilst not requisite for modifying the code, will be very useful for fine-tuning the language models, particularly as they grow in size, because it allows the code to make use of the GPU's parallel computing.
- **Software:** Anaconda, a package manager, is a requirement because the project uses Jupyter notebooks, which allow easy to use virtual environments that let us install the other dependencies for the project in an isolated fashion. Another software requirement is Python 3, and, although the code might work with other versions of Python, Python 3.9 is recommended — in particular, the version of Python used throughout the project was 3.9.16. Both the Python and Anaconda requirements can be addressed by installing Python 3 through Anaconda. Anaconda's installation package can be found here: <https://www.anaconda.com/download>.

After Anaconda is installed, you should open the LatinNLG directory (from the cloned GitHub Repository) in a terminal or Anconda Prompt and enter the following commands:

```
conda create -n python=3.9 LatinNLG pip
conda activate LatinNLG
pip install -r requirements.txt
cd LatinBERT
./download.sh
cd ..
```

These commands will create a Python 3.9 Anaconda virtual environment with the necessary Python packages installed and will download the LatinBERT weights in the desired location. And, now running, making any modification, or extending the program can be done.

### B.1.2 Running the Project

Once you have met the requirements, you can run the models to generate new text by entering

```
jupyter notebook
```

which will create a local server and allow you to select one of the Jupyter notebooks. By opening the LatinBERT-FineTune notebook in the browser, you can generate text from a pre-trained LatinBERT model or fine-tune the LatinBERT model on an author's subset. Alternatively, you can select the LatinTransformerFineTune notebook, which is only set-up for generation of a pre-trained model at this point. You can fine-tune a transformer model on an author's subset, however, by modifying the list of authors in the transformer.py file and running

```
python transformer.py
```

## B.2 Summary of Files

This section provides a table containing an explanation for every necessary and non-self-explanatory file in the repository (this excludes `__init__.py`, text, and image files). Some of the files are grouped together if it is redundant to have them listed separately (e.g., each of the different author model weights are grouped together).

| Name of the File(s)     | Its Role  |
|-------------------------|---|
| Data/dataExp.py         | dataExp.py contains the CorpusInterface class, which is used throughout the research project. The CorpusInterface loads either a provided pickled corpus or it creates a new pickled corpus by associating all the authors of the different Latin corpora to the author's works after applying preprocessing techniques. If the Latin corpora are not downloaded, it will also call the necessary functions to download them. The Corpus is equipped with different auxiliary functions that were designed to extract authors and their texts in different formats dependent. |
| Data/fetch.py           | fetch.py has a function that retrieves the different Latin corpora and tokenizer models.  |
| Data/plot.py            | plot.py contains a number of ways to plot information about the Corpus Interface.   |
| Data/preprocess.py      | preprocess.py defines a PreProcessor class for the Corpus Interface, and the main utility of the class is to apply the desired preprocessing techniques to text.  |
| Data/text_corpus.pickle | text_corpus.pickle is the most recent serialised corpus object. It's the Corpus Interface's dictionary associating the authors to their works.  |

|   |  |
|---|--|
| LatinBERT/download.sh   | download.sh allows you to download the LatinBERT weights directly to the repository.   |
| LatinBERT/gen_berts.py  | gen_berts.py defines the class for LatinBERT and largely follows the same class definition in the original LatinBERT GitHub page, but the class definition in the GitHub uses an unsupported version of a function from the Hugging Face Transformers Library whereas this version is compatible with the Hugging Face Transformers Library. |
| LatinBERT/LatinTok.py   | LatinTok.py contains the class definition for LatinBERT's tokenizer.   |
| LatinBERT/predict_words.py  | predict_words.py is used for performing LatinBERT text generation.   |
| LatinBERT/<br>special_tokens_map.json,<br>tokenizer_config.json,<br>tokenizer.json,<br>latin.subword.encoder) | (special_tokens_map.json, tokenizer_config.json, tokenizer.json, and latin.subword.encoder) These files all contain information for LatinBERT's tokenizer and are necessary to load the tokenizer.   |
| Results/quickshuf.py  | quickshuf.py is used to shuffle the model's generated texts into a random order for the survey.  |
| Transformer/<br>("Author"_model.pt)   | (Author) These files, where "Author" can be replaced by any of the authors in the project, are the weights for the Transformer.  |
| LatinBERT Analysis.ipynb  | This Jupyter notebook was used to perform initial investigation into how well LatinBERT does with generating text without any fine-tuning.   |
| LatinBERT-FineTune.ipynb  | This Jupyter notebook walks through the process of fine-tuning and text-generation for LatinBERT.  |
| LatinTransformerFineTune.ipynb  | This Jupyter notebook walks through the process of generating new text with the transformer model architecture.  |
| transformer.py  | transformer.py is the file to train the transformer on particular author-subsets, which are determined by the authors list near the start of the file.   |

**Table B.1:** A Summary of all the Files

### B.3 Direction for Improvement

In this project, there are some immediate areas for improvement that can be addressed quickly:

- **Enabling training as well as fine-tuning for the transformer in its notebook.** As the class architecture is already defined, it would only require including a training loop in the notebook like in the LatinBERT notebook.
- **Creating a new pre-trained Latin transformer and then fine-tuning in on author subsets.** As the Corpus Interface has been fixed since the first methodology for the transformer

was attempted (pre-training a transformer on all of the text and then fine-tuning on an author), it is worthwhile to try again and see if it yields any immediate improvement.

- **Including the newline character in the dataset when generating poetry.** One of the results from the survey was that the models struggled with poetry, likely due to the absence of a newline character, so it is worth seeing if its inclusion improves the poetry generation.
- **Designing a user interface for the models.** Currently, the models are being run through a Jupyter notebook or trained on the terminal, but it would be nice to have a clean user interface without compromising on the functionality.

## Appendix C

# Generated Output

### C.1 Survey Text Generated

| Model       | Intended Author | Generation  | Approximate Translation <sup>1</sup>   |
|-------------|-----------------|---|--|
| ChatGPT     | Caesar          | <p>Olim sapiens vir nomine Rufo erat, qui multos annos in Gallia commoratus est. Ille tempus suum in studiis et litteris consumebat, sed postea ad militiam se contulit et ad Caesarem venit. Illo tempore Caesar in Italia bellum gerebat, et Rufo statim se ad eius partes adiunxit. Cum Caesar in Galliam rediit, Rufo cum eo venit et multa pro Caesare pugnavit.</p> <p>Sed Rufo non solum fortis miles sed etiam sapiens vir erat. Semper cum Caesar consilium capiebat, Rufo ei consilium dabat et multa utilia suggerebat. Interdum autem cum militibus suis in castris morabatur et ad eos litteras scribebat. In una epistula scripsit: "Hoc tempore in Gallia non solum pugnamus sed etiam discimus, ut postea sapientiores simus."</p> <p>Ita Rufo non solum fortis miles sed etiam sapiens vir fuit, cuius memoria semper apud milites Caesaremque carissima fuit.</p> | <p>Once upon a time there was a wise man named Rufus, who lived many years in Gaul. He spent his time in studies and literature, but afterwards he turned to the military and came to Caesar. At that time Caesar was waging war in Italy, and Rufus immediately joined his side. When Caesar returned to Gaul, Rufus came with him and fought many battles for Caesar.</p> <p>But Rufus was not only a brave soldier but also a wise man. Whenever Caesar took a decision, Rufus gave him advice and many useful suggestions. Sometimes he stayed with his soldiers in the camp and wrote letters to them. In one letter he wrote: "At this time in France we not only fight but also learn, so that we may be wiser afterwards."</p> <p>Thus Rufus was not only a brave soldier but also a wise man, whose memory was always dear to the soldiers and to Caesar.</p> |
| LatinBERT   | Caesar          | <p>olim sapiens vir nomine rufo erat. hic cum equitibus colloqui venit. idem cum suis agit liberius et magna parte militum disceditur. tum vero etiam reliquis civitatibus terror et a quibusdam dissentire videbatur. tum vero ex omnibus civitatibus principes convenerunt</p>  | <p>Once upon a time there was a wise man named Rufus. He came here to talk with the knights. He does the same with his own people more freely, and a large part of the soldiers leave, but then also the rest of the states were alarmed, and some seemed to disagree. then the princes from all the states assembled</p>  |
| ChatGPT     | Cicero          | <p>Olim sapiens vir nomine Rufo erat, qui in re publica multos annos honorifice versatus est. Cum illo tempore res publica maxime turbulenta esset, Rufo saepe Caesarem adversatus est, ceterosque populares homines fortiter accusavit.</p> <p>Sed Rufo non solum eloquentia sed etiam sapientia pollebat, et interdum Caesaris acerbas ambitiones vituperabat. Cuius consilia multi viri boni secuti sunt, et civitas a calamitate servata est.</p> <p>Cum autem Rufo in senatum rediret, conspectus est ut fuit omnium carus et gratus. At non solum gratiam sed etiam laudem meruit, quod consilio et virtute rem publicam servavit. Et postea inter eos qui patriae bene meruerunt numeratus est, cuius memoria in saecula saeculorum vivet.</p>   | <p>Once upon a time there was a wise man named Rufus, who was honorable for many years in public affairs. As the state was at that time most tumultuous, Rufus often opposed Caesar, and strongly accused the other popular men.</p> <p>But Rufus excelled not only in his eloquence, but also in his wisdom, and sometimes reproached the bitter ambitions of Caesar. Many good men followed his advice, and the city was saved from disaster.</p> <p>But when Rufus returned to the senate, it was evident that he was beloved and appreciated by all. But he earned not only thanks but also praise, because he preserved the state by his counsel and courage. And afterwards he was numbered among those who have done well for his country, whose memory will live forever.</p>  |
| Transformer | Vergil          | <p>olim sapiens vir nomine rufo erat, otis quos illius incidit, provolvendum pecudes; at illi specubus uter corpora tepéfant mavora, ut rebus eurytis subducere claudio et magno de gentes luctantur longe commortalibus arma.</p>  | <p>once upon a time there was a wise man named Rufus; but he saw how the bodies of the sparrows were warmed, so that they might bring down the lame and great things of the nations, with their mortal weapons far away.</p>   |
| Transformer | Cicero          | <p>olim sapiens vir nomine rufo erat, ortes moribus illi operumque bonis severitatem. ita mutabunt praesentiri sibi autem populari riges acro esse videtur. tu ad philosophisous non hoc imaginium esse pecuniam: vetteris platonis tum commendo exstittise nisi</p>  | <p>Once upon a time there was a wise man named Rufus. so they will change the way they are presented to him, but it seems that the people's rules are keen. You do not think that these images are money for the philosophers: I recommend to the old Plato that they existed only then</p>  |

<sup>1</sup>Approximate translation provided by google translate, <https://translate.google.co.uk/>

|             |        |  |  |
|-------------|--------|--|--|
| ChatGPT     | Vergil | Olim sapiens vir nomine Rufo erat, quem Romae omnes digni laude viri colebant. Ingenio doctus, et artibus liberalibus imbutus, Rufo multas scientias e studio addiscere curavit. Sed Rufo praecipue poesis cordi erat, et carmina elegiaca componere sapientissimus existimabatur. Namque carminibus suis, amoris et virtutis laudes celebravit, et deorum divina munia mirabili modo descripsit. Ingenio praeditus et forma egregia, Rufo multarum feminarum adulescentiam amorem adferre poterat. Sed, ne quis eorum amore captus duceretur, Rufo semper modestissime se gessit et abstinuit ab omni lascivia. Inter Graecos versatus, Rufo multa ab eis didicit et artem poeticam perfecit. Quae tam perfecte scripsit, ut ea ad aeternitatem pervenirent et omnes carminibus eius semper laeti essent. | Once upon a time there was a wise man named Rufus, whom all the worthy men of Rome worshiped. A learned genius, and imbued with liberal arts, Rufus took care to learn many sciences from study. But Rufus was particularly fond of poetry, and was considered the wisest to compose elegiac poems. For with his songs he celebrated the praises of love and virtue, and described the divine functions of the gods in a wonderful manner. Gifted with talent and excellent form, Rufus was able to bring the love of many young women. But, in order that no one might be led astray by their love, Rufus always behaved in the most modest manner and abstained from all lasciviousness. Living among the Greeks, Rufus learned much from them and perfected the art of poetry. What he wrote so perfectly that they would reach eternity and all would be happy with his poems forever. |
| LatinBERT   | Vergil | olim sapiens vir nomine rufo erat sed nunc lumine da nomine dicunt, huic cognomen ; iulius nomen tuum dicunt : cui gravido cognomen fuerat, sed aurea nomen, huic cognomen de patre fuit. cornelio ordine nemo patricigravide sanguine esse voluerunt.   | once upon a time there was a wise man named Rufus, but now they say give him a name of light; They say your name is Julius: to whom it had been a surname when pregnant, but the name was golden, to this one it was a surname from his father. In the order of Cornelius, no one wanted to be of the patriarchal blood.   |
| ChatGPT     | Ovid   | Olim sapiens vir nomine Rufo erat, Ingenio doctus, consiliisque sagax, Linguam callidam, promptamque ad verba iocosa, Aurea cuius erat sanguinis origo. Fortunae motus non impavidus horret, Nec laudis avidus, nec acri invidia Turbatur, sed in medio haud inscius urbis Naturae rerum structuram scrutatur. Mille modis vitae ludens experiensque, Quaeque canit, natales sua carmina vates Mutat, et haec illi vitam tam bene ducit, Quod carmen tantum viveret immortalis.  | Once upon a time there was a wise man named Rufus, Learned wit, and shrewd in counsels, A clever tongue, ready for witty words, Gold was the origin of his blood. The movement of fortune does not tremble fearlessly, Neither greedy for praise, nor bitter envy He was disturbed, but not unconscious in the middle of the city He examines the structure of things in nature. Playing and experiencing a thousand ways of life, Each of them sings their own birthday songs He changes, and this leads his life so well, That song would only live immortal.  |
| LatinBERT   | Cicero | olim sapiens vir nomine rufo erat sed etiam vir et virtute divina gloria, pompeius, is nimirum ille quidem, ne consultus quidem, sed tamen artis suae defensor et temporum dico memoria dignissimus, omni memoria temporum dignissimus, vobis erit iudicio   | Once upon a time there was a wise man named Rufus, but he was also a man of divine glory, Pompeius, he, of course, was not even a counselor, but still a defender of his art and a teller of the times, worthy of memory, worthy of all the memory of the times, he will be judged by you  |
| Transformer | Caesar | olim sapiens vir nomine rufo erat, oallis cominuti ex oppido prohiberent, ad quam regionem consuevit, et siciliam communicio posset, quod illum ipsi quam celerrebribet, quaeque esse aut animadvereret se consulerent.  | Once upon a time there was a wise man named Rufus, who, being crushed by all, would stop from the town, to which country he was accustomed, and could communicate with Sicily, that they would consult him as quickly as he could, and whatever he noticed or noticed.   |
| Transformer | Ovid   | olim sapiens vir nomine rufo erat, od patiarer virgo, nec avers vitiet nivea sub axe, in chisaeque variarum thybri tempus hlarum texta syraco-rumque implevit aesone, esse mei. adspiciens ubi iuque complecta lubra licebit; iuppitis ausis lustravit in undis  | Once upon a time there was a wise man named Rufus, who would suffer as a virgin, nor would the snow fall under his axe. looking where it will be possible to embrace the joy; He traveled the waves with the boldness of a hunter  |
| LatinBERT   | Ovid   | olim sapiens vir nomine rufo erat, a gente tua sumepossedi. de gente syarchesia primus, in illa divitiis carens bona, nulla domo, et quamvis osi, tamen uxor, et illi gratas erat ; nec tamen amissa mansit  | Once upon a time there was a wise man named Rufus, whom I took from your people. He was the first of the race of the syarches, in that country lacking in wealth and possessions, and having no house, and although he was estranged, he still had a wife, and she was grateful to him; and yet he did not remain lost   |

Table C.1: The generated text included in the Survey

## C.2 LatinBERT Initial Analysis

| Author | Correct Continuation | Generated Continuation |
|--------|----------------------|------------------------|
|--------|----------------------|------------------------|







