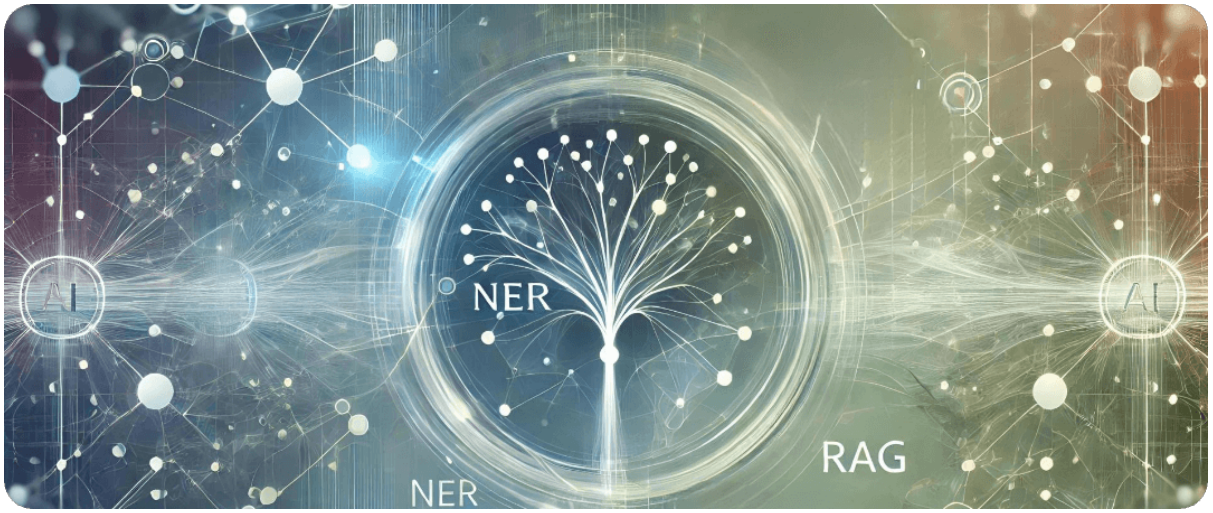

Text Normalization Leveraging NER and Semantic-Aware Few-Shot Prompting with LLMs

Michalis Katras

February 24, 2025

Abstract

This report explores two approaches for normalizing raw composition writer information: Named Entity Recognition (NER) using a fine-tuned SpaCy transformer model and semantic-aware few-shot prompting leveraging OpenAI's large language models with structured output. The results demonstrate the effectiveness of combining deterministic rules with NLP techniques to achieve accurate text normalization. This report provides detailed explanations of the methods, implementations, and suggestions for further improvements.

Contents

1 Introduction

Text normalization is a crucial task in natural language processing that involves transforming raw text into a standardized format. In the context of composition writer information, normalization aims to remove redundant details and retain only the relevant writer names.

1.1 Objective

The primary objective of this project is to develop an effective solution for normalizing raw composition writer information. The solution should be able to:

- Remove non-author-related information and redundant details.
- Accurately extract and retain the relevant writer names.
- Handle diverse and complex text structures, including non-Latin characters.

To achieve this, two distinct approaches were explored:

- **NER using SpaCy Transformer**: Fine-tuning a SpaCy NER model [2] based on the RoBERTa transformer [4] to identify and extract writer names.
- **Semantic-Aware Few-Shot Prompting with LLM**: Using OpenAI's large language models [5] (LLMs) to generate structured outputs that retain only the writer's names while excluding redundant information. This approach employs semantic similarity to select relevant examples, which are then provided to the model for few-shot prompting.

1.2 Limitations

One significant limitation of this project was the time constraint. The task was completed within one week, alongside another assignment, which limited the extent of experimentation and refinement. Consequently, there may be areas where further optimization and exploration could enhance the performance and accuracy of the proposed solutions.

Despite these limitations, the approaches presented in this report demonstrate the potential for combining deterministic rules with advanced NLP techniques to achieve accurate text normalization.

2 Provided Dataset

The dataset provided is a CSV file containing 10,000 raw compositions from writers' texts and their normalized versions. The dataset includes the following columns:

- **raw_comp_writers_text**: Original text containing composer and writer information.
- **CLEAN_TEXT**: The clean, normalized text after removing redundant information.

A snapshot of the provided data is shown in Table 1:

raw_comp_writers_text	**CLEAN_TEXT**
"SUBZEROSWIZ,Yeat"	SUBZEROSWIZ/Yeat
Ray Davies	Ray Davies
Yuvanshankar Raja & Ramajogayya Sastry	Yuvanshankar Raja/Ramajogayya Sastry
Auditory Music	Auditory Music
"<Unknown>/Wright, Justyce Kaseem"	Wright/Justyce Kaseem
Sameer/Nadeem-Shravan	Nadeem-Shravan

Table 1: Examples of raw and normalized composition writer texts.

2.1 Analysis

In this section, we analyze the data set to understand its characteristics and identify any potential issues or patterns that could impact the normalization process.

We first examine how many values in the `CLEAN_TEXT` column are identical to those in the `raw_comp_writers_text` column. The results indicate that 6,323 entries, or 63.23% rows, of the dataset have identical columns.

Next, we check for rows that have NaN values in the `CLEAN_TEXT` column. It is calculated that about 13.41% (1341 entries) have a NaN value in the column CLEAN_TEXT. While some of these cases are expected, as they lack any writer information (e.g., UNKNOWN WRITER (999990) → NaN), others contain writer details in the `raw_comp_writers_text` column but still have NaN values in the `CLEAN_TEXT` column (e.g., Phanendra/Shakti → NaN). These instances are considered outliers since they can affect model training, potentially leading to incorrect inferencing. However, isolating and removing such cases is challenging, and it is not something we will address in this project.

We also analyze the most common words in the `raw_comp_writers_text` column to identify frequent words that are not related to writer's information (e.g. Copyright Control). These words can be removed to simplify the text using rule-based methods. A histogram of the top 50 most common words is presented in Figure 1.

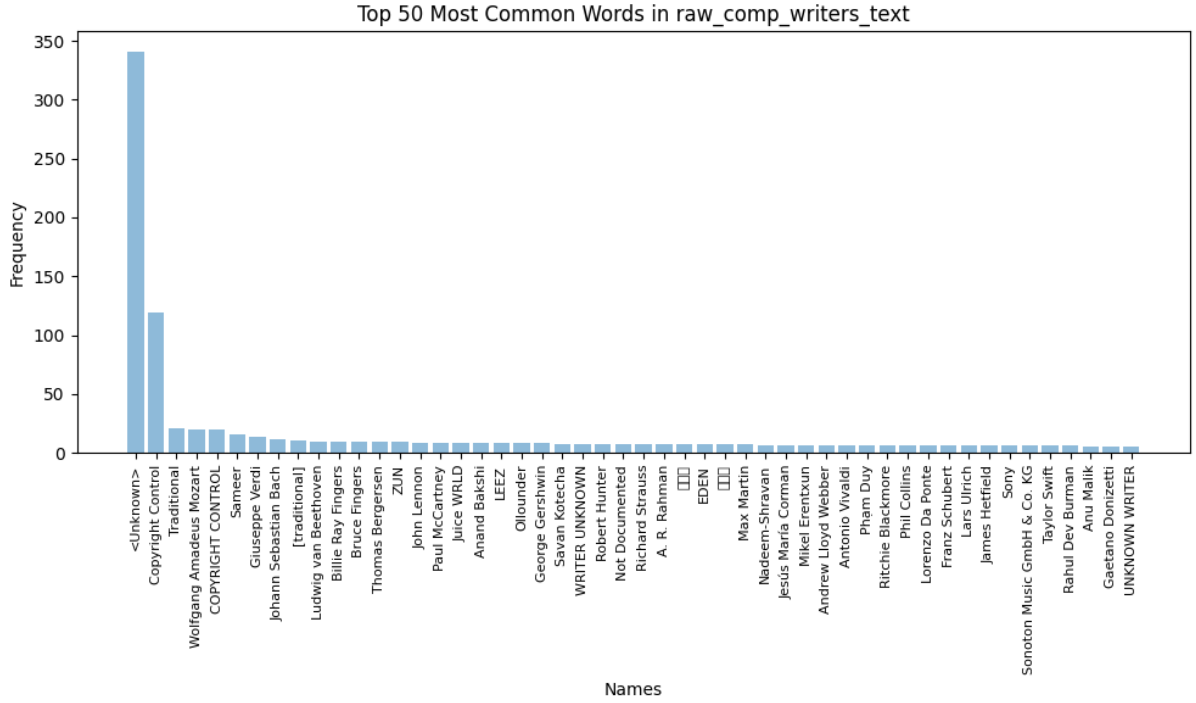

Figure 1: Most frequent words in the raw_comp_writers_text column.

To isolate names from the `CLEAN_TEXT` column, we assume that writer names are separated by a frontslash (`/`). Additionally, for a split value to be considered a valid name, we assume that it must contain more than three alphabetical characters. This approach helps filter out false values in the `CLEAN_TEXT` column, such as "THE," "&", and other irrelevant tokens.

Ensuring this consistency is crucial for accurately labeling the data, which is necessary for training a Named Entity Recognition (NER) model. Ideally, each entity would be manually labeled for maximum accuracy, but due to time constraints, we opted for an automated approach. A histogram of the top 50 most common names extracted can be visualized in Figure 2.

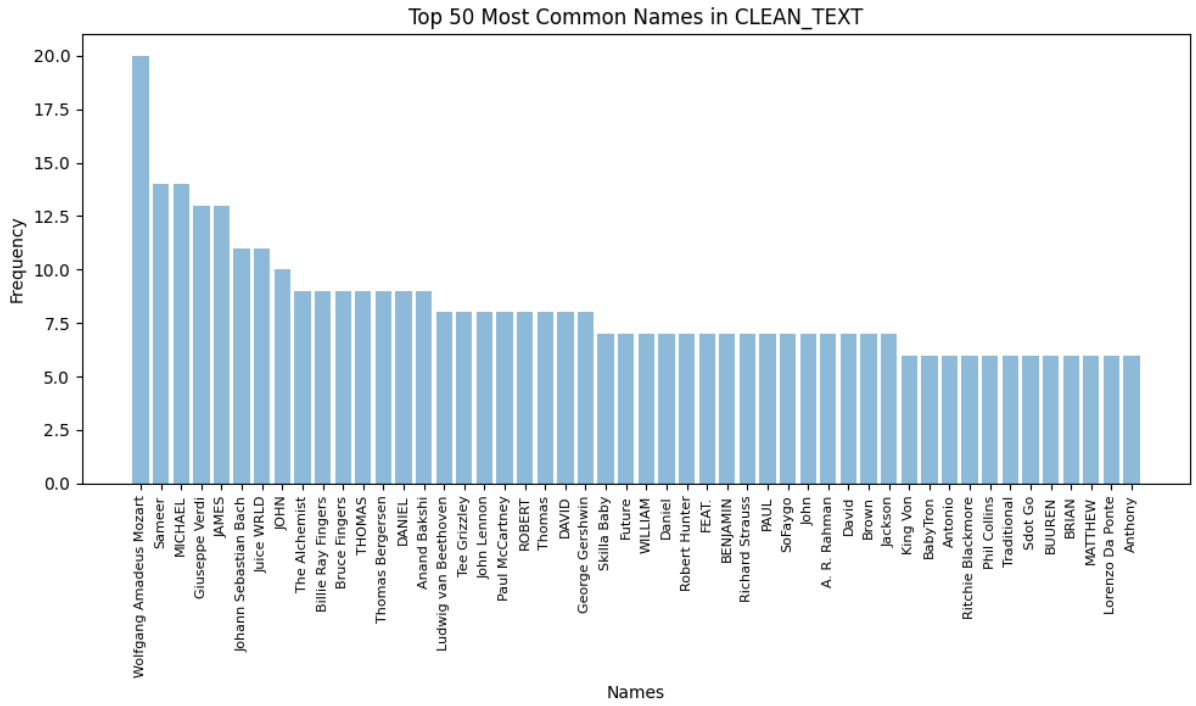

Figure 2: Most frequent names in the CLEAN_text column.

3 Data Labelling & Preprocessing

In this section, we will outline the preprocessing steps applied in each of the proposed approaches. Additionally, we will describe the methodology used to automatically label the dataset for training a Named Entity Recognition (NER) model.

3.1 Prompt-based approach

In this approach, preprocessing is performed to ensure clean and structured text data before using it as an input for the LLM model. The key preprocessing steps involve removing noisy names, standardizing delimiters, and handling missing or empty entries. By establishing deterministic rules, we can streamline the raw data and eliminate some redundancies, making it easier for the LLM to generate accurate and consistent results.

3.1.1 Preprocessing Steps

- **Noisy Name Removal:** A predefined set of non-informative names (such as *UNKNOWN WRITER*, *Copyright Control*, and *Traditional*) is removed from the dataset. These names do not contribute useful information and could negatively impact model performance. The names were manually selected from the top 150 most common words in the column containing the raw information.
- **Delimiter Cleaning and Standardization:**
  - Extra delimiters (e.g., multiple consecutive slashes or commas) are removed.
  - Double delimiters (e.g., `//`) are corrected to a single delimiter.
  - Leading and trailing delimiters are stripped to maintain consistency.
  - Spaces are trimmed to ensure clean text formatting.
- **Handling Empty or Missing Data:**
  - Any text entries that are empty or become empty after preprocessing are replaced with an empty string instead of `NaN`.
  - This prevents irrelevant placeholders from interfering with training.

- **Dataset Splitting and Storage:**
  - The processed dataset is split into 2 small subsets.
  - A small subset (around 100 samples) is extracted for creating the vector storage to semantically retrieve relevant examples for few-shot prompting.
  - A test set of 1,000 samples is generated, with any missing values in the `CLEAN_TEXT` column removed.
  - The cleaned datasets are saved as CSV files for further use.

It is important to note that, due to the time constraints of this project, there was insufficient time to optimize the vector storage creation and access methods. As the number of examples in the storage increases, the time required for inference also rises. This is why the example set was kept to just 100 samples. For model evaluation, the sample size is limited to 1,000, considering the cost of calling OpenAI's API. This number was considered sufficient for a proof-of-concept approach.

3.2 NER-based approach

In this approach, we preprocess and prepare the dataset for training a Named Entity Recognition (NER) model using spaCy [2]. The preprocessing steps ensure that the dataset is clean, well-structured, and ready for training while filtering out irrelevant or noisy data.

3.2.1 Preprocessing Steps

Similar to the LLM-based approach, a predefined list of noisy or irrelevant names is removed from the dataset to prevent the model from encountering unnecessary noise during both training and inference. Additionally, the same preprocessing steps are applied to clean and standardize delimiters, as well as handle NaN values, ensuring a consistent and well-structured dataset.

3.2.2 Data Splitting and Transformation

The dataset is split into training, validation, and test sets to ensure that the model is evaluated on different subsets of the data:

- **Splitting:** The data is divided into training (80%), validation (10%), and test (10%) sets, with the test set used for final evaluation and the validation set used to tune the model.
- **SpaCy Data Format:** The training, validation, and test sets are transformed into a format compatible with spaCy. This involves converting the text and the labeled entities into SpaCy's custom format, which is used for model training.
- **Saving the Processed Data:** The resulting datasets are saved in multiple formats, including CSV, JSON, and spaCy's native DocBin format, making them ready for use in the spaCy pipeline.

3.2.3 Automated Entity Labeling

To automate the process a list of names is extracted from the CLEAN_TEXT column. As mentioned before, to extract names from the CLEAN_TEXT column, we assume that writer names are delimited by a forward slash (/). Furthermore, we define a valid name as any token containing more than three alphabetic characters. This method effectively filters out non-relevant tokens like "THE," "&," and other symbols in the CLEAN TEXT column.

The extracted names are then used to perform one-to-one string matching in the column containing the raw information. If a match is found, the row is retained; otherwise, it is discarded. This step is crucial for transforming the dataset into the required format for training a NER model.

The labeling process is automated by extracting unique names from the cleaned text and associating them with the `PERSON` entity type. For each entity found in the raw text, the corresponding indexes (start and end positions) are identified and labeled. These labeled entities are then transformed to spaCy's native DocBin format, making them ready for fine-tuning the model. An example JSON structure of a labeled datapoint is shown below:

```
{"annotations": [{"text": "Deondre cloyd/Not Documented","entities": [[0,13,"PERSON"]]}
```

While this approach enables fast and efficient entity labeling for a large number of data points, it has some limitations. Many rows were discarded because the writer information did not match exactly across both columns. For example, the pair `<Unknown>/Wright, Justyce Kaseem` → `Justyce Kaseem Wright` would be discarded due to inconsistencies in name formatting (e.g., first and last names are not always presented in the same order). Ideally, the dataset should be labeled by separately identifying first and last names to ensure a consistent and compact normalization format. Additionally, the rule-based method we used to extract names relies on assumptions, meaning it cannot guarantee that all names are captured. Despite these drawbacks, we chose this approach due to the project's time constraints. While manual annotation would provide more consistent and accurate results, it would be more time-consuming.

4 Proposed Implementations

4.1 Semantic-Aware Few-Shot Prompting

In recent years, Large Language Models (LLMs), such as OpenAI's GPT-4o [5], have helped natural language processing by offering capabilities for understanding, generating, and transforming text. These models are trained on vast datasets and can perform a wide range of tasks, from answering questions to generating structured outputs. One of the advantages of LLM APIs is their ability to adapt to a wide variety of use cases, including complex text normalization tasks.

By leveraging LLM APIs, we can efficiently handle text normalization by guiding the model to clean, standardize, and format text based on predefined rules or schemas. The inherent flexibility and contextual understanding of LLMs allow for accurate processing, making them useful for generating consistent, well-structured outputs across a range of input scenarios.

4.1.1 Semantic Few-Shot Prompting

Few-shot prompting is a technique that allows a model to learn from a small number of examples provided within the prompt itself. In a few-shot scenario, the model is presented with a few examples of the task it needs to perform, allowing it to generalize and apply the patterns seen in the examples to new, unseen inputs. This is particularly useful for tasks like text normalization, where providing a few examples can guide the model to produce consistent results.

Semantic few-shot prompting enhances this approach by not only providing examples but also ensuring that the examples used are semantically relevant and contextually aligned with a specific user request. By leveraging semantically rich examples, the model's performance can be further improved, as it can better understand the underlying patterns and nuances of the input data.

4.1.2 Structured Outputs with Custom Schemas

Structured outputs ensuring that the results generated by language models conform to specific formats. In this context, custom schemas can be defined to enforce particular structures for the output, ensuring consistency and standardization. A common approach involves using libraries like Pydantic, which provides a way to validate and manage data structures by specifying fields, types, and validation rules. By enforcing structured outputs with custom schemas, it becomes easier to transform raw model responses into clean, standardized data that adheres to predefined formats.

4.1.3 Architecture

The pipeline illustrated in Figure 3 outlines a systematic process for handling normalized text and generating its normalized equivalent using OpenAI's GPT-4o model. The key components and their interactions in this pipeline are as follows:

1. **Raw text:** The process starts with the user providing a string of one or more normalized inputs separated by a new line.

2. **Preprocessing:** Then the preprocessing steps mentioned in the previous section are applied to the input.

3. **Retrieving Contextually Relevant Examples:** The system retrieves relevant examples from a vector store built using 100 randomly selected samples. This vector store is implemented with ChromaDB [1] and leverages OpenAIEmbeddings [3] to encode textual data. To identify the most

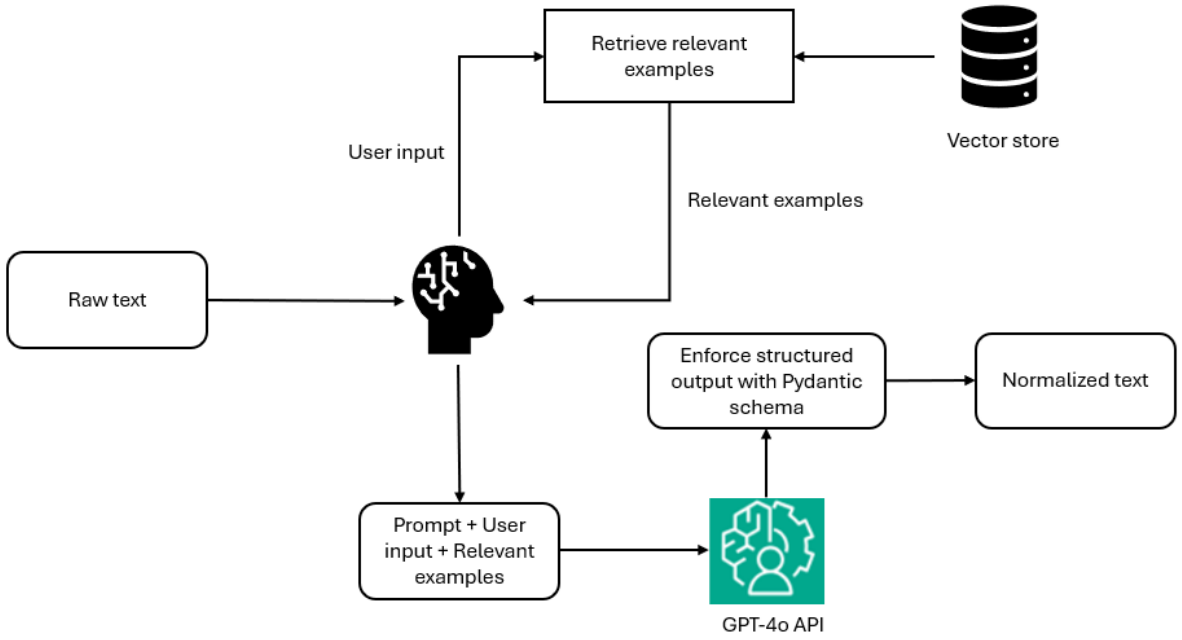

Figure 3: Graph representation of the semantic-aware few-shot prompting approach.

relevant examples, the system employs a semantic similarity algorithm, which selects the top k=5 examples that are most contextually similar to the user input.

4. **Prompting:** The next step involves constructing the model's prompt, which includes three key components: a system prompt providing general task instructions, the user's query, and the retrieved relevant examples. This structured prompt helps guide the model toward generating accurate and contextually appropriate responses.

5. **Structured output:** The final step involves structuring the model's response into a useful and standardized format. To achieve this, we employ a nested Pydantic schema that organizes the normalized output for each desired text. Utilizing nested schemas enables the system to infer multiple normalized outputs within a single query.

4.2 Fine-Tuning RoBERTa base Transformer for NER

In this approach, we fine-tune a pre-trained RoBERTa base Transformer model to perform Named Entity Recognition (NER). Unlike prompt-based approaches, fine-tuning a transformer model leverages deep contextual embedding relative to the dataset.

4.2.1 Why RoBERTa?

RoBERTa (Robustly Optimized BERT Pretraining Approach) [4] is an optimized variant of BERT, trained with:

- Dynamic masking, ensuring exposure to a wider variety of training samples.
- Larger batch sizes and extended training steps improving model generalization.
- Byte-Pair Encoding (BPE) tokenizer, which better handles out-of-vocabulary (OOV) words.
- Multilingual inputs, allowing it to handle multiple languages effectively.

4.2.2 Model Fine-Tuning & Postprocessing for Entity Recognition

Once the data is correctly formatted, the fine-tuning process becomes straightforward. The open-source library SpaCy [2] is leveraged for this task, providing a flexible framework for training custom models.

After training the model, it can be used to predict "PERSON" entities within the text. The final step involves post-processing these predictions, where the identified entities are combined into a single, normalized format by joining them into one string.

5 Experiments & Results

This section presents the experiments conducted and the relevant results for the implementations outlined in the previous section. It is important to note that, due to time constraints, no further optimization was performed. The primary objective is to demonstrate that these methods are capable of solving the given task and possess potential for future improvements.

5.1 Evaluating Prompt-based approach

To evaluate the proposed approach, a subset of the dataframe was used, excluding the samples utilized for creating the vector storage. The validation dataset consists of 1,000 samples. The decision to evaluate the model on only a small portion of the remaining dataset, rather than the entire set of 9,900 data points, was made to limit the cost of using the API for all remaining samples.

To evaluate the performance with used exact string matching between the ground truths (CLEAN_TEXT column) and the predicted response from the pipeline. For this experiment, the resulting accuracy was 83.94%.

Although this performance is already promising, there are a few important considerations to note. The dataset used for this task contains some erroneous values (outliers) that, as previously mentioned, are difficult to exclude due to time limitations. For instance, in cases involving non-Latin characters, the ground truth of the dataset was marked as NaN, even though writer information was indeed present. While the proposed model handles these cases correctly, the comparison with the erroneous ground truth lowers the accuracy of the validation. While this is not always the case, it represents a significant portion of the dataset and, as such, warrants mention.

5.2 NER Training

The model was trained for a total of 87 epochs, with nearly every epoch showing an increase in accuracy and a decrease in loss. The entire training process took approximately 450 minutes, averaging around 5 minutes per epoch.

Figure 4 presents the results from fine-tuning a Named Entity Recognition (NER) model with spaCy. The metrics recorded include loss values for both the tranformer and NER, as well as performance metrics such as precision (ENTS_P), recall (ENTS_R), and the overall F1-score (ENTS_F).

The results show significant improvements in the model's performance as the training progresses. In the early epochs, the model has a relatively high loss value of 1548.26 for the transformer and 390.00 for the NER, with an F-score of only 43.38, indicating poor entity recognition. However, as the training continues, the loss values decrease significantly, especially for the NER task, reaching a minimum of 42.26 and 59.24, respectively.

The precision (ENT_P) and recall (ENT_R) values also improve steadily as the model learns. At epoch 0, precision is 51.87 and recall is 37.27, reaching 96.74 and 98.24 at the end of the training, respectively. The precision and recall metrics indicate the model's growing ability to correctly identify and capture entities, with precision ensuring fewer false positives and recall ensuring fewer false negatives.

The harmonic mean of precision and recall (ENTS_F), follows a similar upward trajectory, reaching 97.48 by epoch 87. Finaly, he overall performance score is peaking at 0.98 by the later epochs.

In summary, the model demonstrates excellent performance across all metrics as it progresses through the training epochs. This indicates that the fine-tuning process successfully enabled the model to refine its entity recognition capabilities. The loss values stabilize, and both precision and recall are balanced, leading to an efficient and accurate NER model.

5.3 NER Validation

The evaluation of the Named Entity Recognition (NER) model on the test dataset yielded promising results. The model achieved a precision of 97.20%. This suggests that the model is effective at minimizing false positives, meaning it rarely classifies non-entity words as entities. Furthermore, the recall of 98.32% indicates that the model successfully identified 98.32% of all actual "PERSON" entities in the dataset.

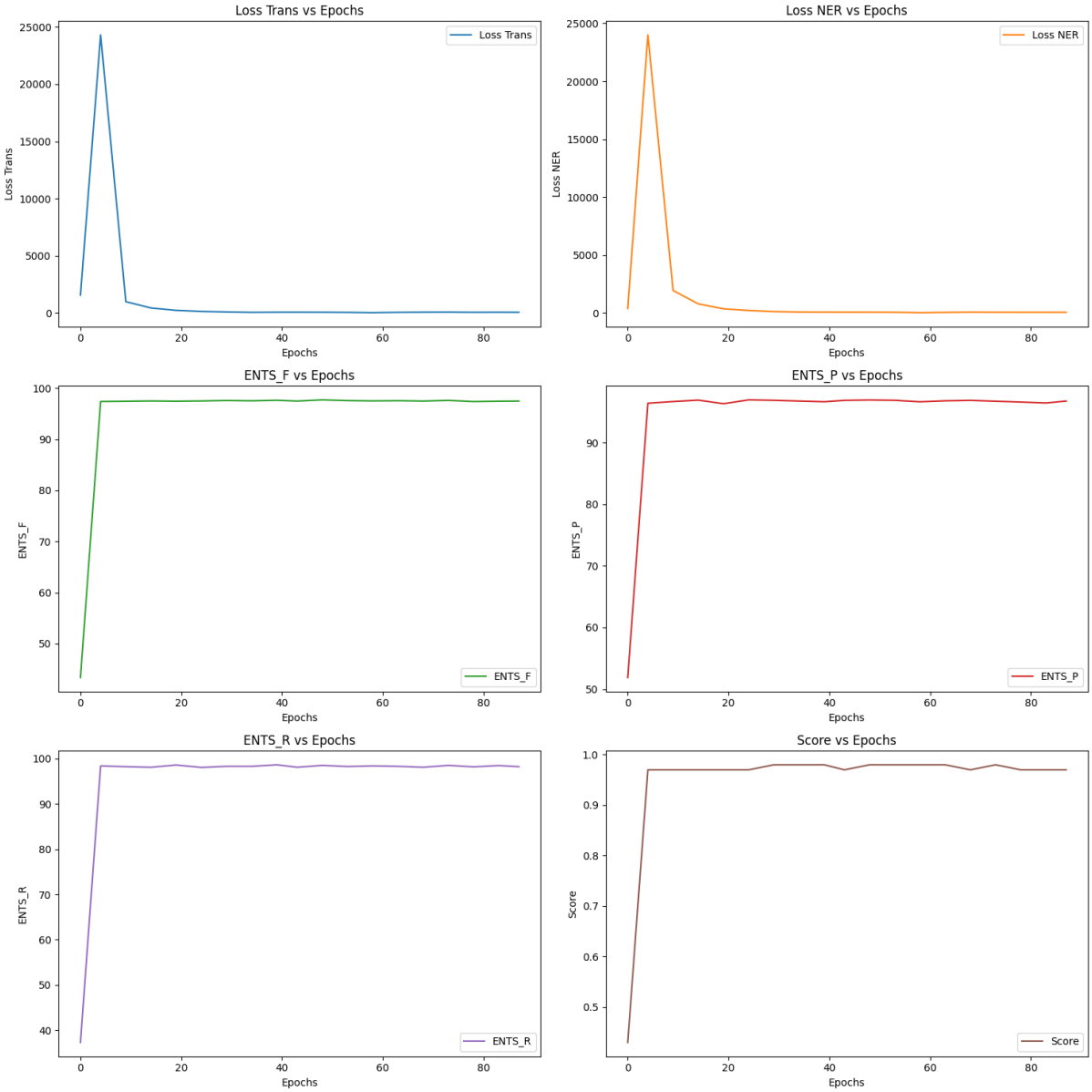

Figure 4: Training results for NER transformer.

This combination of high precision and recall suggests the model performs effectively also on unseen data. A summary of the aforementioned metrics is shown in Table 2.

Entity Type	**Precision (P)**	**Recall (R)**	**F1-Score (F)**
PERSON	97.20	98.32	97.76

Table 2: NER Performance on the test set.

Both models are integrated into a user interface that allows users to select their preferred model, input raw text, and apply normalization. Figure 5 illustrates the developed application.

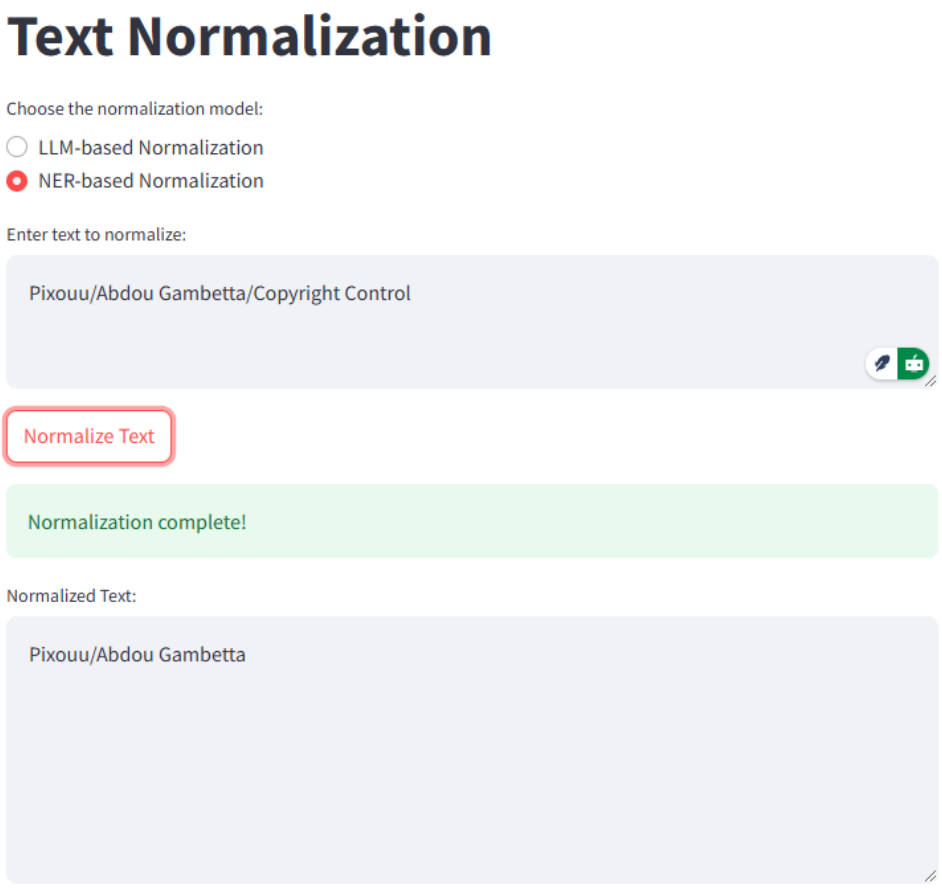

Figure 5: User interface for model selection and text normalization.

6 Conclusions

In this report we presented two end-to-end implementations for solving the task of normalizing raw composition writer information leveraging rule-based techniques combined with NLP. The results shown that both proposed approaches are suitable and perform well in the asked task. Although this is a proof-of-concept assignment with limited time, optimizing and fine-tuning the solutions can result in more accurate predictions.

To further enhance the implementation, more robust and effective deterministic rules could be applied to simplify the input to the models. Additionally, the original dataset could be filtered to exclude outliers, allowing for more representative metrics and results to be observed.

For the prompt-based solution, optimizing and fine-tuning the prompt would help better articulate the task and minimize small errors in the output. Another potential improvement involves the design of the vector storage and the selection of examples. By targeting specific edge cases and challenging data for the text normalization task, the database could be enriched to improve the model's inference. Currently, the selection process is random. Moreover, the database could be constructed once rather than each time a query is made to the LLM, which would reduce inference time and allow for a larger database with more examples.

In the case of the NER solution, optimizing the quality of the labeled dataframe is crucial. Currently, the labeling process is automated, assigning a single entity. However, as observed in the dataset, the names in the raw and clean columns appear in different formats. Manually labeling the dataset with additional entity types—such as first names, last names, or even frequently occurring but irrelevant words—could enable the model to make more meaningful predictions. This would also improve the flexibility of the model when constructing the final normalized text.

References

[1] Chroma. The ai-native open-source embedding database, 2024. https://www.trychroma.com/.

[2] Matthew Honnibal and Ines Montani. spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. To appear, 2017.

[3] LangChain. Openai text embeddings integration., 2024. https://python.langchain.com/docs/integrations/text_embedd

[4] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized BERT pretraining approach, 2019.

[5] OpenAI. Gpt-4o, 2024. https://platform.openai.com/docs/models/gpt-4o.