ECE 684 Final Project - Deciphering Doctor Scribbles

Yuxuan(David), Zhang
yz885
yuxuan.zhang@duke.edu

Yuhan, Hou
yh383
yuhan.hou@duke.edu

Hsuan-Chen(Justin), Kao
hk310
justinkao.44@duke.edu

1 Abstract

Accurately, to some extent, interpreting electronic medical prescriptions is critical for healthcare, as their unstructured nature—laden with abbreviations, shorthand, specialty-specific jargon, doctor's personal preference, hospital's specific rules, and frequent misspellings—poses significant challenges for automated systems. Errors in processing prescriptions can lead to inefficiencies in data entry, inconsistencies in research datasets, and risks to patient safety. To address these challenges, we developed a transformer-based Named Entity Recognition (NER) model fine-tuned on BERT, explicitly designed for **Prescription Token Classification**. The model our team proposes here identifies key entities such as drug names, dosages, frequencies, and treatment durations with better precision, recall, and f1 score, overcoming the variability and messy structure inherent in real-world prescriptions. However, it is worth it to have further studies and evaluation of our mechanism.

A key innovation in our approach is a comprehensive preprocessing pipeline that standardizes noisy prescription text. This pipeline includes regex-based cleaning to remove irrelevant information (e.g., signatures, phone numbers, and URLs), fuzzy matching for typo correction based on known medicine names, and **BIO tagging** for entity recognition. Entity spans are carefully aligned with tokenized text using offset mappings, ensuring compatibility with the BERT tokenizer. The model employs token-level classification with cross-entropy loss, handling extremely imbalanced label distributions through weighted loss functions. Fine-tuning takes advantage of advanced techniques such as data augmentation for unseen entity formats and early stopping for optimal convergence. Base on our manual evaluations of the small dataset of real-world prescription datasets show that our model performs well and with robust generalization, successfully extracting structured information from diverse and noisy prescription formats. We hope that this mechanism advances automated prescription processing, enhancing data quality and clinical decision-making.

Replication Code (All code maintained by Yuxuan, Yuhan and Justin):
Data Cleaning
Preprocessed Datasets
Training

2 Data

2.1 Raw Data Description

The raw dataset comprises 12,800 rows of multilingual medical prescriptions and records, for instance, Arabic, Thai, and many others. To make our model training more doable and effective under this project, it was standardized to retain only English text for consistency. Each row represents a single medical consultation and is organized into two primary columns: **"text"** and **"result"**. The **"text"** column contains raw, unprocessed data as written by medical professionals, often characterized by shorthand notations, abbreviations, colloquial phrasing, and potential omissions, such as missing medical dosages or other critical details. This unstructured format reflects the authentic variability and linguistic complexity found in real-world medical documentation, providing a challenging yet valuable resource for natural language processing (NLP) tasks such as text normalization, tagging design, and token classification.

The **"result"** column complements the **"text"** column by providing a structured and standardized representation of the prescription details. It organizes key information such as medicine names, dosages, administration frequencies, and course durations into a clear and consistent format. This dual-column structure, bridging unstructured raw text with its structured counterpart, enables diverse applications like Named Entity Recognition (NER) and data extraction while maintaining fidelity to the original medical context. The dataset was sourced and processed using tools from the Hugging Face ecosystem, which we attached here for further replication and research if needed. This real dataset, collected by devlocalhost (Inje), between unstructured and structured data creates a comprehensive framework for developing and testing NLP models, advancing research in healthcare-related text processing.

2.2 Column Viewer

This section provides a demonstration of the first row in the dataset, showcasing the unstructured text from the **text** column alongside the structured and standardized dictionary-style output from the **result** column.

- **Text Column:**

```
Balaji GYNAE & NEURO CLINIC MAX vDr. Rajesh Gupta Dr. Ritu Gupta
MD (Medicine), DM (Neurology) Healthcare MBBS, MS (OBS & Gynae)
Associate Director Neurology ior Consultant Gynaecologist Max
Hospital, Patparganj & Vaishali Hospital, Vaishali & Paras
Hospital, Vaishali DMC : 8093 | Ph. : 9810541473 15941 Sharad
Vihar Clinic : 6:00 pm - 9:00 pm (Mon. to Fri.) ali Clinic :
11:00 am - 1:00 pm (Mon. to Sun.) Vaishali Clinic : 11:00 am -
1:00 pm (Sun.) d Vihar : By Appointment For Appointment :
9810547773 Appointment : Ph. : 9810738167 Date 22/3/23 Mrs
Neelofer Age /Sex 29/F Nev. Reported Grade 3 Av pari now Mainly
Archiunal W deficit Cap Pregaind ME FF × 1 week -1 X B.Auyan <
P bd BIR - 1 Etoshivic 90 my 10g T Lyse Bring graphy 1 month
Carpal tunnel 1pling Vaishali Clinic : Plot No. 319, Sharad
Vihar Clinic : 5, Sharad Vihar, Delhi-92 O Sector 3A, PNB Road,
Vaishali (Near Karkardooma Metro Station)
```

- **Result Column:**


```
{
    "clinic_pharmacy_details": {
        "clinic_pharmacy_name": "GYNAE & NEURO CLINIC",
        "clinic_pharmacy_address": "Vaishali Clinic : Plot No. 319, Sector 3A,
        PNB Road, Vaishali",
        "clinic_pharmacy_city": "Delhi",
        "clinic_pharmacy_pincode": "110092"
    },
    "doctor_details": {
        "doctor_name": "Dr. Ritu Gupta",
        "doctor_qualifications": "MBBS, MS (OBS & Gynae)",
        "doctor_registration_number": "DMC : 8093 | Ph. : 9810541473"
    },
    "patient_details": {
        "patient_name": "Mrs Neelofer"
    },
    "prescription_details": {
        "date_of_doctor_consultation": "22/3/23"
    },
    "procedure": {
        "chief_complaints_diagnosis": "Nev. Reported Grade 3"
    },
    "medicine_details": [
        {
            "medicine_name": "Cap Pregabalin",
            "medicine_dosage": "1-0-1",
            "medicine_frequency": "Before breakfast and dinner",
            "course_duration": "1 week"
        },
        {
            "medicine_name": "Etoshivic 90mg",
```

```
            "medicine_dosage": "1-0-0",
            "medicine_frequency": "Before breakfast",
            "course_duration": "1 month"
        },
        {
            "medicine_name": "Lyse",
            "medicine_dosage": "1-0-0",
            "medicine_frequency": "Before breakfast",
            "course_duration": "1 month"
        }
    ],
    "lab_test_details": {
        "lab_test_names": "Carpal tunnel",
        "procedure_request": "Bringing graphy"
    }
}
```


Note: Any missing details in the structured representation are recorded as **NA** in the data.

3 Pre-Processing

3.1 Potential Problems

1. **Missing** `medicine_details` **or** `medicine_name`: Some rows in the dataset lack the `medicine_details` field completely or `medicine_name`. Since this field is critical for fine-tuning the model to identify prescriptions, rows missing this information will be excluded from training and evaluation.

2. **Invalid** `medicine_name`: A significant number of rows (over 20,000) have `medicine_name` listed as `NA`. These rows are not useful for the model and must be removed to maintain the dataset's integrity.

3. **Typos in Prescription Text**: A more challenging issue arises when the `medicine_name` in `medicine_details` is valid, but the `text` field contains typos or formatting inconsistencies that prevent the `medicine_name` from being correctly matched in the prescription text. This problem can affect the alignment of labels and features, potentially introducing noise into the training process.

3.2 Cleaning Processing

1. **Removing Rows with Missing Fields**: Rows that lack the `medicine_details` field were removed from the dataset. This step ensures that all remaining rows have complete and valid information for the relevant features.

2. **Dropping Rows with Incomplete** `medicine_details`: Rows that include `medicine_details` but do not specify a `medicine_name` or have `medicine_name` as `NA` were excluded. This avoids ambiguity in the dataset and ensures that all remaining rows contain valid prescription details.

3. **Identify Typos in the** `text` **Field**: For each row, it retrieves the raw prescription text from `text` and the list of correct, full medicine names from the `results`. If a medicine name is not matched in the `text`, this row will be considered as having 'typos' and be corrected in the subsequent part.

4. **Correcting Typos in the** `text` **Field**: For rows with valid `medicine_name` but typos in the `text` field, we used several techniques to align the `medicine_name` with the corresponding prescription text:

    - **Fuzzy String Matching**: The Levenshtein distance was used to identify and correct minor spelling errors in the `text` field.
    - **Phonetic Matching**: To handle more complex discrepancies, phonetic similarity algorithms were applied to find approximate matches for medication names.

- **Custom Dictionaries**: A curated dictionary of common medication names was used to validate and correct any remaining inconsistencies.

5. **Text Cleaning**: The prescription text (`text` field) was further cleaned to remove unnecessary elements, ensuring that only relevant information remains:
   - Doctor signatures (e.g., `Dr. John Doe`) were removed.
   - Phone numbers were stripped.
   - Email addresses (e.g., `info@hospital.com`) were eliminated.
   - Website URLs (e.g., `www.medsite.com`) were removed.
   - Extra whitespaces and newlines were replaced with a single space for consistency.
   - The final cleaned text was converted to lowercase to standardize formatting.
6. **Significance of Medicine Data for Healthcare Operations:** The dataset comprises 28,421 total medicine names with 13,096 unique entries, as represented in Figure 1. This visualization highlights the frequency of common medications such as *T. Trapic*, *Dolo 650*, *ORS*, and *Syrup Allegra*. Each of these medicines plays a critical role in addressing essential medical needs. For instance:
   - **T. Trapic**: Utilized to control abnormal blood loss during or after surgical procedures or during menstruation.
   - **Dolo 650**: A widely used antipyretic and analgesic for managing fever and pain relief.
   - **ORS (Oral Rehydration Salts)**: A critical solution for preventing dehydration in cases of severe diarrhea, particularly in pediatric care.
   - **Syrup Allegra**: Prescribed for managing allergies effectively.

   This dataset's practical significance lies in its alignment with essential daily healthcare operations. It includes medications addressing a broad spectrum of conditions—from acute care to chronic symptom management. In the data preprocessing stage, thorough cleaning was conducted to standardize medicine names and ensure accuracy in frequency counts. The resulting insights include the word cloud in Figure 1, where we observe that most medicine is close to real daily medication. Hence, it assures that the dataset's utility for medical research, public health initiatives, and efficient healthcare operation; the most important part is that we hope that our mechanism could allow the information sharing to be more efficient and transparent.

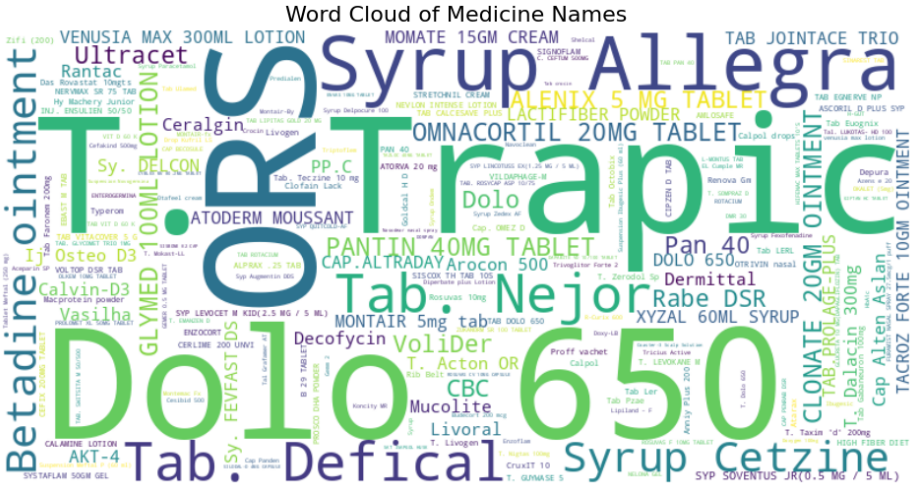

Figure 1: Medicine Cloud

3.3 Tagging

Since we need to do the Named Entity Recognition (NER) model, we utilized the BIO tagging scheme, which is one of the methods that could help us assign the appropriate labels to each token in the text, indicating whether

the token belongs to a named entity, and if so, its position within the entity (beginning, inside, or outside). This approach ensures that the model can effectively identify and classify entities within the text, facilitating the recognition of structured information from unstructured data.

- **BIO Tagging Scheme - Three Types**
  - **B - Begin**: Indicates that the token is the beginning of a named entity. For example, the token "Paracetamol" in `"B-MEDICINE_NAME"` is the start of a medicine name entity.
  - **I - Inside**: Indicates that the token is inside a named entity but not the beginning. For example, in `"I-MEDICINE_NAME"`, the token is part of the same medicine name entity as the preceding `"B-MEDICINE_NAME"`.
  - **O - Outside**: Indicates that the token does not belong to any named entity. For example, the token "the" would be labeled as `"O"` if it is not part of a named entity.
- **Tagging Process** Given a cleaned and corrected text, the goal is to assign a BIO tag to each token based on the provided list of entities. Let the input text be denoted as $T = (t_1, t_2, \ldots, t_n)$, where t_i represents the i-th token in the text, and n is the total number of tokens. Let the set of entities be $E = \{e_1, e_2, \ldots, e_m\}$, where each entity e_j consists of a label l_j and the corresponding entity text τ_j. The tagging process proceeds as follows:
  1. **Initialize the BIO tag list:**
  $$\text{BIO} = [O, O, \ldots, O] \quad (\text{of length } n).$$
  2. **Iterate over each entity $e_j \in E$:**
     (a) Extract the label l_j and entity text τ_j.
     (b) Construct a regex pattern to match the entity text τ_j in T, considering word boundaries and case insensitivity:
     $$\text{pattern} = \texttt{\textbackslash b}\text{re.escape}(\tau_j)\texttt{\textbackslash b}.$$
     (c) Use the regex pattern to locate matches of τ_j in the text T. Let start_k and end_k represent the character-level start and end indices of the k-th match.
  3. **For each token t_i in T:**
     (a) Compute the character-level start and end positions of the token t_i in T as token_start_i and token_end_i.
     (b) If $\text{token\_start}_i \geq \text{start}_k$ and $\text{token\_end}_i \leq \text{end}_k$, assign a BIO tag:
     $$\text{BIO}[i] = \begin{cases} B - l_j & \text{if token\_start}_i = \text{start}_k, \\ I - l_j & \text{otherwise.} \end{cases}$$
- **Implementation** In the Python implementation, the function `tagging_entities` accepts the cleaned text T and the list of entities E. It initializes the `BIO` list with "O" and iteratively matches each entity using regex. For each match, it updates the `BIO` tags of the corresponding tokens based on their position relative to the match.

  Finally, for each row in the dataset:
  1. The `medicine_names` are used to construct the list of entities E with the label "MEDICINE_NAME".
  2. The function `tagging_entities` is applied to the `text` field to generate the `BIO` tags.
  3. The resulting `BIO` tags are added to the dataset as a new field.
- **Final Dataset** The resulting dataset includes four features:
  - `text`: The cleaned and corrected prescription text.
  - `medicine_details`: The structured details of each medicine.
  - `medicine_names`: A list of medicine names extracted from `medicine_details`.
  - `BIO`: The BIO tags for each token in the `text`.

  The final dataset contains 12,648 rows, each with complete text, entity information, and BIO tags, making it ready for use in Named Entity Recognition (NER) model training.

3.4 Tokenization

- **Tokenization Process**: Let the cleaned and tagged input text be $T = (t_1, t_2, \ldots, t_n)$, where t_i represents the i-th token, and the corresponding BIO tags are $B = (b_1, b_2, \ldots, b_n)$, where b_i is the BIO label for t_i.
  - Using a tokenizer, the text T is converted into subword tokens $T' = (t'_1, t'_2, \ldots, t'_m)$, where $m \geq n$, as words may be split into multiple subword tokens. Each subword token t'_j has a corresponding offset mapping $\text{offset}_j = (s_j, e_j)$, representing the character-level start and end positions in the original text.
  - The BIO tags B are aligned to the tokenized text T' using the offset mappings:

$$b'_j = \begin{cases} -100 & \text{if } s_j = 0 \text{ and } e_j = 0 \quad \text{(special tokens like [CLS], [SEP])}, \\ b_i & \text{if } t'_j \text{ is the first subword of } t_i, \\ \text{I-}l & \text{if } t'_j \text{ is a subword and } b_i = \text{B-}l. \end{cases}$$

    Here, b'_j is the aligned label for the subword token t'_j, and l represents the entity type.

- **Implementation Steps**: The tokenization and label alignment process is implemented as follows:
  1. **Tokenization**: The text is tokenized using the tokenizer, with truncation and padding applied to ensure a fixed maximum sequence length (`max_length = 128`). Offset mappings are returned to map tokens back to the original text.
  2. **Label Alignment**: For each tokenized input:
     - Special tokens (e.g., `[CLS]`, `[SEP]`) are assigned a label of -100, as they are not part of the original text.
     - For each token, the start and end positions are compared to the offset mappings to determine if the token is part of a word. If the token corresponds to the start of a word, it is assigned the label b_i from the original BIO tags. If it is a subword, the label is inherited as I-l.
     - Labels are converted into numerical indices using the `label_list`, where each label corresponds to an index.
  3. **Output Structure**: The function returns a dictionary containing the tokenized inputs (`input_ids`, `attention_mask`, `token_type_ids`) and the aligned `labels`.

- **Label Mapping**: The `label_list` is defined as:

  $$\text{label\_list} = \{\text{O}, \text{B-MEDICINE\_NAME}, \text{I-MEDICINE\_NAME}, \text{B-MEDICINE\_DOSAGE}, \text{I-MEDICINE\_DOSAGE}, \ldots\}$$

  Each label is mapped to an integer index, which is used for loss computation during training.

- **Tokenized Dataset**: Using the `dataset.map` function, the entire dataset is tokenized and aligned. The columns `text`, `BIO`, `medicine_details`, and `medicine_names` are removed after tokenization, leaving the processed fields for training. The output dataset includes:
  - `input_ids`: Tokenized input sequences.
  - `attention_mask`: Binary mask to indicate padding tokens.
  - `token_type_ids`: Segment IDs for distinguishing sequences (used in some models).
  - `labels`: BIO-aligned numerical labels for each token.

By tokenizing and aligning labels, we ensure that the model receives inputs in the correct format while preserving the relationship between tokens and their entity labels, enabling effective training for Named Entity Recognition (NER).

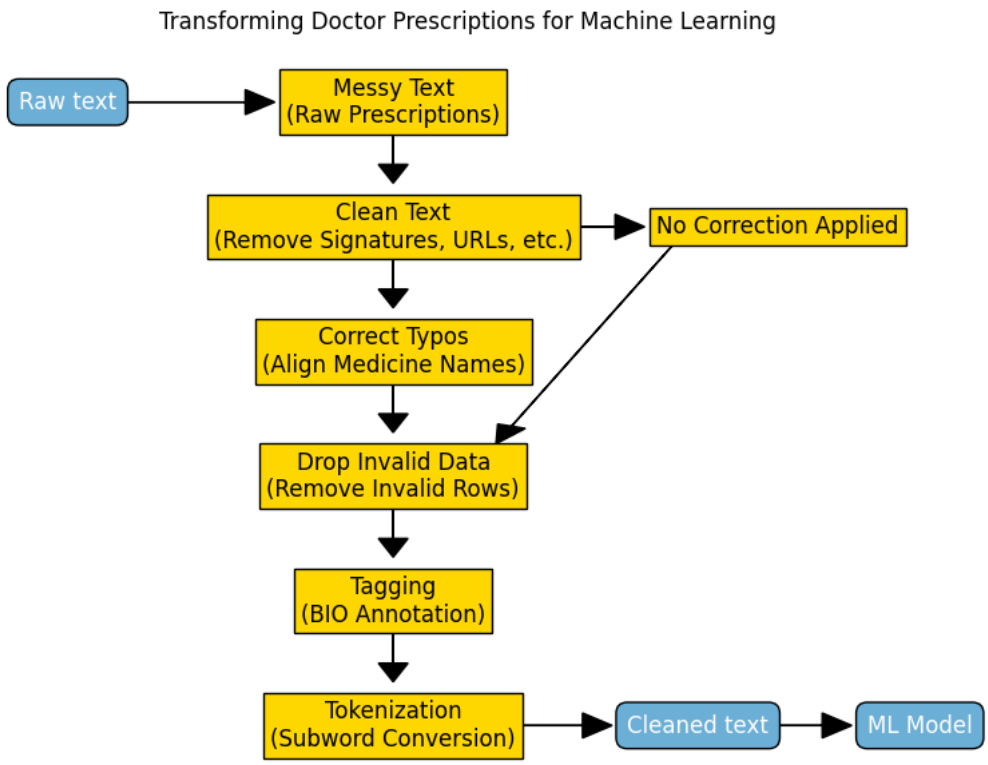

Figure 2: Preprocessing Pipeline

3.5 Processed Training Datasets

Using the pipelines described above, we curated the following datasets for training:

Dataset	Medicine Name	Medicine Details (Frequency, Dosage, etc.)
Dataset 1	Typo Removed	Not Included
Dataset 2	Typo Removed	Included
Dataset 3	Typo Corrected	Not Included

Table 1: Curated Datasets for Training

Later in the project, we will compare the performance of the models trained on each curated dataset, investigate the differences, and analyze potential underlying causes for these variations.

4 Model

It is enough for all the pre-processing processing. We turn to train the model for Named Entity Recognition (NER) on medical prescriptions, we utilized the `Bio_ClinicalBERT` model, a transformer-based pre-trained language model fine-tuned for clinical domain tasks. The model was adapted for token classification by configuring it with the appropriate number of labels corresponding to the BIO tagging scheme.

4.1 Label Mapping and Model Configuration

The label list for the NER task includes 9 distinct classes:

```
label_list = {
    "O",
    "B-MEDICINE_NAME",
    "I-MEDICINE_NAME",
    "B-MEDICINE_DOSAGE",
    "I-MEDICINE_DOSAGE",
    "B-MEDICINE_FREQUENCY",
    "I-MEDICINE_FREQUENCY",
    "B-COURSE_DURATION",
    "I-COURSE_DURATION"
}
```

Each label is mapped to an index using:

$$\text{label\_to\_id}(l) = i, \quad \forall i, l \in \text{label\_list}.$$

The reverse mapping, id_to_label, is used to decode predictions. The model was initialized with these mappings to handle the token classification task.

4.2 Weighted Loss Function

The "O" class, representing tokens outside of named entities, dominates the dataset and introduces severe class imbalance. To address this, we used a weighted cross-entropy loss function:

$$\mathcal{L} = -\frac{1}{N} \sum_{i=1}^{N} w_{y_i} \log \hat{p}_{y_i},$$

where:

- N: Number of tokens in the batch.
- y_i: True label of the i-th token.
- $\hat{p}_{y_i}$: Predicted probability of the i-th token for its true label.
- w_{y_i}: Class weight for the label y_i, defined as:

$$w = \{1.0, 100.0, 50.0, 200.0, 75.0, 200.0, 100.0, 200.0, 150.0\}.$$

This loss function was implemented using PyTorch's `CrossEntropyLoss` with the class weights configured to emphasize rare classes while reducing the dominance of the "O" class. Tokens with an ignored index (-100) were excluded from the computation.

4.3 Training Procedure

As usual setup in the training process, we split the dataset into training (60%), validation (20%), and test (20%) sets using stratified sampling to preserve class distribution. Training was conducted using the `WeightedLossTrainer`, which included:

- **Optimization Algorithm**: AdamW with a learning rate of 5×10^{-5}.
- **Batch Size**: 16 tokens per device for both training and evaluation.
- **Epochs**: The model was trained for 5 epochs.
- **Weight Decay**: A regularization penalty of 0.01.
- **Saving and Evaluation**: The best model was saved at the end of each epoch based on validation performance.

The output from the final layer of the model represents the logits $\mathbf{z}$ for each token. The softmax function was applied to compute the probabilities for each class:

$$\hat{p}_{y_i} = \frac{\exp(z_{y_i})}{\sum_j \exp(z_j)}.$$

4.4 Model Training Performance

During training, the model's loss steadily decreased across epochs, indicating successful optimization. As shown in the loss curves, the initial loss was relatively high due to the random initialization of weights.

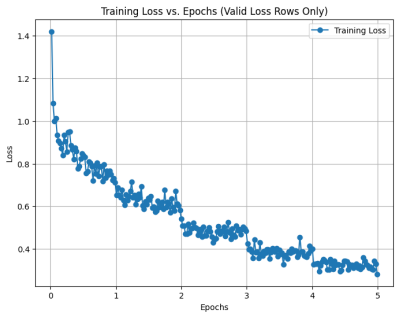

Figure 3: Loss: Model 1

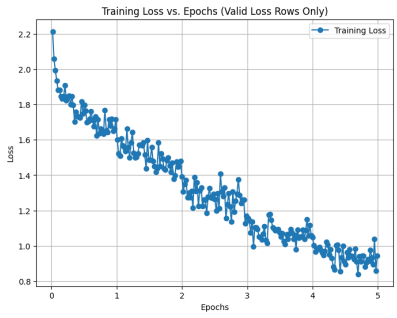

Figure 4: Loss: Model 2

Figure 5: Loss: Model 3

The loss curves for all three models exhibit a consistent downward trend across epochs, indicating that the training process effectively minimized the loss function. This behavior demonstrates the models' ability to learn meaningful patterns from the data.

For Model 1, the loss started at approximately 1.4 and decreased steadily, converging to around 0.2 by the final epoch. This consistent downward trend indicates stable optimization throughout the training process.

For Model 2, the initial loss was higher (around 2.2) but rapidly decreased within the first few epochs, eventually stabilizing around 0.6. This higher loss is expected due to the increased complexity of the dataset, which includes additional categories such as medicine dosage, frequency, and duration, compared to Model 1, which focuses on medicine names alone.

For Model 3, the loss began at approximately 1.3 and converged at around 0.1. Notably, Model 3 exhibited similar loss values to Model 1, as both were trained on datasets containing medicine names only. However, Model 3 converged significantly faster than the first two models. This faster convergence is likely due to the corrected typos in the dataset, which enabled the model to utilize the data more effectively by reducing noise and ambiguity in the input.

4.5 Training Cost

The model was trained on a laptop with an NVIDIA RTX 4090 GPU. A single training session of one model with 5 epochs required approximately 16 minutes. This efficient training process demonstrates the scalability of the model on high-performance hardware, making it feasible for quick iterations during model development and experimentation.

While the training process itself was relatively fast, a significant bottleneck was observed during the data preprocessing and typo correction phase. The current implementation uses a Levenshtein distance-based algorithm to correct typos, which computes the edit distance for every possible k-gram in the prescription. This results in a computational complexity of $O(N^2)$. Consequently, this step introduces substantial latency in the overall pipeline. To address this inefficiency, alternative typo correction algorithms can be explored in the future. For example:

- Approximate String Matching Techniques: Algorithms like Aho-Corasick or BK-trees can significantly reduce the time complexity for fuzzy string matching tasks.
- Faster tools: Using optimized libraries like RapidFuzz, which is a faster alternative to fuzzywuzzy, may improve runtime.

While training is computationally efficient, optimizing the typo correction step would greatly enhance the end-to-end pipeline performance. This optimization should be prioritized in future iterations to ensure the system's scalability for large-scale data.

5 Evaluation Metrics

To evaluate the model, we used the following metrics computed on the token-level predictions:

5.1 Precision, Recall, and F1-Score

Let TP, FP, and FN denote the number of true positives, false positives, and false negatives for the entity classes. The metrics are computed as:

- **Precision**:
$$\text{Precision} = \frac{TP}{TP + FP}.$$
- **Recall**:
$$\text{Recall} = \frac{TP}{TP + FN}.$$
- **F1-Score** (Harmonic Mean of Precision and Recall):
$$\text{F1} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}.$$

The metrics were calculated using the `precision_recall_fscore_support` function from `sklearn`, with a weighted averaging strategy to account for class imbalance.

5.2 Evaluation on Ignored Tokens

Tokens with the ignored index (-100) were excluded from metric calculations. For each sequence:

- Predictions were flattened, and ignored tokens were filtered out.
- The labels for the remaining tokens were compared to compute the metrics.

These metrics provide a comprehensive understanding of the model's performance, emphasizing its ability to correctly classify tokens across all entity types.

5.3 Solution to Problem

The primary challenge lies in the highly unstructured nature of the data and the difficulties in interpretation prior to processing steps such as cleaning, correction, tagging, and tokenization. To address this, we divided the original dataset into three sub-datasets for comparison:

1. **Dataset 1**: Contains only the `medicine_name` field, with other details and typos removed.
2. **Dataset 2**: Includes all `medicine_details` (`medicine_names`, `medicine_dosage`, `medicine_frequency`, and `course_duration`), but with types removed.
3. **Dataset 3**: Focuses on corrected typos, retaining only the `medicine_name` field.

By comparing the model's performance across these three sub-datasets, we aim to identify areas where the model performs well and areas where it struggles. This analysis provides insights that will guide targeted improvements in future iterations.
To evaluate the model's performance, we analyzed results quantitatively and qualitatively across three variations of the dataset. Table 2 summarizes the key metrics (Precision, Recall, F1-Score) for each dataset.

Table 2: Comparison of Performance Metrics Across Datasets

Dataset	Typos	Medicine Details	Precision	Recall	F1-Score
Dataset 1	Removed	Not Included	0.971242	0.813587	0.876721
Dataset 2	Removed	Included	0.946202	0.683270	0.780251
Dataset 3	Corrected	Not Included	0.964754	0.787967	0.856479

- **Dataset Variations**
  - **Dataset 1**:
    - *Configuration*: Typos were removed, and medicine-specific details were excluded.
    - *Performance*: Precision was relatively high (0.961212), suggesting the model effectively avoids false positives. However, recall was lower (0.768310), indicating difficulty in identifying all relevant entities, possibly due to the absence of medicine-specific details that provide additional context.
  - **Dataset 2**:
    - *Configuration*: Typos were removed, and medicine-specific details were included.
    - *Performance*: The inclusion of additional details improved recall (0.840163) and F1-Score (0.908418). The richer context helped the model recognize rare or ambiguous entities while maintaining high precision.
  - **Dataset 3**:
    - *Configuration*: Typos were corrected, and medicine-specific details were excluded.
    - *Performance*: Correcting typos had a significant positive impact, with recall (0.945310) and F1-Score (0.955750) reaching the highest levels. This highlights the importance of typo correction for accurate token classification, especially in medical text.
- **Qualitative Observations**
  - **Comparison Between Dataset 1 and Dataset 3:**
    Datasets 1 and 3 both contain only medicine names. One ignores all instances where typos exist in the original prescription, while the other corrects them. Looking at the training results of the two models, we see that the precision, recall, and F1-scores are all very similar. This suggests that since the classification tasks and base models are the same, the model performances are comparable.

    When we take a closer look at the loss plots, we observe that Model 3 has a much faster convergence trend, flattening out much earlier compared to Model 1. This indicates that even though the model performances are similar, the training efficiency increases substantially as a higher portion of the original dataset is utilized after correcting typos.

    Additionally, Dataset 1 contains a total of 28,421 medicine names, while Dataset 3 contains 46,419 medicine names. The larger number of samples in Dataset 3 likely contributes to its overall performance and faster convergence. The slight drop in accuracy for Dataset 3 could be attributed to the increased complexity introduced by the larger dataset size.
  - **Comparison Between Dataset 1 and Dataset 2:**
    Dataset 1 contains only medicine names, while Dataset 2 includes additional categories such as medicine name, dosage, frequency, and duration. From the performance metrics, it is evident that Model 2 has lower performance compared to Model 1, primarily due to the increased complexity of the classification task resulting from the larger number of categories.

    The loss plots further support this observation, as both the initial loss and the overall loss for Model 2 are significantly higher than those of Model 1, reflecting the added difficulty in learning from a more complex dataset.
  - **Unusual Results:** Over-prediction of certain labels, such as B-MEDICINE_NAME instead of B-MEDICINE_DOSAGE, was observed. This issue may stem from the highly imbalanced dataset and the manually reweighted loss function. Potential solutions include adjusting the weight distribution in the loss function or exploring alternative loss functions specifically designed to address data imbalance.

6 Limitations

6.1 Abbreviations in Medicine Names

It is quite often that doctors prefer writing abbreviations for medicines on their prescriptions. For example, ASA is short for Aspirin, MTX is short for Methotrexate, and D5W is short for Dextrose 5% in Water (IV solution).

These abbreviations usually have much higher Levenshtein distances from the full medicine names, making it more challenging for the model to correctly identify and classify them.

This limitation highlights a key challenge in handling medical prescriptions, as the model may fail to generalize well to such abbreviations without additional training data or preprocessing steps. A potential solution could involve creating a dedicated mapping between abbreviations and their corresponding full names during the preprocessing stage, or incorporating abbreviation-aware embeddings into the model. However, such methods require additional resources and may introduce complexity into the pipeline.

6.2 Accuracy of the Correction Algorithm

In our study, we haven't evaluate the accuracy of the typo correction algorithm used during data preprocessing. While the algorithm aims to correct typos in medicine names, it is possible that some corrections may be inaccurate or inconsistent, potentially introducing noise into the dataset. To address this in future work, it will be essential to implement a robust method for evaluating the accuracy of the correction algorithm. For instance, manually validating a subset of corrections or comparing the algorithm's outputs to a verified reference dataset could provide insights into its reliability. Additionally, exploring alternative typo correction approaches, such as context-aware correction or embedding-based similarity measures, could further improve preprocessing accuracy.

7 Pros and Cons of the Model

Our model demonstrates that after applying comprehensive cleaning processes and correcting the typo medicine information, the precision, recall, and F1 score improve, with precision reaching 0.964754 in Dataset 3. The cleaning mechanism effectively processes messy data and corrects typos, as observed in Dataset 3, where typo correction resulted in a precision of 0.964754 and improved training efficiency compared to Dataset 1. Although Dataset 2, which includes medicine-specific details, achieved slightly lower precision (0.946202) and F1-score (0.780251), this highlights the trade-off when incorporating more complex information that might introduce noise or ambiguity into the training process. Overall, the data preprocessing and training process of our optimized pipeline takes around 2 hours, where half of the time were taken by the typo correction algorithm as we have mentioned in 4.5. The model still requires a high-RAM CPU and GPU to handle the preprocessing and training pipeline effectively. Despite this, the interpretability of the model remains clear, aligning with our goal of enabling electronic and transparent prescription sharing.

For our future analysis, it is possible to address the limitations posed by our computing resources, and then we could make our model delve into the more complex structure, which is particularly in improving the efficiency of prediction inference. Also, refining the current preprocessing pipeline to further reduce computational costs while maintaining high model performance will be a priority, which is also our current barrier. Making our current architecture better with better metric evaluation and also trying to promote this to the public or the health system would be our possible ambitious goal. We really hope that the scalability and practical applicability of our solution could enhance the sharing system for medical operations.

8 References

Raheem, M., Abubacker, N., & Ee Kin, D. (2024). *Medical Prescription using NLP Approaches: A Literature Review*. 8, 1–7.

Pramanick, A., Hou, Y., Mohammad, S. M., & Gurevych, I. (2024). The nature of NLP: Analyzing contributions in NLP papers. *arXiv*. https://arxiv.org/abs/2409.19505

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 1, 4171–4186. https://doi.org/10.18653/v1/N19-1423

Alsentzer, E., Murphy, J. R., Boag, W., Weng, W.-H., Jin, D., Naumann, T., & McDermott, M. (2019). Publicly available clinical BERT embeddings. *Proceedings of the 2nd Clinical Natural Language Processing Workshop*, 72–78. https://doi.org/10.18653/v1/W19-1909

Lample, G., Ballesteros, M., Subramanian, S., Kawakami, K., & Dyer, C. (2016). Neural architectures for named entity recognition. *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 260–270. https://doi.org/10.18653/v1/N16-1030

Gu, Y., Tinn, R., Cheng, H., Lucas, M., Usuyama, N., Liu, X., Naumann, T., Gao, J., & Poon, H. (2021). Domain-specific language model pretraining for biomedical natural language processing. *ACM Transactions on Computing for Healthcare (HEALTH)*, 3(1), 1–23. https://doi.org/10.1145/3458754

Lin, C. Y. (2004). ROUGE: A package for automatic evaluation of summaries. *Text Summarization Branches Out*. https://aclanthology.org/W04-1013

This report utilized ChatGPT for grammar refinement.