Methodology Description

I. Data utilization

The competition organizer provided two datasets on Hugging Face: `darrow-ai/LegalLensNER` and `darrow-ai/LegalLensNER-SharedTask`.

- The first dataset, `LegalLensNER`, has two splits: a training split with 710 samples and a test split with 617 samples, totaling 1,327 samples.
- The second dataset, `LegalLensNER-SharedTask`, contains only a training split with 976 samples, all of which are included in the first dataset.

Given this overlap, we selected the 976 samples from the `LegalLensNER-SharedTask` dataset as our training set. The remaining 351 samples from the `LegalLensNER` dataset were used as our development set.
They are saved as `trainset_NER_LegalLens.csv` and `devset_NER_LegalLens.csv` files in our repository.

II. Data preprocessing

Our preprocessing pipeline involves several key classes and steps to prepare the data. Below is a detailed description of the preprocessing components:

1. InputExample Class

- **Purpose:** Represents a single example in the dataset, which includes the sentence (as a list of words) and the corresponding sequence of labels.
- **Attributes:**
  - `guid`: A unique identifier for the example.
  - `words`: A list of words in the sentence.
  - `labels`: A list of corresponding labels for each word.

2. InputFeatures Class

- **Purpose:** Converts the `InputExample` into features that are compatible with the language model, such as token IDs, masks, and segment IDs.
- **Attributes:**
  - `input_ids`: Token IDs generated from the tokenizer.
  - `input_mask`: A mask to distinguish between actual tokens and padding.
  - `segment_ids`: Used to identify different segments in LMs like BERT.
  - `predict_mask`: Indicates which tokens should be considered during prediction.
  - `label_ids`: The label IDs corresponding to the tokens.

3. NERLensDataProcessor Class

- **Purpose:** Handles the preparation and processing of NER data specific to the LegalLensNER dataset.

- **Functions:**
    - `get_train_examples`, `get_dev_examples`, `get_test_examples`: Load and process the training, development, and test datasets.
    - `get_labels`, `get_num_labels`, `get_label_map`: Provide label-related information, including label types, the number of labels, and a mapping from labels to IDs.
    - `_read_data`: Reads data from CSV or Excel files, with special handling for test files that lack labels.
    - `_create_examples`: Converts raw data into `InputExample` instances.

4. example2feature Function

- **Purpose:** Converts an `InputExample` into an `InputFeatures` instance, handling tokenization, label alignment, and sequence truncation.
- **Process:**
    - Tokenizes each word in the sentence.
    - Aligns the labels with the tokenized subwords. For any subword tokens, the label 'X' is assigned. For example, the word *'sheepmeat'* is tokenized into *'sheep'*, *'##me'*, and *'##at'*, and the labels for the subwords *'##me'* and *'##at'* would be 'X'.
    - Truncates the sequence if it exceeds the maximum sequence length.
    - Appends special tokens `[CLS]` and `[SEP]` to the sequence.
    - Converts tokens to their corresponding IDs and prepares masks and segment IDs.

5. NerDataset Class

- **Purpose:** Custom Dataset class to wrap the processed examples and facilitate their use..
- **Attributes:**
    - `examples`: The list of `InputExample` instances.
    - `tokenizer`: The tokenizer used to process the words.
    - `label_map`: Mapping from labels to IDs.
    - `max_seq_length`: Maximum sequence length for the model.
- **Functions:**
    - `__len__`: Returns the number of examples.
    - `__getitem__`: Converts an example into features using the `example2feature` function.
    - `pad`: Pads all sequences in a batch to the maximum sequence length in the batch.

6. Dataloader Preparation

- **Purpose:** Converts the dataset into dataloaders for training, validation, and testing.
- **Process:**

    - The `NerDataset` instances for training, validation, and testing are passed into PyTorch's `DataLoader`, with shuffling enabled for the training set.
    - The `pad` method ensures that all batches are padded consistently, allowing for efficient batching during training and evaluation.

III. Model's architecture, settings and hyperparameters:

1. Architecture

The model architecture is designed to leverage the power of a pre-trained language model (LM) combined with a Conditional Random Field (CRF) layer. This combination allows the model to effectively capture contextual information from the text and ensure that the output sequence of labels adheres to valid label transitions.

a. Pre-trained Language Model (LM)

- The language model, which is transformer-based, is responsible for generating contextualized word embeddings from the input text. These embeddings encapsulate rich semantic information by considering the entire context of each word in the sentence.
- The LM essentially serves as the emission component in the CRF model, producing a score (or feature) for each label at each position in the sequence

b. Conditional Random Field (CRF) Layer

- The CRF layer is used to model the dependencies between labels in the sequence, ensuring that the predicted label sequence follows valid transitions.
- **Details:**
    - The CRF layer operates on the features (emission scores) produced by the LM and considers potential label transitions.
    - It uses transition parameters to score the likelihood of moving from one label to another in the sequence.
    - The CRF optimizes the Maximum Likelihood Estimate (MLE) by calculating the probability of the correct label sequence given the input features.
    - It enforces constraints on label transitions, such as disallowing transitions to the start label or from the stop label, which helps in maintaining a valid sequence structure.

c. Model Components

- **Language Model Integration:**
    - The model begins by passing input sequences through the pre-trained language model to obtain contextualized embeddings
    - These embeddings are then fed into the linear layer to map them into the label space, creating the emission scores for the CRF.
- **Forward Algorithm:**
    - This algorithm performs a recursive computation to sum over all possible label sequences, calculating the total log-probability of the sequence under the model.

- **Viterbi Algorithm:**
  - This algorithm is used during inference to find the most likely sequence of labels by performing a max-product operation over the possible label transitions, effectively decoding the best label sequence.
- **Scoring:**
  - The model calculates the score of a given label sequence based on the emission scores and the transition probabilities. This score is then used in the loss function to compute the negative log-likelihood.
- **Negative Log-Likelihood (NLL) Loss:**
  - The training objective is to minimize the NLL loss, which measures the difference between the predicted label sequence probabilities and the actual label sequences in the training data.

2. Settings and hyperparameters

a. Model Settings

- **Language Model: Legal Longformer** (`lexlms/legal-longformer-base)`
  - This pre-trained language model is specialized for processing legal text and handling long sequences efficiently.
  - This LM showed the best performance on the Dev set **(0.9238 Macro-F1)**
- **Max Sequence Length:** `256`
  - The maximum number of tokens that the model will process for each input sequence.

2. Training Settings and Hyperparameters

- **Initial Learning Rate:** `5e-5`
  - The starting learning rate for fine-tuning the pre-trained language model
- **Learning Rate for CRF and Fully Connected Layer:** `8e-5`
  - A slightly higher learning rate specifically for the CRF layer and the fully connected layer, allowing them to adapt faster compared to the pre-trained components.
- **Weight Decay (Fine-Tuning):** `1e-5`
  - Regularization parameter to prevent overfitting during the fine-tuning of the language model.
- **Weight Decay (CRF and Fully Connected Layer):** `5e-6`
  - Regularization parameter applied to the CRF and fully connected layers to prevent overfitting.
  - The number of times the entire training dataset is passed through the model.
- **Gradient Accumulation Steps:** `1`
  - The number of steps to accumulate gradients before updating the model parameters. With a value of 1, the model parameters are updated after every batch.
- **Warmup Proportion:** `0.1`

    - The proportion of the total training steps used for learning rate warmup. During this phase, the learning rate gradually increases to its initial value to improve model stability early in training.
- **Batch Size:** `16`
- **Total Training Epochs:** `30`
    - The best epoch was the 18th

IV. Post-processing

In the competition datasets, tokens may be either full words or subwords. In our model, subwords are predicted with the 'X' label, which is not part of the official label set. To correct this, we perform a post-processing step where:

1. Any predicted 'X' label is replaced with the label of the preceding token.
2. If the preceding token is a 'B-' (beginning) label, the 'X' label is converted to the corresponding 'I-' (inside) label of the same type.