Unlocking Legal Text: Advancing Named Entity Recognition in the Legal Domain

Samit Kapadia sakapadia@ucsd.edu
Rupashi Sangal rsangal@ucsd.edu
Pradeep Yarlagadda s1yarlagadda@ucsd.edu
Harshad Sagi hsagi@ucsd.edu

1 Introduction

Named Entity Recognition (NER) is an important task in the domain of Natural Language Processing (NLP), centering around the identification and classification of named entities within textual data. NER systems are frequently utilized as the initial stage in various tasks such as question answering, information retrieval, co-reference resolution, topic modeling, etc (Yadav and Bethard, 2019). Despite recent advancements in this domain, its application is significantly challenging in specialized domains, like Legal documents (Cardellino et al., 2017), due to the distinctive structure of the legal documents and domain-specific keywords. On the other hand, developing NER-based systems for legal documents offer many advantages such as automatic information extraction, indexing, and summarizing. Also, NER-based systems can also streamline the process of regulatory compliance and monitoring. Hence, the integration of NER in the legal system not only broadens the scope of language models but also aids humans in extracting valuable insights and efficiently navigating through the vast amount of legal text.

In previous works (Li et al., 2022), researchers have primarily focused on developing supervised machine learning-based algorithms for Named Entity Recognition (NER) tasks. These algorithms typically require a significant amount of annotated data to train effectively. However, when it comes to NER tasks specific to legal documents, there are several shortcomings associated with available legal datasets. One of the main challenges is the limited availability of large-scale, high-quality annotated legal datasets. Creating these datasets involves manually annotating entities within legal documents, which is a time-consuming and expensive process.

Another challenge is the lack of diversity in legal datasets. Existing legal datasets tend to be skewed towards specific domains or jurisdictions, which may not adequately represent the diversity of legal documents encountered in real-world scenarios (Kalamkar et al., 2022). This lack of diversity can limit the effectiveness and robustness of NER models when applied to different legal contexts. Additionally, legal datasets often suffer from imbalanced entity distributions. Certain entity types, such as common names or generic terms, may occur frequently, while others, such as specific legal clauses or obscure legal terminology, may appear less frequently. To circumvent these shortcomings, we aim to explore Zero-shot and Few-shot learning-based approaches on pretrained language models.

2 Related work

Named Entity Recognition (NER) has been studied extensively, with studies spanning from traditional statistical models, as presented by (Borthwick et al., 1998) to more contemporary approaches involving deep neural networks, as demonstrated by (Li et al., 2022). Traditional machine learning techniques focused on using decision trees, conditional random fields, hidden Markov models, and support vector machines for NER. However, these became less prevalent with the rise of deep learning models like LSTM and transformers as they can learn better representations and capture complex patterns in text more effectively. (Yadav and Bethard, 2019).

One of the notable approaches in LSTM-based models for Named Entity Recognition is BiLSTM-CRF (Huang et al., 2015). This model combines bidirectional LSTMs to capture contextual information from both left and right contexts and a CRF layer for sequence labeling. It excels at long-range dependencies and nested entities, en-

suring coherent predictions. However, its computational complexity, training data requirements, and dependence on large annotated datasets pose challenges.

LegalBERT (Chalkidis et al., 2020) is another seminal work in NER that uses a transformer-based model specifically designed for legal text understanding. It utilizes a multi-layer transformer architecture, pre-trained on a large corpus of legal texts, followed by fine-tuning on specific NER tasks. LegalBERT benefits from contextualized representations and self-attention mechanisms, enabling it to capture complex patterns and dependencies in legal text.

Previous work (Ziletti et al., 2022), has also employed Zero-shot and Few-shot learning paradigms using Large Language Models to address the challenges of lack of annotated datasets in the biomedical domain. In this work, researchers posed the task of NER as a binary token classification in which the token is labeled as being present in the searched entity or not. Inspired by its significant improvement over previous works in bio-medical domain, we plan to extend this idea to the legal domain and experiment with different models as proposed in the approach.

3 What you proposed vs. what you accomplished

In our project proposal, we outlined several tasks that we intended to accomplish. Here is a summary of what we proposed versus what we actually achieved:

- Collect and preprocess the dataset: DONE - this phase involved the acquisition and subsequent preprocessing of the dataset in accordance with three distinct tasks: Spacy, fine-tuning, and one-shot learning.

- Test on Baseline: DONE - Using the CPU configuration and prioritizing accuracy as the optimization objective, we trained the complete dataset on the Spacy NER component pipeline.

- BiLSTM + CRF Model: NOT DONE. We encountered resource limitations that prevented us from training this model. Instead, we decided to allocate our compute resources to training transformers, which offered the potential for better results.

- Transformers: DONE - We experimented with two latest model architectures, namely DeBERTa and ELECTRA to investigate their performance on our downstream task of token classification. We explored various versions of these models and performed fine-tuning on the entire model, rather than just the model's head.

- Zero-shot and One-shot learning using Transformers: DONE - We experimented with Zero- and Few-shot learning by transforming the task of multi-class token classification into a binary token classification

- Prompt Engineering: DONE- In our proposal, we aimed to apply prompt engineering to an Indian legal NER dataset and experiment with different learning scenarios. We successfully applied prompt engineering and conducted experiments with zero-shot, one-shot, and few-shot learning, effectively achieving our objectives.

4 Dataset

We used the Indian Legal NER Dataset (Prathamesh Kalamkar, 2022), which is a corpus of 46,545 annotated legal named entities mapped to 14 legal entity types. The dataset is extracted from Preamble and Judgement documents of the Indian judicial system. Table 1 and 2 present a concise summary of the basic statistics of the data.

Table 1: Data distribution

Type	Number of Records
Train data: Judgement	9435
Train data: Preamble	1560
Test data: Judgement	949
Test data: Preamble	125

Table 2: Input Text Analysis

Type	Min	Max	Average
Number of Characters	14	23960	316
Number of Words	3	11571	84
Number of Tokens	7	14191	120

Table 3 provides a comprehensive description of various legal entity types. Our objective is to accurately identify these entities contained within the

input text. These entities can range from individual characters to multiple words. As we can see, this is a non trivial task as legal entities differ from standard named entities and are region-specific, requiring a tailored approach for each country's legal data.

In Figure 1, we present an illustrative example of an input judgement text. The identified entities are highlighted in distinct colors, visually depicting the task we aim to achieve.

Table 3: Indian Legal Named Entities Definitions

Named Entity	**Description**
COURT	Name of court mentioned
PETITIONER	Name of petitioners/appellants/revisionist from current case
RESPONDENT	Name of respondents/defendants/opposition from current case
JUDGE	Name of judges from current/previous cases
LAWYER	Name of lawyers from both parties
DATE	Any date mentioned in the judgment
ORG	Name of organizations apart from court
GPE	Geopolitical locations like names of states, cities, villages
STATUTE	Name of law/act mentioned in judgement
PROVISION	Sections, sub-sections, articles, orders, rules under a statute
PRECEDENT	All past court cases referred to in the judgement as precedent. It consists of party names + citation or case number
CASE_NUMBER	All other case numbers mentioned in the judgment
WITNESS	Name of witnesses in current judgment
OTHER_PERSON	Name of all other persons not included in petitioner, respondent, judge & witness

4.1 Data preprocessing

An example of the JSON format present in our dataset is depicted in Figure 2. This format illustrates the structure and organization of the data we are working with.

4.1.1 Preprocessing for Spacy Pipeline

To preprocess the data for the Spacy pipeline, the following steps were performed:

- Data Loading: The data, initially in JSON format (as shown in Figure 2), was loaded for processing.
- Formatting: Preprocessing was conducted to transform the data into the required format. For example, the text and entity labels were extracted from each annotation in the training data.
- Spacy Language Model: The English language model provided by Spacy was loaded to facilitate further processing.
- Creating a DocBin Object: A DocBin object was created to store the processed training examples.
- Iterating over Annotations: The code iterated over each annotation in the training data. For each annotation, a Spacy document was generated from the text.
- Processing Entity Labels: The entity labels were processed to create entity spans within the document. If it was not possible to create a span for an entity, it was skipped. Otherwise, the span was added to a list of entities.
- Filtering Entities: The list of entities was filtered to eliminate overlapping or conflicting spans.
- Assigning Entities to the Document: The filtered entities were assigned to the document, and the document was added to the DocBin object.
- Saving Processed Examples: Once all the training examples were processed, the DocBin object, containing the preprocessed data, was saved to disk.

The same preprocessing process was repeated for the development set.

```
{'id': 'd28bf7a50caf436db52174a77603fb77',
 'annotations': [{'result': [{'value': {'start': 196,
      'end': 209,
      'text': 'Supreme Court',
      'labels': ['COURT']},
     'id': 'EJFNTHLN',
     'from_name': 'label',
     'to_name': 'text',
     'type': 'labels'},
    {'value': {'start': 235,
      'end': 276,
      'text': 'Surajit Sarkar Vs. State of West Bengal36',
      'labels': ['PRECEDENT']},
     'id': 'CTU8Y7DS',
     'from_name': 'label',
     'to_name': 'text',
     'type': 'labels'}]}],
 'data': {'text': '".........Deficiencies in investigation by way of omissions and lapses on the part of investigati
ng agency cannot in themselves justify a total rejection of the prosecution case." \n\n Hon\'ble the Supreme Court ha
s again in the case of Surajit Sarkar Vs. State of West Bengal36, has addressed the issue and held that:-\n49.'},
 'meta': {'source': 'criminal_allahabad_high_court judgement https://indiankanoon.org/doc/69718660'}}
```


Figure 1: Raw JSON Data Record

4.1.2 Preprocessing for Pytorch Transformers

The following steps were performed for data preprocessing:

- Extracted the input text and associated annotations
- Tokenized the input text using a model-specific FAST tokenizer. A maximum token length of 120 was set, which corresponds to the average number of tokens observed in Table 2. To ensure faster model training, any excess text beyond this length was truncated.
- For each annotated token, identified their token positions using offsets_mapping and word_ids.
- Assigned label IDs (number mapping to each named entity) to corresponding tokens.
- Prepare data with input IDs, attention masks, and labels for every record.
- Combine processed train, test and validation datasets into a tokenized dataset object.

Through these steps, the data was appropriately formatted for training the token-based classification model using the transformers Trainer module.

4.1.3 Preprocessing for Zero-Shot & One-Shot Approach

This approach aims to transform a multi-class classification problem into a binary classification task. Consequently, it requires the data to be structured in a dataframe format with columns representing the entity-class, word-list, and labels. In this context, the labels are represented as a vector of binary values.

To get this desired format, the following steps are executed:

- Tokenization: The input text is split into individual words manually. The start and end indices of each word are stored for later use.
- Assigning Labels: For each annotated token, the corresponding token positions are identified using the start and end positions obtained in the previous step. The specific label ID is then assigned to the corresponding word token(s) using the annotations from raw data.
- For every record, a set of unique labels, list of words, label IDs is stored.
- Individual rows are created corresponding to unique labels present in every record. The final labels for each row consists of binary values (0 or 1) that indicate whether a word in the input text matches the corresponding unique class.

5 Baselines

During the training of the Spacy NER component pipeline, we employed a CPU configuration while prioritizing accuracy as the optimization objective. To optimize the training process and prevent memory issues, we made specific adjustments to the config.cfg file.

One modification we implemented was fixing the max_length of the input sequence to 120. This decision was based on our analysis, which revealed that over 90% of our dataset contained input sequences with less than 120 words. By setting the max_length to this value, we ensured that the model's training process focused on the majority of the dataset, rather than fixating on the largest input sequence. This adjustment improved the training process's efficiency and reduced the likelihood of encountering memory limitations.

Additionally, we set the max_steps parameter to 5500. This selection was informed by our experimentation, where we observed that the F1 score

Table 4: Model Performance Summary on Test Set

Approach	Model	P	R	F1
Spacy-Transformers + Transition-Based Parser	baseline	81.24	77.87	79.52
	roberta-base	**89.05**	88.90	**88.98**
	deberta-v3	88.03	**88.97**	88.50
	electra-base	86.86	86.30	86.58
End-to-End Transformers	deberta-base	84.53	88.12	86.29
	deberta-v3	83.73	87.79	85.71
	deberta-kfold	**85.87**	**89.87**	**87.82**
	electra-base	80.01	86.79	83.31
	electra-conll03	81.68	87.65	84.56
Prompt Engineering	GPT 3.5 - Zero Shot	84.60	85.70	85.15
	GPT 3.5 - One Shot	**90.10**	**89.50**	**89.80**
	GPT 3.5 - Few Shot	87.50	87.95	87.72

of the model began to plateau around this step count. By fixing the max_steps, we aimed to reduce the overall runtime of the training process without compromising the model's performance.

To monitor the model's progress during training, we set the evaluation_frequency to 200. This configuration ensured that we received updates on the F1 score after every 200 steps. Regular evaluations allowed us to track the model's performance and make any necessary adjustments or optimizations throughout the training process.

By making these specific modifications to the *config.cfg* file, we fine-tuned the training process of the Spacy NER component pipeline, optimizing it for accuracy and ensuring efficient memory usage. These adjustments were solely made to significantly improve the overall training experience. We did not perform any hyperparameter tuning on the dev or test set, to result in better performance.

6 Approach

6.1 Spacy-transformers with Transition-Based Parser

Our conceptual approach involved setting up a Spacy pipeline for Spacy's NER (Named Entity Recognition) component. This approach included training a total of four models: one baseline model using a transition-based parser, and three transformer models that also utilized a transition-based parser. The pipeline configuration was established to enable training and evaluation of each model separately, allowing us to compare their performance.

Regarding the working implementation, we successfully set up the Spacy pipeline. To train each model, we made necessary modifications to the *config.cfg* file by specifying the name of the transformer model to be used. This allowed us to customize and train the models based on our requirements.

We primarily relied on the official Spacy documentation for guidance and assistance during the implementation. The documentation provided valuable insights into the usage and customization of the Spacy pipeline, enabling us to build our models effectively.

In terms of the models we implemented ourselves, we incorporated three transformer models: *roberta-base*, *microsoft/deberta-v3-base*, and *google/electra-base-discriminator*. The associated configuration files for all four models can be found in the repository's *config* folder. Additionally, we provided a *spacy_pipeline.ipynb* notebook that encompasses the complete working pipeline setup, along with four other notebooks, named *spacy_(name of model).ipynb*, corresponding to each individual model.

For our experiments, we employed different computing resources. The baseline model was trained on CPU hardware using Jupyter Notebook. As for the three transformer models, we utilized Google Colab Pro with an A100 GPU and the high RAM option to avoid any out-of-memory errors.

The training duration for each model varied. The baseline model took approximately four hours to train, while each transformer model required approximately two hours to complete the training process.

In terms of results, we achieved notable performance improvements with the transformer mod-

Figure 2: Input Text with Identified Named Entities

els compared to the baseline. The baseline model achieved an F1 score of 0.78. On the other hand, the roberta-base model achieved an F1 score of 0.89, followed closely by the microsoft/deberta-v3-base model with a score of 0.88, and the google/electra-base-discriminator model with a score of 0.87. These results indicate the effectiveness of the transformer models in enhancing the NER task's performance.

6.2 End-to-End Transformers

Our second approach involved leveraging two state-of-the-art transformer models: DeBERTa and ELECTRA. These models have demonstrated superior performance compared to the popular RoBERTa-base model in various natural language processing (NLP) tasks. We fine-tuned different versions of these models, namely DeBERTa-base, DeBERTa-base-v3, ELECTRA-base-discriminator, and ELECTRA-base-fine-tuned on the CoNLL 2003 dataset. (Tjong Kim Sang and De Meulder, 2003)

To adapt the models to our domain-specific dataset, we performed fine-tuning on all layers of the transformers, rather than just the final layer. This allowed the models to capture the intricacies of our data more effectively. The training process consisted of 15 epochs with a batch size of 16. We utilized the Adam optimizer with specific hyperparameters, including a learning rate of 0.00005, beta1 of 0.9, beta2 of 0.999, and weight decay of 0.01.

The implementation of these models was inspired by this work (Moonat, 2021). The corresponding Jupyter notebooks for each model, namely *deberta-base.ipynb*, *deberta-v3-base.ipynb*, *electra-base.ipynb*, and *electra-conll03.ipynb*, provides comprehensive results for training, validation, and testing across all legal named entities. These notebooks also included loss curves for visual analysis.

For computational resources, we conducted our experiments using Google Colab GPUs and UCSD's Datahub GPUs. Initially, we encountered Out of Memory Issues due to our preprocessing approach, which did not truncate long sentences. As a result, the models defaulted to considering the largest number of input tokens (14191 as per Table 2) as the input size for the transformer. By specifying the maximum length and enabling truncation, we successfully resolved this issue.

The runtime for training each model was approximately 30 minutes following the aforementioned modification.

Among the four models, the best result was achieved using the *deberta-base* model, with an F1 score of 86.3. To further enhance our results and address model limitations, we performed K-Fold validation by splitting the dataset into 10 chunks and training 10 new models, where each model used a different chunk as the validation set. Finally, we performed model ensembling by averaging the softmax probabilities across all models and selecting the maximum value. This ensemble approach, implemented in *deberta_kfold.ipynb*, yielded a slightly improved F1 score of 87.8, making it our best model.

A summary of the results for all the models is provided in Table 4.

6.3 Zero-Shot and One-Shot Learning

For our third approach, we explored zero- and few-shot NER, drawing inspiration from (Ziletti et al., 2022). Zero and few-shot machine learning algorithms can effectively address the challenges associated with a scarcity of annotated data. Zero-

shot learning enables the identification of named entities in text without any prior examples during the training phase. On the other hand, few-shot learning facilitates the training of a more efficient model for named entity recognition, using only a handful of annotated example sentences.

In this approach, we convert the task of multi-class token classification into binary token classification. The core intuition behind this approach is that the model will be capable of learning the semantic relations between the provided and potential classes. Figure[3]

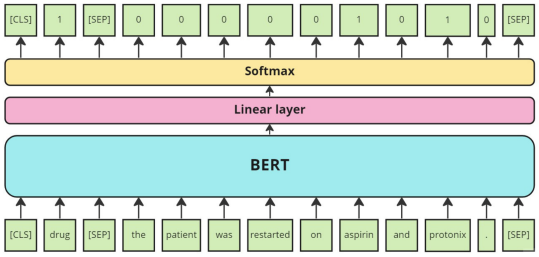

Figure 3: Model architecture for Zero- and Few-shot learning

Model	Precision	Recall	F1
Zero-Shot	82.38	83.56	83.72
One-shot	84.45	84.87	84.66
10-shot	84.89	85.13	85.01
100-shot	85.13	85.54	85.34

Table 5: Results of Zero- and Few-shot learning on the test set with 'PETITIONER' as the hidden class

We conducted our experiments using the bert-base-cased model. For Zero-shot learning, we designated PETITIONER as the hidden class, meaning that the inputs of this class were excluded from the training dataset. Following the training, the model was evaluated solely on the hidden class in the test set. The results on a sub-sampled dataset are presented in Table 1. For the few-shot approach, we provided 1, 10, and 100 examples of the hidden class during the training phase.

6.4 Prompt Engineering

Prompt engineering plays a pivotal role in shaping the learning process of language models. It serves as a form of guide, helping models to better understand the task at hand and navigate through the specificities of the given data. It's like providing a roadmap that helps the model to process data in a more focused and effective way. In essence, prompt engineering aids models in generating more accurate and contextually appropriate outputs.

In our recent project, we explored the power of prompt engineering with an Indian legal dataset meant for Named Entity Recognition (NER). Due to computational constraints and the desire to capture dataset diversity, we decided to conduct our experiment using a random sample of 100 data points.

We created prompts using examples from our dataset, aiming to drive the model towards generating more precise outputs. This technique is akin to one-shot learning, where the model is provided with a single example to learn from. We provided one example for each entity present in our dataset, aiming to enhance the model's contextual comprehension and improve its ability to accurately identify and classify entities.

For instance, in one of the data points, Baliram Sharma (P.W.5) states in his evidence that 16 years ago at the time of occurrence, there was hulla in the village that MCC Party members had arrived. This prompt helps the model understand the context and entities involved, with Baliram Sharma being the witness and MCC Party being the organization mentioned.

Beyond this, we experimented with zero-shot, one-shot, and few-shot learning scenarios. In the zero-shot learning scenario, the model was expected to rely solely on its pre-training to generalize, with no specific examples provided. On the other hand, few-shot learning gave the model a small set of examples to learn from. Our goal was to compare the performance of the model under these different learning scenarios, to figure out which approach yielded the best results for our task of Named Entity Recognition in an Indian legal context.

The results of these iterative experiments, detailed in Table 4, showed that one-shot learning gave us the highest accuracy, followed by few-shot learning, and finally zero-shot learning. This experimental process was instrumental in understanding the strengths and weaknesses of each approach, allowing us to effectively achieve our objectives.

By incorporating the prompt you provided, we can further enrich the discussion on the specific entities and their context within the prompt engineering framework.

7 Error analysis

For error analysis, first we consider the best performing transformer model: deberta-kfold. Figure 3 summarizes the performance of the model at entity level.

	Label	Precision	Recall	F1-score	Number
0	CASE_NUMBER	0.731707	0.857143	0.789474	70
1	DATE	0.923729	0.947826	0.935622	115
2	RESPONDENT	0.571429	0.571429	0.571429	7
3	PETITIONER	0.600000	0.750000	0.666667	16
4	WITNESS	0.866667	0.951220	0.906977	41
5	COURT	0.906736	0.930851	0.918635	188
6	NO_TAG	0.869817	0.893954	0.881720	2018
7	GPE	0.632653	0.688889	0.659574	45
8	PRECEDENT	0.652632	0.849315	0.738095	73
9	PROVISION	0.919708	0.906475	0.913043	139
10	STATUTE	0.894444	0.993827	0.941520	162
11	JUDGE	0.607143	0.894737	0.723404	19

Figure 4: Model Performance at Entity Level

Analyzing the results above, we observe that the entities 'Respondent' and 'Petitioner' exhibit lower accuracy compared to others. To gain insights into the model's mispredictions, we examined cases where the model confidently predicts an incorrect answer. Consider the following example: *The respondent No. 2/DDA shall also take appropriate steps in accordance with law in case of violation of such stipulation in the letter of allotment by the unaided schools.*

In this case, the model correctly identifies 'DDA' as the respondent since the text contains the word 'respondent'. The model also performs well when the respondent is a person's name. However, it struggles to predict the entity correctly in cases where the aforementioned patterns are not present. For instance: *Writ Tax Petition No.524/2021), dated 30.09.2021, reported in 2021(10) TMI 517, the learned Single Judge had quashed the notices issued under Section 148 of the Act.* Here, the respondent is '2021(10) TMI 517', which is identified by the date and special number instead of a name, without the word 'respondent' preceding it. As a result, the model encounters difficulty in recognizing it as the respondent. This suggests that the model would benefit from additional contextual information or more examples to improve its classification of such entities.

During our prompt engineering exploration, we discovered that the model occasionally recognized more entities than required, leading to incorrect identifications and false positive predictions. For example, For this example *"The land-owning agency later submitted a revised proposal to the Authority in respect of plot No. 1 (out of the eight plots), vide communication dated 31.1.2020."*, the model incorrectly identified 'plot No. 1' as a PROVISION entity, despite it not being mentioned in the ground truth. However, it correctly recognized '31.1.2020' as a DATE entity. This emphasizes the need for careful fine-tuning of prompt engineering to avoid such erroneous identifications.

By conducting further in-depth analysis of the results, various other patterns and insights can be uncovered.

8 Contributions of group members

All members of the group contributed equally to these areas: Data pre-processing (as every approach required the data in a different format), error analysis and report writing.

The individual tasks were divided and accomplished as given below:

- Test on Baseline: Samit
- Spacy-transformers+Transition-Based Parser: Samit
- End-to-End Transformers: Rupashi
- Zero-shot and One-shot learning using Transformers: Pradeep
- Prompt Engineering: Harshad

9 Conclusion

In this research project, we focused on advancing Named Entity Recognition (NER) in the legal domain. We experimented with various models and techniques, including spacy NER, spacy-transformers, fine-tuning using transformers, and zero and one-shot learning for NER. Prompting strategies were also employed to align with the trend of Language Model Models (LLMs). Our results were comparable to state-of-the-art models, demonstrating the effectiveness of our approaches.

Throughout the project, we gained insights into NLP tools and techniques. Model selection and configuration played a significant role in NER performance. Fine-tuning and k-fold validation improved model accuracy. Transformer-based models showed promise, but training and

fine-tuning required substantial computational resources. Prompting strategies and determining optimal prompts proved challenging but valuable for accurate legal entity extraction.

Despite challenges, our project delivered positive outcomes. Computational resources and time were needed for transformer model training. Ensuring component compatibility, such as spacy-transformers integration, required attention. Determining effective prompts for accurate legal entity extraction was non-trivial. However, our perseverance and methodological rigor allowed us to overcome obstacles and achieve noteworthy results.

Future directions for this project include incorporating additional legal datasets, exploring domain adaptation techniques, investigating more ensembling methods, leveraging active learning to optimize annotation, and delving into model interpretability for legal NER tasks. These avenues hold promise for broader understanding, improved performance, and insight into the decision-making process of NER models in the legal domain.

References

Borthwick, A., Sterling, J., Agichtein, E., and Grishman, R. (1998). NYU: Description of the MENE named entity system as used in MUC-7. In *Seventh Message Understanding Conference (MUC-7): Proceedings of a Conference Held in Fairfax, Virginia, April 29 - May 1, 1998*.

Cardellino, C., Teruel, M., Alemany, L. A., and Villata, S. (2017). A low-cost, high-coverage legal named entity recognizer, classifier and linker. In *Proceedings of the 16th Edition of the International Conference on Articial Intelligence and Law*, ICAIL '17, page 9–18, New York, NY, USA. Association for Computing Machinery.

Chalkidis, I., Fergadiotis, M., Malakasiotis, P., Aletras, N., and Androutsopoulos, I. (2020). Legal-bert: The muppets straight out of law school.

Huang, Z., Xu, W., and Yu, K. (2015). Bidirectional lstm-crf models for sequence tagging.

Kalamkar, P., Tiwari, A., Agarwal, A., Karn, S., Gupta, S., Raghavan, V., and Modi, A. (2022). Corpus for automatic structuring of legal documents. In *Proceedings of the Thirteenth Language Resources and Evaluation Conference*, pages 4420–4429, Marseille, France. European Language Resources Association.

Li, J., Sun, A., Han, J., and Li, C. (2022). A survey on deep learning for named entity recognition. *IEEE Transactions on Knowledge and Data Engineering*, 34(1):50–70.

Moonat, D. (2021). Fine-tune ner. https://github.com/dmoonat/Named-Entity-Recognition/blob/main/Fine_tune_NER.ipynb.

Prathamesh Kalamkar, Astha Agarwal, A. T. S. G. S. K. V. R. (2022). Named Entity Recognition in Indian Court Judgments. *arXiv preprint arXiv:2211.03442v1*.

Tjong Kim Sang, E. F. and De Meulder, F. (2003). Introduction to the conll-2003 shared task: Language-independent named entity recognition. *arXiv preprint cs/0306050*.

Yadav, V. and Bethard, S. (2019). A survey on recent advances in named entity recognition from deep learning models.

Ziletti, A., Akbik, A., Berns, C., Herold, T., Legler, M., and Viell, M. (2022). Medical coding with biomedical transformer ensembles and zero/few-shot learning. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Industry Track*. Association for Computational Linguistics.