AI-Powered Legal Decision Support System

A Project Report

Presented to

Dr. Chris Pollett

Department of Computer Science

San José State University

In Partial Fulfillment

Of the Requirements for the Class

CS 297

By

Alisha Rath

December 2024

I. ABSTRACT

This paper explores the development of an AI-powered legal decision support system to predict case outcomes from court documents, initial filings, and transcripts. Traditional legal analysis faces challenges such as inconsistencies and subjective interpretations. Leveraging machine learning (ML) and large language models (LLMs) like DistilBERT, this project aims to create a robust tool for predicting outcomes such as "guilty" or "not guilty." It will compare different fine-tuning techniques to optimize accuracy. Branting et al. (2021) emphasize the necessity for scalable and explainable legal prediction systems to ensure transparency and reliability in AI-driven legal decisions [1]. Ejjami (2024) discusses how AI can streamline judicial workflows and democratize access to legal insights, particularly benefiting smaller firms and individuals with limited resources [2]. Tsihrintzis et al. (2024) provide a comprehensive overview of advanced AI models that support complex decision-making processes, which forms the foundation for this project's methodology [3]. The system aims to improve legal decision-making, streamline workflows, and enhance access to legal insights.

TABLE OF CONTENTS

I. INTRODUCTION

The legal field is undergoing a significant transformation with the advent of artificial intelligence (AI) technologies. Traditional legal case analysis often grapples with challenges such as subjective interpretation, time-intensive processes, and inconsistencies in decision-making. In response to these challenges, this project aims to develop an AI-powered legal decision support system designed to predict case outcomes based on initial filings, court documents, and transcripts.

Leveraging machine learning (ML) and large language models (LLMs) like DistilBERT, the proposed system seeks to provide a reliable, data-driven approach to legal decision-making. The integration of AI in legal systems is becoming increasingly important, as highlighted by Branting et al. (2021), who emphasize the need for scalable and explainable legal prediction systems to ensure transparency and trust in AI-driven decisions [1]. By incorporating advanced AI techniques, this project aims to enhance the accuracy and efficiency of legal predictions, addressing both structured and unstructured data from diverse legal documents.

Ejjami (2024) underscores the transformative potential of AI in streamlining judicial workflows and democratizing access to legal insights, particularly benefiting smaller firms and individuals with limited resources [2]. This project aligns with these insights, aiming to create a system that not only predicts outcomes such as "guilty" or "not guilty" but also provides detailed explanations to support legal professionals in their decision-making processes.

Building on the methodologies discussed by Tsihrintzis et al. (2024), this project's design incorporates a comprehensive overview of advanced AI models that support complex decision-making processes [3]. The development of this AI-powered legal decision support system involves iterative model development, fine-tuning, and rigorous evaluation to

ensure reliability in real-world applications.

The report is organized into several key sections, each addressing specific deliverables and aspects of the project. It begins with Deliverable I, focusing on the development of a classification model using supervised learning to predict case outcomes based on legal documents. Following this, Deliverable II explores the integration of large language models (LLMs) like DistilBERT, detailing how these models enhance the analysis of complex legal language and improve predictive accuracy. Deliverable III addresses the cleaning and preprocessing of the legal case dataset, which is essential for model training. The report proceeds to Deliverable IV, where experiments evaluating different fine-tuning methods are discussed, highlighting the effectiveness of various approaches in enhancing model performance. Finally, the findings and insights from the project are summarized in the Conclusion, followed by a list of References that provide a foundation for further research and development in the field of AI in legal decision-making.

II. DELIVERABLE 1: FINE-TUNING INSIGHT

This deliverable centers on building a foundational understanding of supervised fine-tuning using large language models (LLMs) with a focus on predicting legal case outcomes. Fine-tuning involves adapting a pre-trained model to perform a specific task by adjusting it with new data, and is crucial for making AI models more relevant to specialized domains. Here, a particular parameter-efficient method called Low-Rank Adaptation (LoRA) is used, allowing for adjustments to be made with minimal computation while retaining the essential learned parameters of the model. In this deliverable, I followed a structured tutorial on fine-tuning with LoRA, progressing through several steps designed to understand and practically apply fine-tuning in a way that will later support my legal case prediction model.

1. **Understanding Supervised Learning.**

To build a strong foundation for fine-tuning, I began with supervised learning, the most common approach in machine learning (ML) for mapping input data to labeled outputs. Here's an outline of my study:

1.1. Basics of Supervised Learning:

In supervised learning, models learn by associating input data with the desired output labels based on a provided dataset. For legal case outcome predictions, this means training a model to understand patterns in case documents or transcripts and to predict outcomes like "guilty" or "not guilty."

Supervised learning requires a substantial volume of labeled data, which presents challenges in certain domains like law, where data may be less accessible or inconsistently labeled.

1.2. Applications in Classification Tasks:

Supervised learning has widespread applications, including classification tasks like sentiment analysis, document categorization, and in our case, legal decision-making.

For legal predictions, this technique is critical, as it allows the model to leverage structured patterns in previous case data to make informed predictions about similar cases.

1.3. Challenges:

A primary challenge is acquiring high-quality, labeled datasets. The accuracy of a supervised model heavily depends on the data used for training, and in domains like law, data scarcity can limit model performance.

Another challenge lies in adapting generic models to domain-specific tasks, where language and data patterns can vary significantly from the training dataset originally used for the model.

2. **Exploring Fine-Tuning Concepts:**

After establishing a foundation in supervised learning, I explored fine-tuning, which leverages transfer learning to adapt a pre-trained model to specific tasks. Here, the concept of LoRA was especially relevant:

2.1. Transfer Learning and Model Refinement:

Fine-tuning uses a base model already pre-trained on extensive general-purpose datasets, such as large corpora of text data. By retraining it on a new, specialized dataset, the model can learn to perform a narrower, domain-specific task.

Transfer learning, when applied correctly, allows the model to retain broad language understanding while becoming more adept at identifying patterns specific to legal texts.

2.2. LoRA for Parameter-Efficient Fine-Tuning:

LoRA, or Low-Rank Adaptation, is an advanced fine-tuning approach that focuses on tuning a subset of model parameters while keeping the majority of layers frozen. This approach is both computationally efficient and effective for adapting models with limited resources.

In LoRA, a few lightweight layers are trained on the new data, which allows the model to focus on relevant patterns for the specialized task without the need for intensive retraining of all layers. This makes it particularly well-suited for tasks like legal prediction where domain-specific nuances are essential but computational power may be limited. Figure 1 shows LoRA architecture.

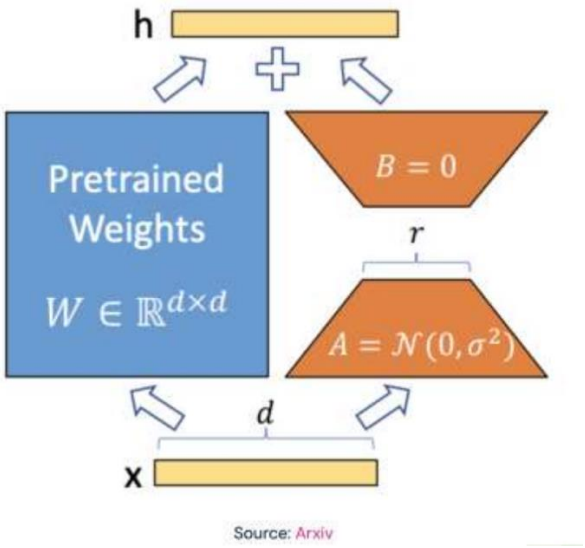


Figure 1: LoRA architecture

3. **Hands-On Tutorial – Fine-Tuning with LoRA:**

To put the theoretical concepts into practice, I followed a hands-on tutorial for fine-tuning DistilBERT using LoRA. This practical exercise covered each stage of the fine-tuning process:

3.1. Environment Setup:

Google Colab, I created a Python environment with the necessary libraries, including Hugging Face's Transformers library and LoRA-specific tools. Google Colab's GPU support was essential in speeding up the training process.

3.2. Model Loading:

The DistilBERT model, a compact version of BERT, was loaded from Hugging Face's model hub. DistilBERT's smaller size is advantageous for tasks requiring efficient computation without significant sacrifices in performance.

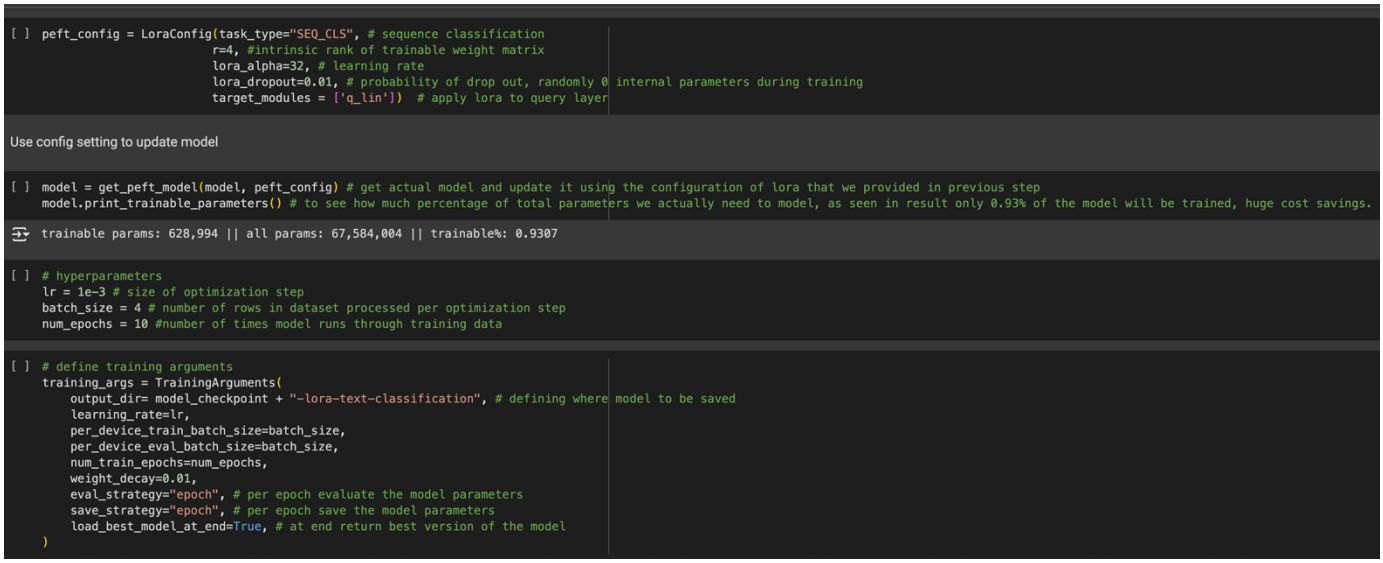
3.3. Data Preparation:

I selected a sentiment analysis dataset to simulate a classification task. The data was cleaned and tokenized, a process that involves converting text into numerical representations, making it compatible with the model's input format.

3.4. Fine-Tuning with LoRA:

The fine-tuning process involved configuring LoRA to adjust only a limited set of parameters while keeping most of the model's architecture intact. This selective tuning allows for rapid adaptation without high computational demands.

I ran the model over multiple epochs (iterations over the entire dataset) to ensure that the fine-tuned parameters were effectively learning patterns from the dataset. Figure 2 shows the code for fine-tuning with LoRA.



```
[ ] peft_config = LoraConfig(task_type="SEQ_CLS", # sequence classification
                             r=4, #intrinsic rank of trainable weight matrix
                             lora_alpha=32, # learning rate
                             lora_dropout=0.01, # probability of drop out, randomly 0 internal parameters during training
                             target_modules = ['q_lin'])  # apply lora to query layer

Use config setting to update model

[ ] model = get_peft_model(model, peft_config) # get actual model and update it using the configuration of lora that we provided in previous step
    model.print_trainable_parameters() # to see how much percentage of total parameters we actually need to model, as seen in result only 0.93% of the model will be trained, huge cost savings.

    trainable params: 628,994 || all params: 67,584,004 || trainable%: 0.9307

[ ] # hyperparameters
    lr = 1e-3 # size of optimization step
    batch_size = 4 # number of rows in dataset processed per optimization step
    num_epochs = 10 #number of times model runs through training data

[ ] # define training arguments
    training_args = TrainingArguments(
        output_dir= model_checkpoint + "-lora-text-classification", # defining where model to be saved
        learning_rate=lr,
        per_device_train_batch_size=batch_size,
        per_device_eval_batch_size=batch_size,
        num_train_epochs=num_epochs,
        weight_decay=0.01,
        eval_strategy="epoch", # per epoch evaluate the model parameters
        save_strategy="epoch", # per epoch save the model parameters
        load_best_model_at_end=True, # at end return best version of the model
    )
```

Figure 2. Fine-Tuning with LoRA

3.5. Evaluation:

I assessed the model's performance using a test dataset, analyzing accuracy to measure the impact of fine-tuning. The results showed improved classification accuracy, affirming that fine-tuning with LoRA enhanced the model's domain-specific performance. Figure 3

shows accuracy per epoch while training the model. Figures 4 and 5 show results before and after training.

```
# creater trainer object
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_dataset["train"],
    eval_dataset=tokenized_dataset["validation"],
    tokenizer=tokenizer,
    data_collator=data_collator, # this will dynamically pad examples in each batch to be equal length
    compute_metrics=compute_metrics,
)

# train model
trainer.train() # from top to bottom you can see training loss decreases which is good, but validation loss is increasing too which is based as overfitting happening
```

[2500/2500 07:41, Epoch 10/10]

Epoch	Training Loss	Validation Loss	Accuracy
1	No log	0.367422	{'accuracy': 0.864}
2	0.432500	0.372985	{'accuracy': 0.882}
3	0.432500	0.615503	{'accuracy': 0.868}
4	0.235200	0.722475	{'accuracy': 0.884}
5	0.235200	0.778032	{'accuracy': 0.889}
6	0.079700	0.870770	{'accuracy': 0.89}
7	0.079700	0.923493	{'accuracy': 0.887}
8	0.029500	0.938481	{'accuracy': 0.888}
9	0.029500	1.009659	{'accuracy': 0.892}
10	0.005700	1.013857	{'accuracy': 0.891}

Figure 3. Accuracy per epoch

Applying untrained model to text

```
# define list of examples
text_list = ["It was good.", "Not a fan, don't recommed.", "Better than the first one.", "This is not worth watching even once.", "This one is a pass."]

print("Untrained model predictions:")
print("----------------------------")
for text in text_list:
    # tokenize text
    inputs = tokenizer.encode(text, return_tensors="pt")
    # compute logits
    logits = model(inputs).logits
    # convert logits to label
    predictions = torch.argmax(logits)

    print(text + " - " + id2label[predictions.tolist()])
```

```
Untrained model predictions:
----------------------------
It was good. - NEGATIVE
Not a fan, don't recommed. - NEGATIVE
Better than the first one. - NEGATIVE
This is not worth watching even once. - NEGATIVE
This one is a pass. - NEGATIVE
```

Figure 4. Prediction before training

Generate prediction or test

```
# define list of examples
text_list = ["It was good.", "Not a fan, don't recommed.", "Better than the first one.", "This is not worth watching even once.", "This one is a pass."]

print("Trained model predictions:")
print("--------------------------")
for text in text_list:
    # tokenize text
    inputs = tokenizer.encode(text, return_tensors="pt").to('cuda') # Ensure inputs are on the same device as the model
    # compute logits
    logits = model(inputs).logits
    # convert logits to label
    predictions = torch.argmax(logits)

    print(text + " - " + id2label[predictions.tolist()])
```

```
Trained model predictions:
--------------------------
It was good. - POSITIVE
Not a fan, don't recommed. - NEGATIVE
Better than the first one. - POSITIVE
This is not worth watching even once. - NEGATIVE
This one is a pass. - NEGATIVE
```

Figure 5. Prediction after training

III. DELIVERABLE 2: CASE DAA PREPARATION

The objective of this deliverable was to identify, analyze, and prepare a suitable dataset for training machine learning models to predict legal case outcomes. Legal case prediction requires datasets that provide structured information about case facts, decisions, and potential metadata, such as identifiers or issue types. After extensive research, I selected the dataset that best met these requirements, considering factors such as case scope, format, linguistic diversity, and overall compatibility with machine learning workflows.

To streamline the data preparation process, I followed a structured approach divided into three primary stages:

1. **Researching Available Datasets:**

To find a dataset suitable for legal case prediction, I explored several data sources:

1.1. Kaggle - Supreme Court Judgment Prediction Dataset: This dataset contains 3,304 cases from 1955 to 2021, covering various Supreme Court cases in the U.S. It includes case identifiers, descriptions of case facts, and judgment outcomes, with labels to indicate case-winning parties. The dataset is designed specifically for natural language processing (NLP) tasks in legal predictions, offering a structured, straightforward format that simplifies preprocessing and integration with models.

1.2. GitHub - Oyez Decision Prediction Dataset: This dataset includes details on case facts and outcomes, with a primary focus on NLP-based predictions. While it provides rich textual data for case descriptions, it has a smaller scope (fewer cases) than the Kaggle dataset and offers fewer metadata elements, making it more limited in versatility.

1.3. Hugging Face - Swiss Judgment Prediction Dataset: This multilingual dataset contains approximately 85,000 cases from the Swiss Federal Supreme Court, offering a cross-linguistic look at judgment predictions. It includes case facts in multiple languages (German, French, and Italian), providing outcomes based on case approvals or dismissals. Although the dataset is substantial in size, it required substantial preprocessing to adapt to English-based models, given that the current project focuses

on U.S. legal case predictions.

2. **Dataset Comparison:**

To determine which dataset was best suited for this project, I compared the three options, considering factors such as the number of cases, linguistic features, and format compatibility. Table 1 shows a summary of the comparative analysis of the datasets:

Dataset Name	Source	Cases	Language(s)	Key Features	Advantages	Disadvantages
Supreme Court Judgment Prediction	Kaggle	3,304	English	Case facts, judgment outcomes	Well-structured, comprehensive, specific to U.S. cases	Limited to U.S. cases
Oyez Decision Prediction	GitHub	~1,000	English	Detailed case facts and outcomes	Rich textual data	Smaller scope, limited metadata
Swiss Judgment Prediction	Hugging Face	85,000	German, French, Italian	Multilingual facts, approval/dismissal outcomes	Large dataset, multilingual diversity	Requires extensive preprocessing

Table 1. Dataset comparison

The Supreme Court Judgment Prediction dataset from Kaggle emerged as the most suitable option for the following reasons:

Rich Metadata: It includes essential metadata such as unique case identifiers, case facts, and judgments, which are crucial for developing accurate predictive models.

Historical Coverage: The dataset spans cases from 1955 to 2021, providing comprehensive insight into historical precedents, which is beneficial for identifying patterns in case outcomes.

Ease of Access: This dataset's well-structured, labeled format facilitates seamless preprocessing and integration with machine learning models, reducing initial setup time.

3. **Dataset Selection and Rationale:**

Based on the comparative analysis, the Supreme Court Judgment Prediction dataset was chosen as the primary dataset for this project. Key factors influencing this decision included:

3.1. Compatibility with NLP Techniques: Designed for NLP tasks, the dataset aligns well with language-based models, ensuring that case facts and textual descriptions can be leveraged effectively.

3.2. Availability of Judgments: With clear verdict labels (1 for a win and 0 for a loss), this dataset facilitates binary classification tasks suitable for machine learning.

Dataset Analysis and Cleaning

The Supreme Court dataset included case identifiers, textual descriptions of case facts, and judgment labels, which needed preprocessing before model training. The following steps were carried out for data analysis and cleaning:

Data Loading: The dataset was loaded into a Pandas DataFrame, and I performed a preliminary scan to assess the format, column names, and data types. This step ensured that the data was accessible and ready for further processing. Figure 6 shows the code snippet for loading the dataset.

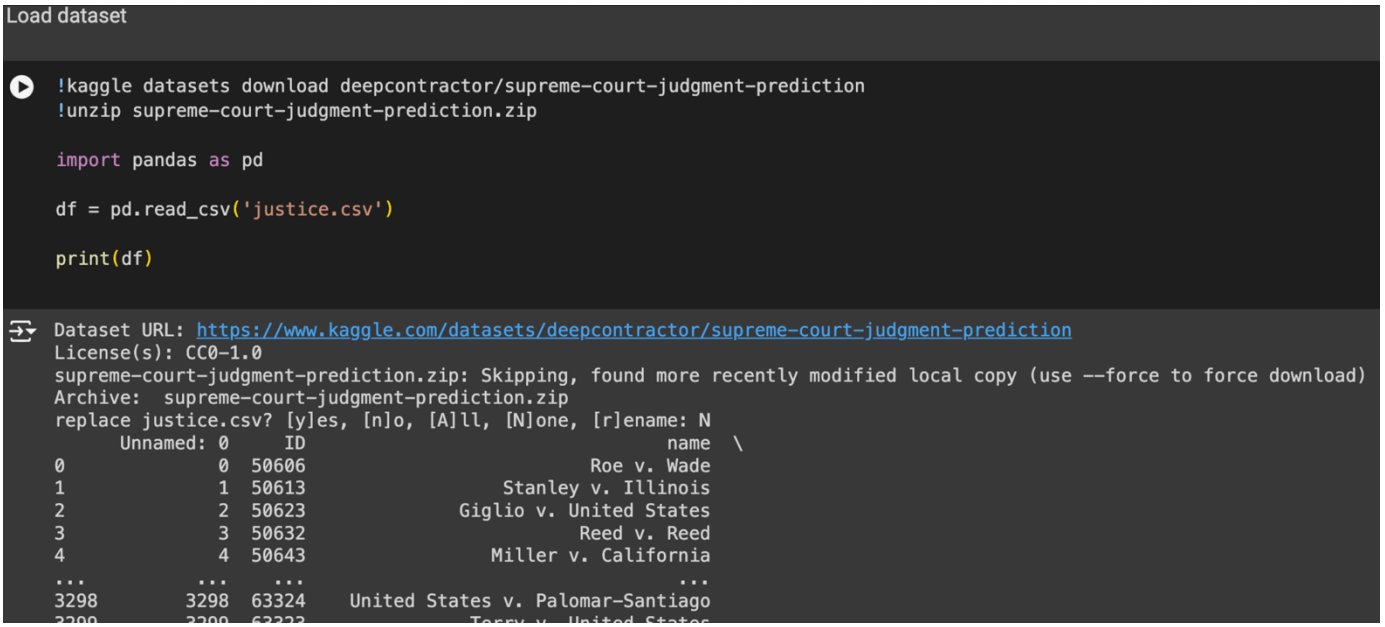


Figure 6. Loading of supreme court dataset

Handling Missing Values: While the dataset was largely complete, any missing values in non-essential fields were dropped to maintain data integrity.

Tokenization and Text Preprocessing: For machine learning compatibility, case facts were tokenized using DistilBERT's tokenizer, converting each text into numerical tokens. The tokenizer's pre-trained vocabulary and embeddings were essential in preparing the data for subsequent model training. Figure 7 shows the preprocessing process of the Supreme Court datatset.

Preprocess dataset

```
# Preprocess the data
# just keep facts and first_party_winner

#drop all rows with na
df = df.dropna()
df = df[['facts', 'first_party_winner']]
df['first_party_winner'] = df['first_party_winner'].astype(int)

#remname facts to text and first_party_winner to label
df = df.rename(columns={'first_party_winner': 'label', 'facts': 'text'})

# remove the p tag from the text
df['text'] = df['text'].str.replace('<p>', '')

print(df)
```

```
                                                   text  label
1     Joan Stanley had three children with Peter Sta...      1
2     John Giglio was convicted of passing forged mo...      1
3     The Idaho Probate Code specified that "males m...      1
4     Miller, after conducting a mass mailing campai...      1
5     Ernest E. Mandel was a Belgian professional jo...      1
...                                                 ...    ...
3297  For over a century after the Alaska Purchase i...      1
3298  Refugio Palomar-Santiago, a Mexican national, ...      1
3299  Tarahrick Terry pleaded guilty to one count of...      0
3300  Joshua James Cooley was parked in his pickup t...      1
3302  The Natural Gas Act (NGA), 15 U.S.C. §§ 717–71...      1

[3098 rows x 2 columns]
```

Figure 7. Preprocessing the Supreme Court dataset

Data Splitting: The dataset was split into training, validation, and test sets to facilitate model evaluation and avoid overfitting. This step allowed me to reserve part of the data solely for testing, ensuring reliable evaluation metrics. Figure 8 shows dataset is split into training and testing sets.

Dividing data to training and testing data from the given dataset

```
from sklearn.model_selection import train_test_split

# Split the dataset into training and testing sets
train_df, test_df = train_test_split(df, test_size=0.2, random_state=42)

# Convert the dataframes to Hugging Face Datasets
train_dataset = Dataset.from_pandas(train_df)
validation_dataset = Dataset.from_pandas(test_df)

train_dataset = train_dataset.select_columns(['label', 'text'])
validation_dataset = validation_dataset.select_columns(['label', 'text'])

# Remove the index column if it exists
if '__index_level_0__' in train_dataset.features:
    train_dataset = train_dataset.remove_columns(['__index_level_0__'])
if '__index_level_0__' in validation_dataset.features:
    validation_dataset = validation_dataset.remove_columns(['__index_level_0__'])


#print(train_dataset)
#print(validation_dataset)

dataset = DatasetDict({
    'train': train_dataset,
    'validation': validation_dataset
})
dataset
```

```
DatasetDict({
    train: Dataset({
        features: ['label', 'text'],
        num_rows: 2478
    })
    validation: Dataset({
        features: ['label', 'text'],
        num_rows: 620
    })
})
```

Figure 8. The code snippet for splitting the dataset into training and validation set

IV. DELIVERABLE 3: VERDICT TRAIN

The primary goal of this deliverable was to fine-tune the DistilBERT-uncased model using a dataset of legal cases to predict verdicts—specifically, whether the first party would win based on the case facts. DistilBERT, a distilled and efficient variant of BERT, is known for performing well on various natural language processing (NLP) tasks with fewer computational resources. With the help of LoRA (Low-Rank Adaptation), a parameter-efficient fine-tuning technique, this project aims to adapt the model to the specifics of legal decision-making by optimizing it for accurate verdict prediction.

Here is a breakdown of each step in this deliverable, from model selection to final evaluation.

1. Selecting DistilBERT for Fine-Tuning

Why DistilBERT? DistilBERT was chosen as the base model for several reasons:

1.1. Lightweight and Efficient: As a streamlined version of BERT, DistilBERT uses fewer parameters and achieves approximately 97% of BERT's performance, making it a practical choice for fast experimentation without requiring extensive computational resources.

1.2. Performance on NLP Tasks: DistilBERT is well-suited for text classification and other NLP tasks, which is essential for the legal verdict classification based on case details.

1.3. Transfer Learning Capabilities: Pre-trained on large language corpora, DistilBERT can be fine-tuned on the specific legal dataset, allowing for quick adaptation to the domain-specific task of verdict prediction.

2. Using LoRA for Efficient Fine-Tuning

Why LoRA (Low-Rank Adaptation)? LoRA was employed to achieve parameter-efficient fine-tuning for DistilBERT. This method offers several advantages:

2.1. Reduced Computational Costs: By freezing most of the model's original parameters and only training small additional matrices, LoRA minimizes the computational load.

2.2. Enhanced Generalization: Freezing the majority of parameters prevents the model from overfitting, especially beneficial when working with a small dataset like the Supreme Court data.

2.3. Flexible Adaptation: LoRA enables model adaptation without fully altering the original model, making it easier to apply updates and optimizations specific to the legal case dataset.

3. Data Preparation and Tokenization

The dataset, containing case facts and corresponding verdict labels (1 for a first-party win and 0 for a loss), was prepared for training by performing the following steps:

Data Splitting: The dataset was divided into training, validation, and test sets to evaluate the model effectively.

Tokenization: Each case's facts were tokenized using DistilBERT's tokenizer, converting the text into numerical tokens that the model could interpret. This step was essential in making the data compatible with the model's input requirements. Figure 9 shows the DistilBERT pre-trained model was loaded and prepared for fine-tuning.

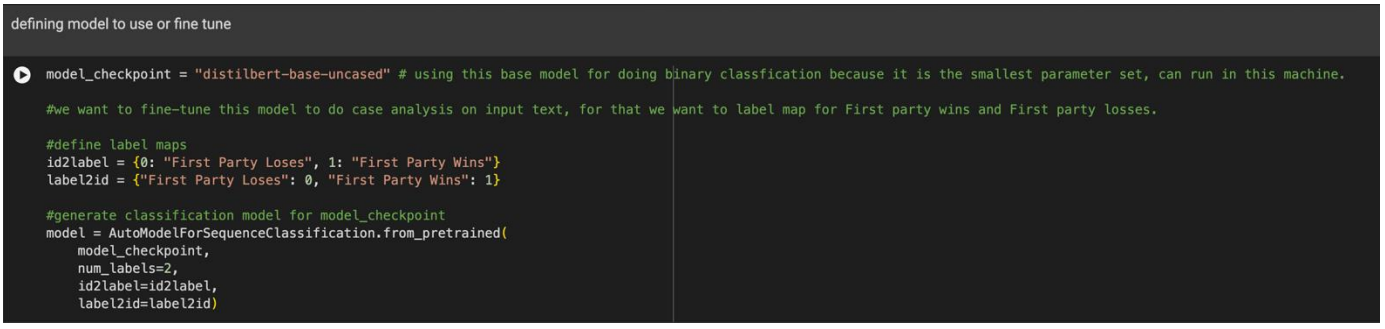


Figure 9. Loading of the base model

4. Establishing a Baseline with an Untrained Model

Before fine-tuning, I conducted an evaluation of the pre-trained, unmodified DistilBERT model on the dataset to establish a baseline performance measure. This initial test provided insight into the model's limitations and areas for improvement, highlighting the need for specialized fine-tuning on legal text.

5. Training the Model with LoRA Configurations

During training, LoRA configurations were employed to adapt the model to the legal case dataset without the risk of extensive overfitting. Figure 10 shows fine-tuning the base model using LoRA. Key aspects of the training process included:

Hyperparameter Tuning: Parameters such as learning rate and batch size were optimized to improve performance.

Monitoring: Training metrics were observed at each epoch to adjust settings dynamically and prevent overfitting.

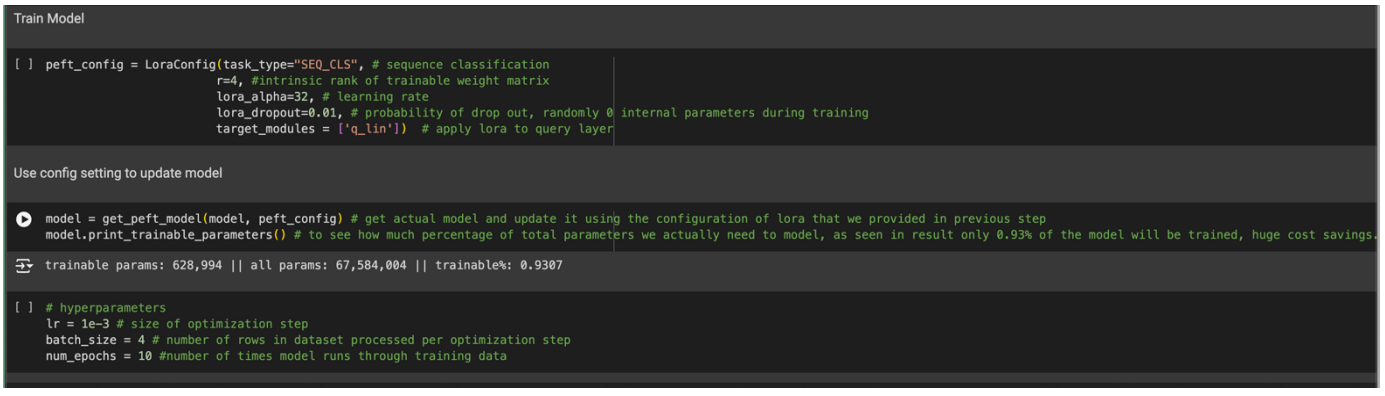


Figure 10. LoRA configs for fine-tuning base model

6. Monitoring Per-Epoch Accuracy and Validation

Throughout training, I recorded the model's accuracy at the end of each epoch and calculated validation metrics to ensure the model's steady improvement. This phase included:

Evaluating Model Performance: Accuracy and validation loss were tracked to identify any

overfitting trends.

Selecting the Best Model: The code was implemented to return the best-performing model based on validation performance, ensuring an optimal version for prediction tasks.

7. Predicting Results with the Trained Model

After the fine-tuning process, the model was evaluated on the test set to assess its ability to classify case outcomes based on facts accurately. This final step was critical in validating the training process and measuring the model's effectiveness for real-world applications in legal decision prediction.

Model Evaluation

To monitor the model's fine-tuning progress, I evaluated its performance using the validation set, tweaking hyperparameters such as learning rate and batch size to strike a balance between accuracy and generalizability. After the fifth epoch, the model's accuracy peaked at around **68.23%**. Both training and validation losses fluctuated over time, signaling potential overfitting. To mitigate this, I implemented a process to retain the best-performing model based on validation results, ensuring that the final model represented an optimal balance between generalization and specificity. Figure 11 shows accuracy and validation loss were plotted per epoch to visually monitor performance.

```python
# Create a Trainer object
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_dataset["train"],
    eval_dataset=tokenized_dataset["validation"],
    tokenizer=tokenizer,
    data_collator=data_collator,
    compute_metrics=compute_metrics,
)

# Train the model
trainer.train()
```

[6200/6200 11:59, Epoch 10/10]

Epoch	Training Loss	Validation Loss	Accuracy
1	0.665800	0.629888	{'accuracy': 0.6741935483870968}
2	0.655000	0.647625	{'accuracy': 0.6741935483870968}
3	0.647600	0.686557	{'accuracy': 0.6741935483870968}
4	0.659600	0.658086	{'accuracy': 0.6725806451612903}
5	0.629300	0.638937	{'accuracy': 0.682258064516129}
6	0.636900	0.641805	{'accuracy': 0.6612903225806451}
7	0.642300	0.666618	{'accuracy': 0.6612903225806451}
8	0.618900	0.659357	{'accuracy': 0.6661290322580645}
9	0.619300	0.655458	{'accuracy': 0.6612903225806451}
10	0.612300	0.658775	{'accuracy': 0.6629032258064517}

Figure 11. Per epoch performance of training the model

V. DELIVERABLE 4: BRIEF IMPACT: COMPARING DIFFERENT FINE-TUNING METHODS

This deliverable examines the experimentation and results of applying various parameter-efficient fine-tuning (PEFT) methods—LoRA (Low-Rank Adaptation), DoRA (Distributed Representation Adaptation), and QLoRA (Quantized LoRA)—on DistilBERT for classifying legal cases. The goal was to assess each method's impact on model efficiency and accuracy, with a focus on how PEFT techniques optimize model tuning, given legal cases' specific demands.

1. Overview of PEFT Methodology

LoRA introduces low-rank update matrices that reduce trainable parameters while maintaining domain-specific adaptation capabilities. This technique leverages flexibility in applying updates, enabling targeted efficiency gains. Figure 16 shows the architecture of how LoRA works.

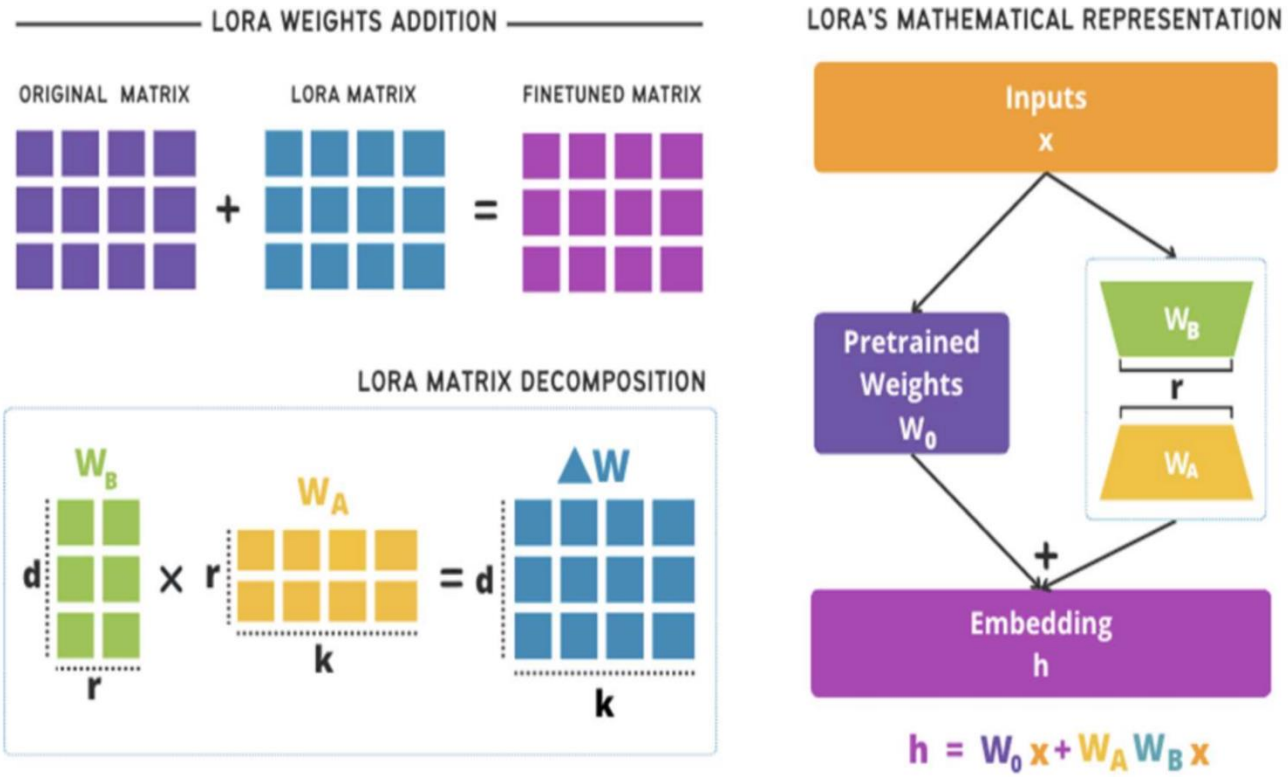


Figure 16. LoRA architectural diagram

DoRA expands LoRA's foundation by distributing low-rank update matrices across layers, which can benefit models dealing with complex data by enhancing representational adaptability. Figure 17 shows the architecture of how DoRA works.

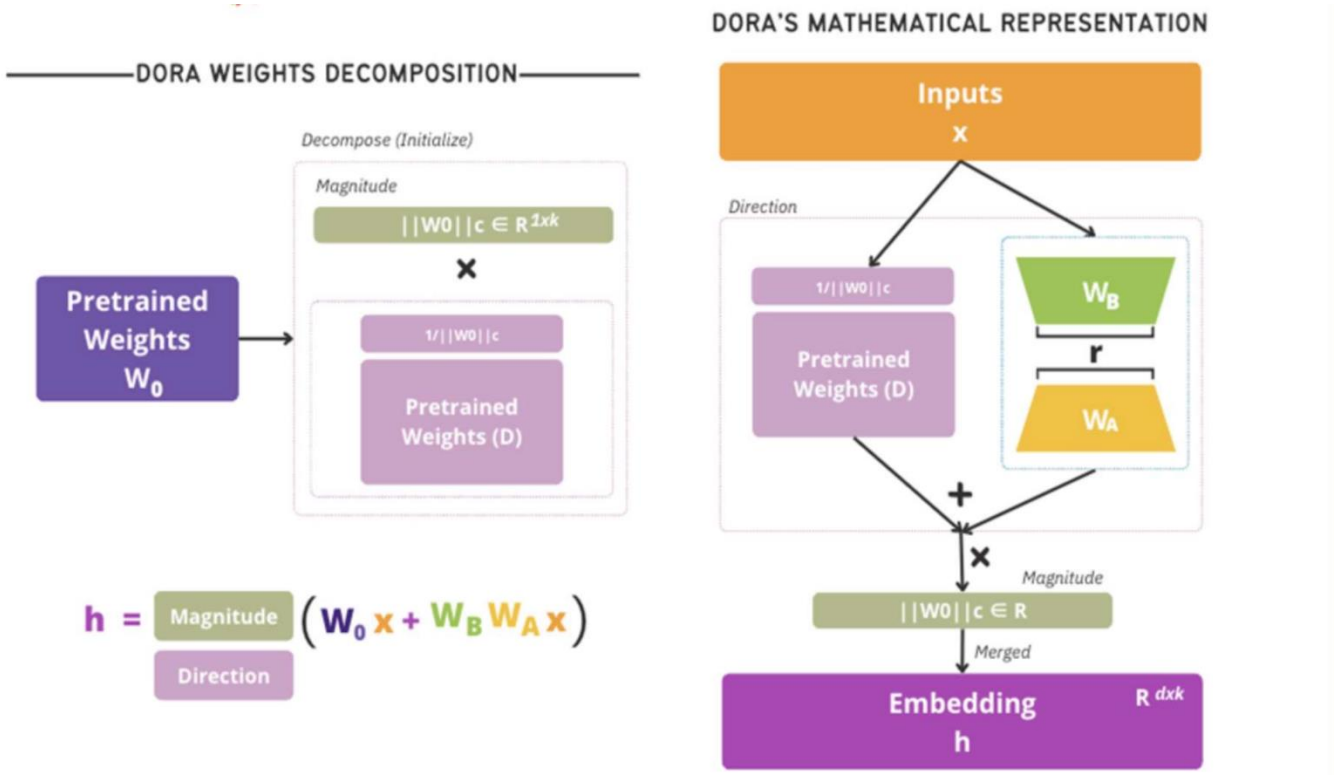


Figure 17. DoRA architectural diagram

QLoRA combines LoRA's efficient parameterization with quantization, minimizing memory footprint to allow fine-tuning on extensive datasets with constrained computational resources. Figure 18 shows the architecture of how QLoRA works.

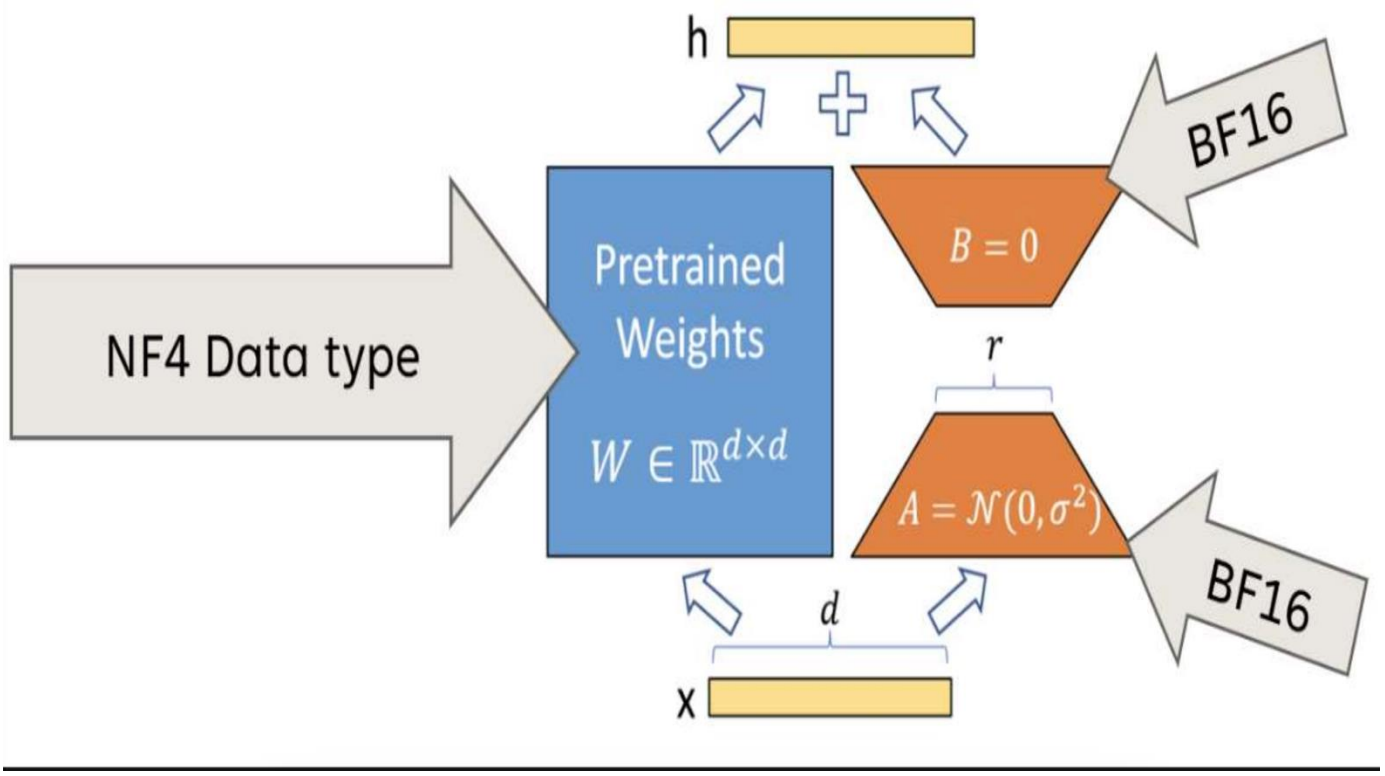


Figure 18. QLoRA architectural diagram

Figure 12 contrasts the configurations of LoRA, DoRA, and QLoRA.

Feature	LoRA	DORA	QLoRA
Adaptation	Low-rank matrices	Dynamic rank adjustment	Low-rank with quantization
Flexibility	Static rank	Adaptive rank	Static rank with quantization
Efficiency	Lower memory usage	Moderate memory usage	Lowest memory usage
Use Cases	NLP tasks	Complex data adaptation	Edge and memory-constrained tasks

Figure 12. LoRA vs DoRA vs QLoRA

2. Experimentation with LoRA, DoRA, and QLoRA on Model Components

Using the defined PEFT classes, LoRA, DoRA, and QLoRA were each applied to different

model components. This enabled a comparative analysis of the impact of each method on accuracy, generalization, and computational efficiency across 10 epochs.

3. Enhanced Evaluation Metrics

To gain a complete view of performance, the following metrics were employed:

Accuracy provides a broad measure of correct classifications.

Precision highlights how many positive predictions are correct.

Recall measures the model's ability to retrieve all relevant cases.

F1 Score balances precision and recall, useful for contexts where both false positives and false negatives are critical.

Figure 13, Figure 14, and Figure 15 show performance comparisons after ten epochs using LoRA , DoRA, and QLoRA with these expanded metrics respectively.

Epoch	Training Loss	Validation Loss	Accuracy	Precision	Recall	F1
1	0.658800	0.627240	0.672581	0.336834	0.498804	0.402122
2	0.647800	0.645574	0.674194	0.337097	0.500000	0.402697
3	0.648100	0.712681	0.674194	0.337097	0.500000	0.402697
4	0.674700	0.629367	0.674194	0.337097	0.500000	0.402697
5	0.645200	0.637002	0.664516	0.512500	0.501776	0.429370
6	0.632400	0.648755	0.667742	0.336039	0.495215	0.400387
7	0.638600	0.674489	0.667742	0.419653	0.496494	0.405001
8	0.624700	0.666173	0.669355	0.503819	0.500249	0.414607
9	0.620500	0.668748	0.654839	0.534550	0.512507	0.473291
10	0.604700	0.684949	0.658065	0.535373	0.511062	0.465726

Figure 13. LoRA accuracy run with embeddings

Epoch	Training Loss	Validation Loss	Accuracy	Precision	Recall	F1
1	0.658700	0.630643	0.674194	0.337097	0.500000	0.402697
2	0.648800	0.654548	0.677419	0.672096	0.507509	0.422332
3	0.639300	0.728879	0.672581	0.537398	0.501362	0.411404
4	0.633400	0.734037	0.653226	0.522553	0.507473	0.463017
5	0.554700	0.762792	0.583871	0.494928	0.495689	0.491713
6	0.522100	0.876906	0.624194	0.522639	0.515361	0.504213
7	0.517700	1.066743	0.595161	0.500281	0.500225	0.494200
8	0.448700	1.230635	0.596774	0.501786	0.501421	0.495305
9	0.398400	1.374483	0.564516	0.503095	0.503079	0.503081
10	0.355500	1.428831	0.575806	0.497122	0.497383	0.495781

Figure 14. DoRA accuracy run

Epoch	Training Loss	Validation Loss	Accuracy	Precision	Recall	F1
1	0.639500	0.640960	0.679032	0.696458	0.509984	0.427415
2	0.656600	0.651041	0.666129	0.447445	0.496577	0.408896
3	0.647600	0.665329	0.670968	0.461851	0.498887	0.406208
4	0.645900	0.672869	0.664516	0.519611	0.503056	0.433352
5	0.616100	0.645803	0.664516	0.577633	0.538870	0.520238
6	0.618400	0.702649	0.658065	0.532592	0.509783	0.462466
7	0.598400	0.708552	0.658065	0.540351	0.513620	0.472078
8	0.593700	0.751789	0.654839	0.544852	0.518902	0.487888
9	0.573700	0.750858	0.633871	0.526867	0.516142	0.499785
10	0.530100	0.778851	0.627419	0.514646	0.508800	0.490971

Figure 15. QLoRA accuracy run

4. Why LoRA Emerged as Most Suitable for Legal Case Prediction

Among the methods tested, LoRA provided the best balance of parameter efficiency, high

performance, and reduced computational requirements. The following points summarize its advantages:

Efficiency: LoRA's low-rank matrix updates allow considerable parameter reduction, critical for tuning large models with limited resources.

Balanced Performance: LoRA consistently demonstrated high scores across all metrics, with superior F1, precision, and recall rates.

Computational Economy: The reduced resource demands of LoRA make it particularly effective for large datasets like legal records, ensuring both high speed and low memory usage.

Interpretability: Explainable AI tools like SHAP and LIME integrated seamlessly with LoRA, offering insight into the decision process - vital for legal AI contexts.

5. Embedding Layer Optimization through layers_to_transform

By focusing LoRA transformations on specific layers (0, 2, 4, and 6), the model achieves targeted adaptation. This approach reduces memory demands, with the trade-off of slightly lower accuracy compared to full-layer fine-tuning. Below are results demonstrating the difference in model performance when additional layers are incorporated. Figure 13 shows fine-tuning using LoRA with multiple-layer embeddings.

6. PEFT Task Types

PEFT optimizes LLMs for several task types, including sequence classification (ideal for this project), token classification, question answering, text generation, and text-to-text tasks. Given the nature of legal case classification, sequence classification was identified as the most suitable, aligning well with the need for accurate category predictions.

7. Explainable AI with LoRA

Explainability Goals: Explainable AI (XAI) was employed to interpret model outputs, using SHAP and LIME to highlight case details influencing predictions. This transparency aids in establishing trust in legal decision-support systems.

Implementing SHAP:

Installation: pip install shap

Model Preparation: Apply LoRA-based fine-tuning.

SHAP Explainer: explainer = shap.Explainer(model)

SHAP Values: shap_values = explainer(test_data)

Visualization: shap.summary_plot(shap_values, test_data)

Attention maps further detailed input features influencing predictions. Figure 19 shows how explainable AI gives weightage to words and provides a result.

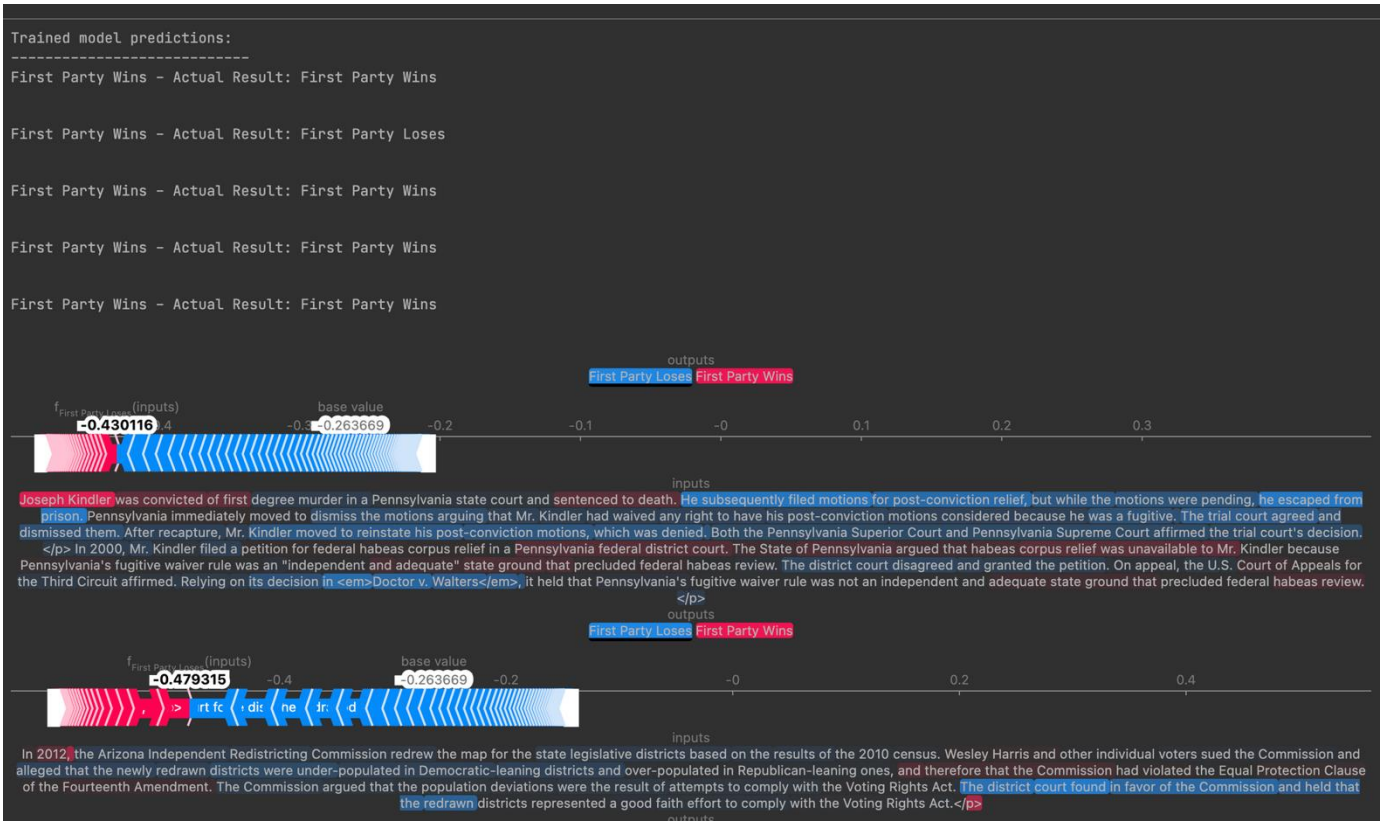


Figure 19. Explainable AI with LoRA

Insights and Future Work

LoRA consistently performed best, with an accuracy of up to 68.4%. PEFT's implementation across models revealed that efficiency and fine-tuning techniques tailored to legal data are essential for optimizing task-specific accuracy. Future directions include testing other models to explore accuracy improvements.

interest.

VI. CONCLUSION

In this project, we explored and applied various parameter-efficient fine-tuning (PEFT) methods, specifically LoRA (Low-Rank Adaptation), DoRA (Distributed Representation Adaptation), and QLoRA (Quantized LoRA), to DistilBERT for legal case classification. Each method's impact on model accuracy, efficiency, and interpretability was examined, with LoRA emerging as the most effective approach for this specialized task. LoRA's low-rank update matrices allowed for targeted adaptation without significantly increasing computational demands, demonstrating balanced accuracy, memory efficiency, and interpretability—key factors in the legal domain where transparency and precision are crucial.

Through a multi-step approach, we compared the performance of LoRA, DoRA, and QLoRA, refining their efficacy using metrics like accuracy, precision, recall, and F1 score. LoRA consistently delivered the highest performance while retaining computational efficiency, making it particularly suited to handling the large, nuanced datasets involved in legal cases. Its effectiveness in capturing domain-specific patterns and minimizing false positives and false negatives highlighted its potential to enhance automated legal prediction models responsibly. Additionally, the experimentation with LoRA-specific layer adjustments via the `layers_to_transform` parameter illustrated how selective fine-tuning can further optimize performance and resource use, providing a path to scalable, specialized adaptations for complex tasks.

Incorporating explainable AI tools such as SHAP and LIME provided transparency into the model's decision-making processes, highlighting influential case factors and contributing to a better understanding of predictions—an essential requirement for trustworthiness in legal applications. By visualizing model attention weights and interpreting SHAP values, we gained

insights into the features that drive predictions, supporting accountability and allowing for more informed evaluations.

Ultimately, this study emphasized the importance of selecting tailored PEFT techniques for specific tasks and demonstrated the role of explainable AI in enhancing the trustworthiness of AI models in legal decision support. Future research can build on these findings by exploring advanced models and further fine-tuning methodologies to improve accuracy, especially for nuanced or complex cases. This project provides a foundation for robust, interpretable legal AI applications that prioritize both efficacy and ethical considerations.

As part of CS 298, we will continue experimenting with different large language models (LLMs) and apply various fine-tuning techniques that were explored in CS 297. This includes utilizing models such as DistilBERT with LoRA, DistilBERT with QLoRA, and other parameter-efficient methods to provide a range of options for users to predict legal outcomes. Additionally, we will develop a user interface that leverages the insights gained in CS 297, allowing users to seamlessly choose and deploy these models for legal case classification. The interface will also incorporate explainable AI tools to ensure transparency, providing users with clear explanations of the factors influencing the predictions. This work will build on the foundation laid in CS 297, further refining the system to create a robust, adaptable AI-powered legal decision support tool.

REFERENCES

[1] L. K. Branting, M. Weissman, C. Brown, C. Pfeifer, S. Ferro, and J. Yeh, "Scalable and explainable legal prediction," *Artificial Intelligence and Law*, vol. 29, no. 2, pp. 213-238, 2021.

[2] R. Ejjami, "AI-Driven Justice: Evaluating the impact of artificial intelligence on legal systems," *International Journal for Multidisciplinary Research*, vol. 6, no. 3, 2024.

[3] NorthernG. A. Tsihrintzis, L. Jain, and V. G. Kaburlasos, *Advances in Artificial Intelligence-Empowered Decision Support Systems*. Springer, 2024.