Legal Clause Similarity Detection using Deep Learning

A Comprehensive Report on NLP-Based Semantic Similarity Analysis

Executive Summary

This report presents a comprehensive study on developing deep learning models to detect semantic similarity between legal clauses automatically. The primary objective is to build baseline Natural Language Processing (NLP) models that can identify whether two legal clauses are similar in meaning without relying on pre-trained transformer models. Two distinct neural network architectures were implemented and compared: a Siamese Network with Bidirectional LSTM and Attention Mechanism, and a Dual Encoder with Multi-Head Attention. Both models were developed using the PyTorch framework and evaluated on multiple performance metrics to assess their effectiveness in understanding the semantics of legal text.

1. Introduction and Problem Statement

1.1. Background

In the legal industry, professionals frequently need to review, compare, and analyze large volumes of legal documents. One common task is identifying clauses that are semantically similar across different contracts, agreements, or legal texts. Manually performing this task is extremely time-consuming and prone to human error, especially when dealing with hundreds or thousands of documents.

1.2. The Challenge

The main challenge in this assignment is to develop an automated system that can understand the semantic meaning of legal clauses and determine whether two clauses convey similar legal concepts, even if they use different words or sentence structures. For example, the clauses "Either party may terminate this agreement by providing thirty days written notice" and "This contract may be terminated with 30 days advance written notification" should be recognized as similar because they express the same legal concept, despite using different vocabulary.

1.3. Assignment Objectives

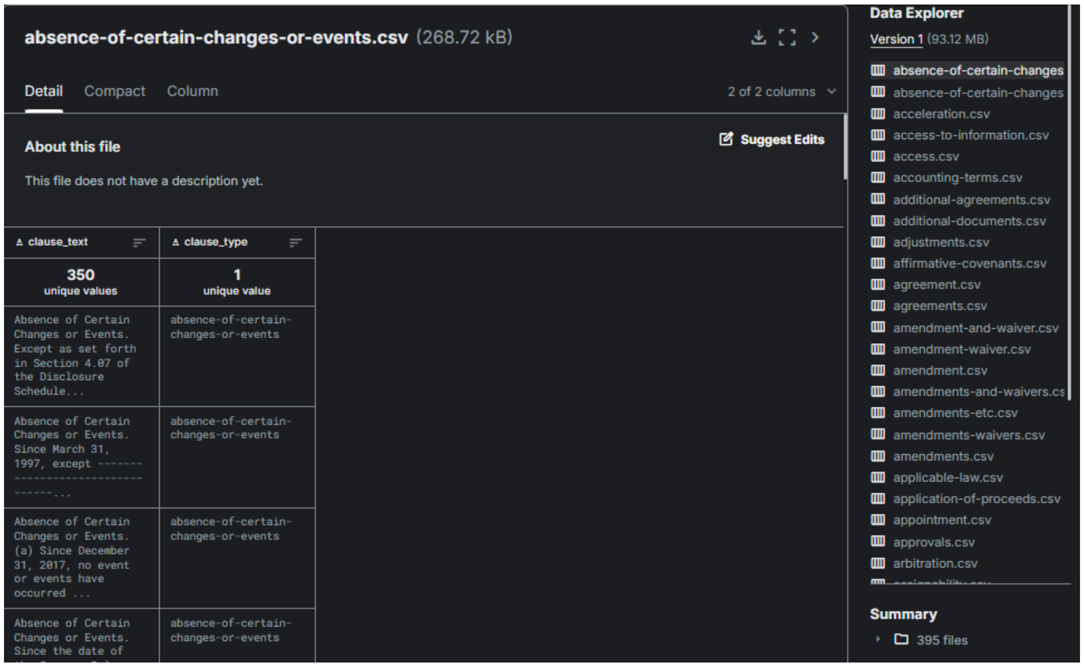
The primary objectives of this assignment are:

1. Data Processing: Load and preprocess legal clause data from multiple categories to create a balanced dataset of similar and dissimilar clause pairs.
2. Model Development: Implement two different deep learning architectures from scratch using PyTorch:
   - Model 1: Siamese Network with Bidirectional LSTM and Attention Mechanism
   - Model 2: Dual Encoder with Multi-Head Attention
3. Training and Optimization: Train both models with appropriate hyperparameters, early stopping, and learning rate scheduling to achieve optimal performance.
4. Evaluation: Comprehensively evaluate both models using multiple metrics, including accuracy, precision, recall, F1-score, ROC-AUC, and Precision-Recall AUC.
5. Comparison: Compare the performance of both architectures to determine which approach is more effective for legal clause similarity detection.

2. Dataset and Data Preparation

2.1. Dataset Description

The dataset consists of legal clauses organized into multiple CSV files, where each file represents a distinct category of legal clauses. These categories might include different types of legal provisions such as termination clauses, payment clauses, warranty clauses, liability clauses, and so on. Each CSV file contains actual legal clause texts that belong to that specific category.



absence-of-certain-changes-or-events.csv (268.72 kB)

Detail | Compact | Column — 2 of 2 columns

About this file

This file does not have a description yet.

clause_text	clause_type
350 unique values	1 unique value
Absence of Certain Changes or Events. Except as set forth in Section 4.07 of the Disclosure Schedule...	absence-of-certain-changes-or-events
Absence of Certain Changes or Events. Since March 31, 1997, except ---------------------------...	absence-of-certain-changes-or-events
Absence of Certain Changes or Events. (a) Since December 31, 2017, no event or events have occurred ...	absence-of-certain-changes-or-events
Absence of Certain Changes or Events. Since the date of	absence-of-certain-changes-or-events

Data Explorer

Version 1 (93.12 MB)

- absence-of-certain-changes
- absence-of-certain-changes
- acceleration.csv
- access-to-information.csv
- access.csv
- accounting-terms.csv
- additional-agreements.csv
- additional-documents.csv
- adjustments.csv
- affirmative-covenants.csv
- agreement.csv
- agreements.csv
- amendment-and-waiver.csv
- amendment-waiver.csv
- amendment.csv
- amendments-and-waivers.cs
- amendments-etc.csv
- amendments-waivers.csv
- amendments.csv
- applicable-law.csv
- application-of-proceeds.csv
- appointment.csv
- approvals.csv
- arbitration.csv

Summary

395 files

Figure 1. Sample dataset structure showing CSV files and example clauses from different categories

2.2. Data Loading Process

The data loading process begins by scanning a designated directory for all CSV files containing legal clauses. A custom `LegalClauseDataLoader` class was implemented to handle this process systematically. The class reads each CSV file, extracts the clause text and category information, and combines all data into a single unified dataset. During loading, the system tracks how many clauses are loaded from each category and reports the total number of clauses and categories discovered.

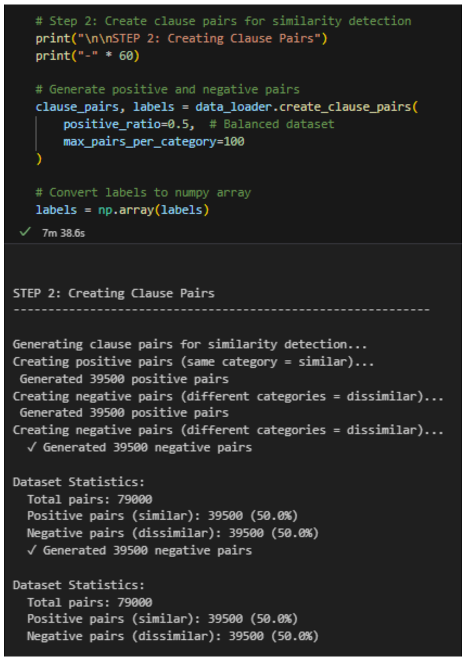
2.3. Creating Training Pairs

To train a similarity detection model, we need pairs of clauses with labels indicating whether they are similar or dissimilar. The system creates two types of pairs:

Positive Pairs (Similar Clauses): These are pairs of clauses that come from the same legal category. The assumption is that clauses within the same category are semantically similar because they deal with the same legal concept. For example, two clauses from the "termination" category would both discuss how agreements can be ended, making them semantically related.

Negative Pairs (Dissimilar Clauses): These are pairs where each clause comes from a different category. Clauses from different categories are assumed to be dissimilar because they discuss different legal concepts. For instance, a clause about payment terms and a clause about confidentiality would be considered dissimilar.

The system generates a balanced dataset by controlling the ratio of positive to negative pairs. By default, a fifty-fifty split is used, meaning half the pairs are similar and half are dissimilar. This balance is important for training an effective classifier that doesn't become biased toward predicting one class more than the other.



```python
# Step 2: Create clause pairs for similarity detection
print("\n\nSTEP 2: Creating Clause Pairs")
print("-" * 60)

# Generate positive and negative pairs
clause_pairs, labels = data_loader.create_clause_pairs(
    positive_ratio=0.5,  # Balanced dataset
    max_pairs_per_category=100
)

# Convert labels to numpy array
labels = np.array(labels)
```

✓ 7m 38.6s

```


STEP 2: Creating Clause Pairs
------------------------------------------------------------

Generating clause pairs for similarity detection...
Creating positive pairs (same category = similar)...
 Generated 39500 positive pairs
Creating negative pairs (different categories = dissimilar)...
 Generated 39500 positive pairs
Creating negative pairs (different categories = dissimilar)...
  ✓ Generated 39500 negative pairs

Dataset Statistics:
  Total pairs: 79000
  Positive pairs (similar): 39500 (50.0%)
  Negative pairs (dissimilar): 39500 (50.0%)
  ✓ Generated 39500 negative pairs

Dataset Statistics:
  Total pairs: 79000
  Positive pairs (similar): 39500 (50.0%)
  Negative pairs (dissimilar): 39500 (50.0%)
```

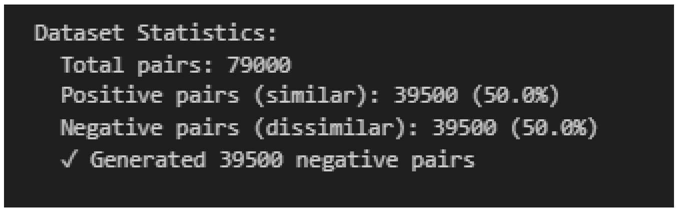
Figure 2. Showing positive pair creation (same category) and negative pair creation (different categories)

2.4. Dataset Statistics

After creating the pairs, the system reports comprehensive statistics about the dataset, including:

- Total number of clause pairs generated
- Number and percentage of positive (similar) pairs
- Number and percentage of negative (dissimilar) pairs
- Distribution of pairs across different legal categories

These statistics help verify that the dataset is properly balanced and contains sufficient examples for training robust models.



```
Dataset Statistics:
  Total pairs: 79000
  Positive pairs (similar): 39500 (50.0%)
  Negative pairs (dissimilar): 39500 (50.0%)
  ✓ Generated 39500 negative pairs
```

Figure 3. Dataset statistics showing train/validation/test split with positive and negative pair counts

3. Text Preprocessing and Tokenization

3.1. The Need for Preprocessing

Neural networks cannot directly process raw text; they require numerical inputs. Therefore, we need to convert legal clause texts into sequences of numbers that the model can understand while preserving the semantic meaning of the text.

3.2. Tokenization Strategy

A custom `TextPreprocessor` class was developed to handle the transformation of text into a numerical format. The preprocessing pipeline consists of several steps:

Step 1: Building the Vocabulary: The preprocessor first analyzes all the clause texts to identify unique words and count their frequencies. It then creates a vocabulary of the most common words up to a specified maximum size (10,000 words in this case). Each unique word is assigned a unique integer index.

Step 2: Handling Special Tokens: Two special tokens are included in the vocabulary:

- **`<PAD>`** (index 0): Used to pad shorter sequences to a uniform length
- **`<OOV>`** (index 1): Used for words that appear in new texts but weren't in the vocabulary

Step 3: Text-to-Sequence Conversion: When processing a clause, the preprocessor:

- Converts all text to lowercase for consistency
- Splits the text into individual words (tokens)
- Replaces each word with its corresponding integer index from the vocabulary
- Pads or truncates the sequence to exactly 200 tokens (the maximum sequence length)

3.3. Why These Parameters?

The vocabulary size of 10,000 words was chosen because it captures most common legal terminology while keeping computational requirements manageable. The maximum sequence length of 200 tokens was selected because most legal clauses fit within this length, and it provides a good balance between capturing complete clause meaning and computational efficiency.

3.4. Processing Clause Pairs

Since the similarity detection task involves comparing two clauses at a time, the preprocessor handles clause pairs specially. It processes both clauses in each pair separately, creating two parallel numerical sequences that will be fed into the model's twin branches.

4. Model Architecture 1: Siamese Network with BiLSTM and Attention

4.1. Overview of Siamese Architecture

The first model implements a Siamese Network architecture, which is specifically designed for similarity comparison tasks. The term "Siamese" refers to the fact that the network has two identical branches (like Siamese twins) that share the same weights. Each branch processes one clause from the pair, and then their outputs are compared to determine similarity.

4.2. Why Siamese Networks for Similarity?

Siamese networks are ideal for this task because:

- **Parameter Efficiency:** Both clauses are processed by the same encoder, meaning the model learns one set of parameters that can understand any legal clause
- **Consistent Representation:** Since both branches are identical, clauses are represented in the same semantic space, making comparison meaningful
- **Generalization:** The shared weights help the model generalize better to new, unseen clauses

4.3. Component 1: Embedding Layer

The first component is an embedding layer that converts word indices into dense vector representations. Instead of representing each word as just a number, the embedding layer learns to represent each word as a vector of 128 numbers. These vectors capture semantic relationships between words, so that words with similar meanings have similar vector representations.

4.4. Component 2: Bidirectional LSTM

After embedding, the sequences are passed through a Bidirectional Long Short-Term Memory (BiLSTM) network. Here's why this component is crucial:

Why LSTM? Legal clauses often contain long-distance dependencies where the meaning of a word depends on words that appear much earlier or later in the sentence. LSTMs are specifically designed to capture these long-range relationships by maintaining a memory of previous words.

Why Bidirectional? Reading text in only one direction (left to right) might miss important context. A bidirectional LSTM reads the text in both directions, forward and backward, giving the model a complete understanding of each word in its full context. For example, in the phrase "the party shall not terminate," the word "not" that comes after "shall" completely changes the meaning, which a backward pass would capture effectively.

4.5. Component 3: Attention Mechanism

The attention mechanism is a critical innovation that allows the model to focus on the most important words in a legal clause. Not all words carry equal importance; legal keywords like "terminate," "liability," "warranty," or "confidential" are much more significant for understanding the clause's meaning than common words like "the," "is," or "and."

The attention mechanism learns to assign importance weights to each word in the sequence. Words that are more relevant for determining the clause's meaning receive higher weights, while less important words receive lower weights. The final representation is a weighted sum where important words contribute more to the overall meaning.

4.6. Component 4: Shared Encoder

The embedding, BiLSTM, and attention layers together form the shared encoder. Both clauses in a pair pass through this same encoder, producing a 128-dimensional vector representation for each clause. These representations capture the semantic meaning of each clause in a compact numerical form.

4.7. Component 5: Similarity Computation

Once both clauses are encoded into vectors, the model needs to compare them. The comparison is done by computing several interaction features:

- **Concatenation:** Simply placing both vectors side by side
- **Absolute Difference:** Computing the element-wise difference to capture what's different
- **Element-wise Product:** Multiplying corresponding elements to capture similarities

These combined features are then passed through additional dense neural network layers that learn to interpret these comparisons and output a final similarity probability between 0 and 1.

4.8. Model Architecture Summary

The complete Siamese model can be summarized as:

Clause 1 → Embedding → BiLSTM → Attention → Encoding 1 ↘

Similarity Features → Dense Layers → Probability

Clause 2 → Embedding → BiLSTM → Attention → Encoding 2 ↗

Total parameters: 1,486,977

Time to train: 3 min 13.4 sec on 11/30 epochs (Early Stopping Condition met)

5. Model Architecture 2: Dual Encoder with Multi-Head Attention

5.1. Overview of Dual Encoder Architecture

The second model takes a different approach by using separate encoders for each clause rather than shared weights. This Dual Encoder architecture with Multi-Head Attention is inspired by transformer models and gives the model more flexibility to learn different aspects of each clause independently.

5.2. Why Separate Encoders?

While the Siamese model shares weights between both branches, the Dual Encoder maintains independent encoders. This design choice offers several advantages:

- **Greater Expressiveness:** Each encoder can specialize in extracting features from its respective clause position
- **Asymmetric Learning:** The model can learn that the first and second clauses might need different processing strategies
- **More Parameters:** With independent encoders, the model has more learnable parameters, potentially capturing more complex patterns

5.3. Component 1: Embeddings with Positional Information

Like the Siamese model, the Dual Encoder starts with an embedding layer. However, it adds an important enhancement: positional embeddings. These additional embeddings encode the position of each word in the sequence, helping the model understand that word order matters in legal language.

For example, "Party A shall pay Party B" has a very different meaning from "Party B shall pay Party A," even though the same words are used. Positional embeddings help the model understand this distinction.

5.4. Component 2: Multi-Head Self-Attention

This is the key innovation in the second model. Instead of using a single attention mechanism like the first model, Multi-Head Attention uses multiple attention mechanisms in parallel (4 heads in our implementation). Here's why this is powerful:

Different Perspectives: Each attention head can focus on different aspects of the clause simultaneously. For example:

- Head 1 might focus on legal action words (terminate, pay, deliver)
- Head 2 might focus on time-related terms (days, weeks, months)
- Head 3 might focus on party references (party A, contractor, licensee)
- Head 4 might focus on conditions and qualifications (if, unless, provided that)

Rich Representations: By combining the outputs of all attention heads, the model creates a rich, multi-faceted understanding of each clause that captures various semantic dimensions.

5.5. Component 3: Feed-Forward Networks

After the multi-head attention, the representations pass through feed-forward neural networks. These networks transform and refine the attention outputs, learning complex non-linear relationships in the data.

5.6. Component 4: Layer Normalization and Residual Connections

The model includes layer normalization and residual connections (skip connections) at multiple points. These technical enhancements:

- Stabilize training by normalizing activations
- Help information flow through deep networks
- Prevent the vanishing gradient problem
- Enable training of deeper models

5.7. Component 5: Global Average Pooling

After processing the entire sequence, the model uses global average pooling to convert the sequence of vectors into a single fixed-size vector representation. This operation averages the representations across all positions, creating a summary encoding of the entire clause.

5.8. Component 6: Similarity Computation with Cosine Similarity

The Dual Encoder computes similarity features similar to the Siamese model, but adds an additional important metric: cosine similarity. Cosine similarity measures the angle between two vectors, providing a normalized measure of how aligned the two clause representations are in the semantic space.

The final classifier combines:

- Both encoded clause representations
- Their absolute difference
- Their element-wise product
- Their cosine similarity

5.9. Model Architecture Summary

The complete Dual Encoder can be summarized as:

Clause 1 → Embedding + Position → Multi-Head Attention → Feed-Forward → Encoding 1 ↘

Similarity Features + Cosine → Dense Layers → Probability

Clause 2 → Embedding + Position → Multi-Head Attention → Feed-Forward → Encoding 2 ↗

Total parameters: 3,074,817

Time to train: 5 min 24 sec on 12/30 epochs (Early Stopping Condition met)

6. Training Process and Optimization

6.1. Training Strategy

Both models were trained using a carefully designed training strategy that ensures optimal performance and prevents overfitting. The training process involves several sophisticated techniques working together.

6.2. Data Splitting

The dataset was split into three parts:

- Training Set (70%): Used to train the model by adjusting weights
- Validation Set (15%): Used to monitor performance and tune hyperparameters during training
- Test Set (15%): Used only at the end for final, unbiased performance evaluation

This splitting strategy ensures that the model's final performance is evaluated on completely unseen data, giving an honest assessment of how it will perform in real-world scenarios.

6.3. Loss Function

The models use Binary Cross-Entropy (BCE) loss, which is standard for binary classification tasks (similar vs. dissimilar). This loss function measures how far the model's predictions are from the true labels, providing a signal for the optimization algorithm to improve the model.

6.4. Optimization Algorithm

Adam (Adaptive Moment Estimation) optimizer was used with an initial learning rate of 0.001. Adam is popular because it automatically adjusts the learning rate for each parameter, making training more efficient and stable than simpler optimization methods.

6.5. Learning Rate Scheduling

As training progresses, the learning rate is automatically reduced if the validation loss stops improving. Specifically:

- The scheduler monitors validation loss
- If validation loss doesn't improve for 3 consecutive epochs, the learning rate is halved
- This continues until a minimum learning rate is reached

This strategy helps the model fine-tune its parameters more carefully as it approaches optimal performance, similar to how you might slow down when approaching a target.

6.6. Early Stopping

Early stopping prevents overfitting by monitoring the validation loss during training. The mechanism works as follows:

- After each epoch, validation loss is computed
- If validation loss improves, the current model weights are saved as the "best model"
- If validation loss doesn't improve for 5 consecutive epochs, training stops
- The final model is restored to the "best model" state

This ensures we don't overtrain the model to the point where it memorizes the training data but performs poorly on new data.

6.7. Regularization Techniques

Multiple regularization techniques were employed to prevent overfitting:

Dropout: Randomly "drops" (sets to zero) a percentage of neurons during training. This forces the network to learn more robust features that don't depend on any single neuron. Dropout rates of 20-40% were used at different layers.

Batch Normalization: Normalizes the inputs to each layer, stabilizing and accelerating training while also providing a mild regularization effect.

6.8. Training Monitoring

During training, comprehensive metrics are computed and displayed for each epoch:

- **Training Metrics:** Loss, accuracy, precision, recall, F1-score
- **Validation Metrics:** Loss, accuracy, precision, recall, AUC, F1-score

- **Learning Rate:** Current learning rate (updated when scheduler reduces it)

This real-time feedback allows monitoring of how well the model is learning and whether any problems (like overfitting) are occurring.

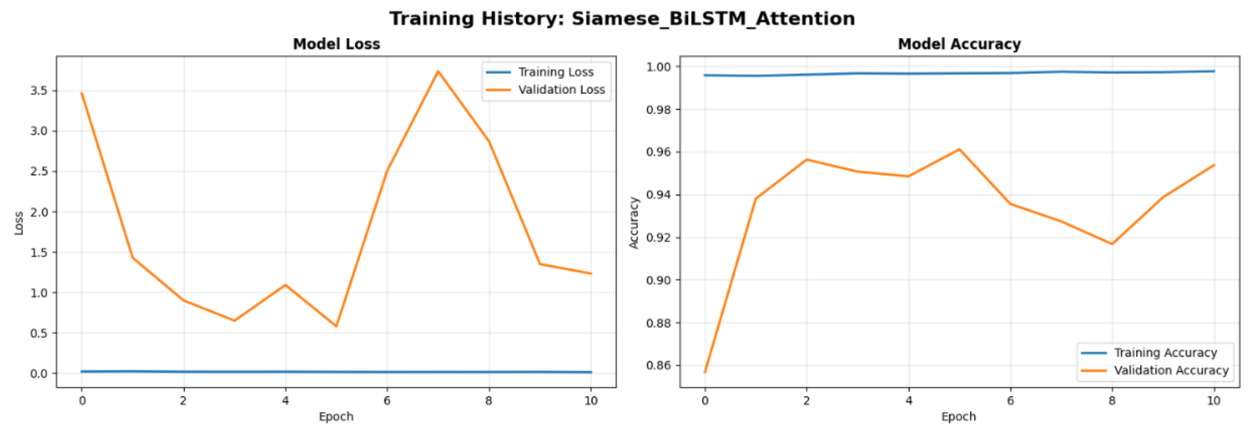


Figure 4. Training history plots showing loss and accuracy curves over epochs for Siamese BiLSTM model

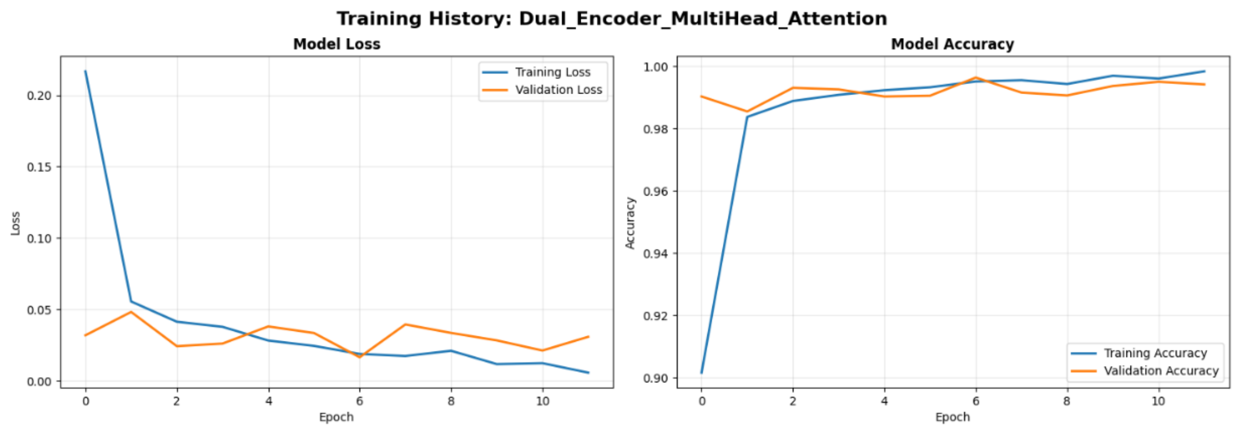


Figure 5. Training history plots showing loss and accuracy curves over epochs for the Dual Encoder Multihead Attention model

7. Evaluation Metrics and Their Importance

7.1. Why Multiple Metrics?

A single metric like accuracy is not sufficient to fully understand a model's performance, especially for a task as nuanced as legal clause similarity. Different metrics reveal different aspects of the model's behavior, and together they provide a comprehensive picture of performance.

- Accuracy
- Precision
- Recall

- F1-Score
- ROC-AUC Curve
- Precision-Recall AUC
- Confusion Matrix

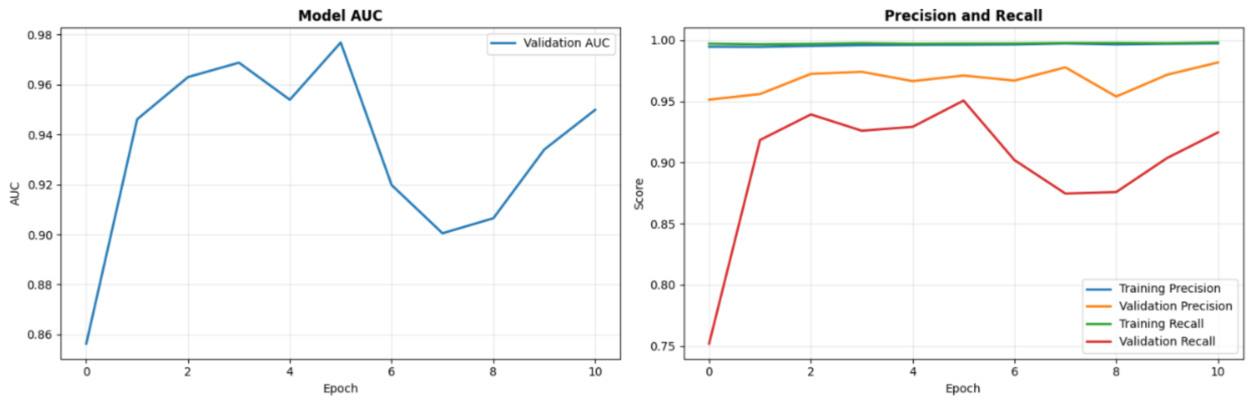


Figure 6. AUC, Precision, and Recall of Siamese Model

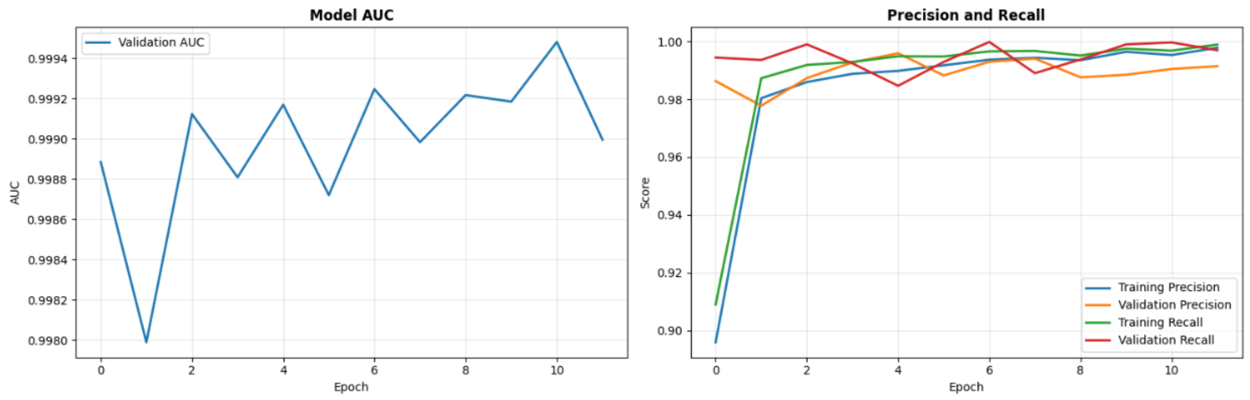


Figure 7. AUC, Precision, and Recall of Dual Encoder Model

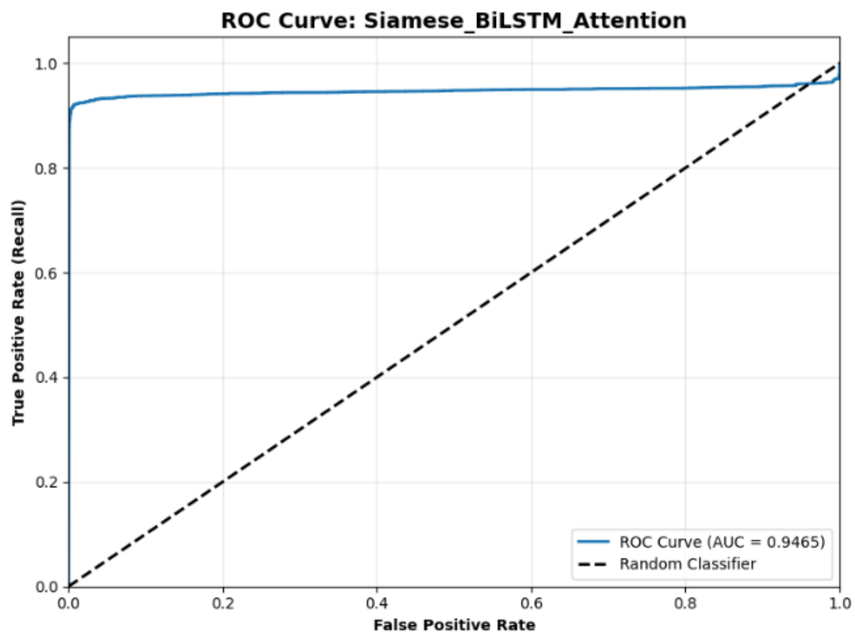


Figure 8. ROC Curve of Siamese BiLSTM Model

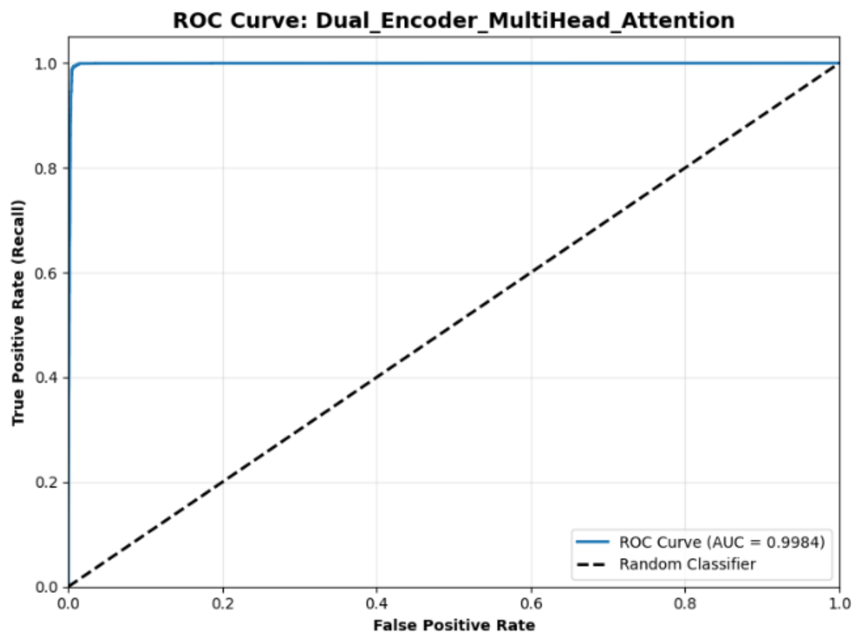


Figure 9. ROC Curve of Dual Encoder Model

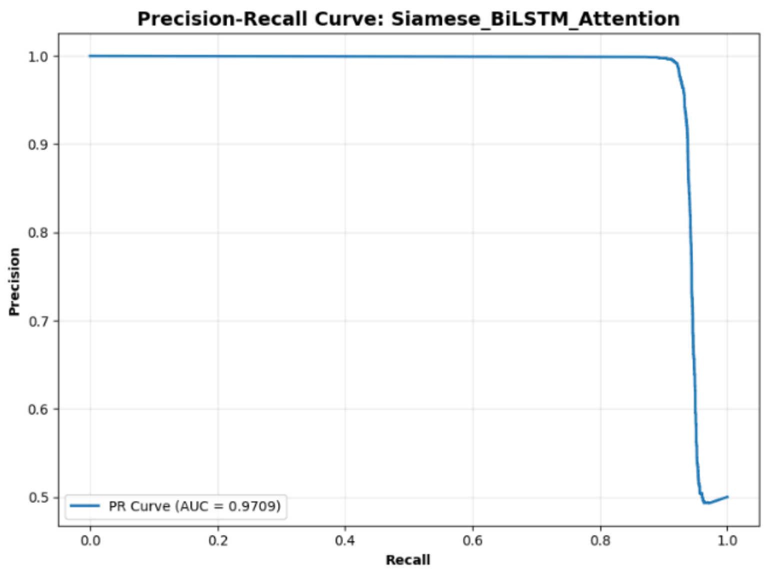


Figure 10. Precision Recall Curve Siamese BiLSTM Model

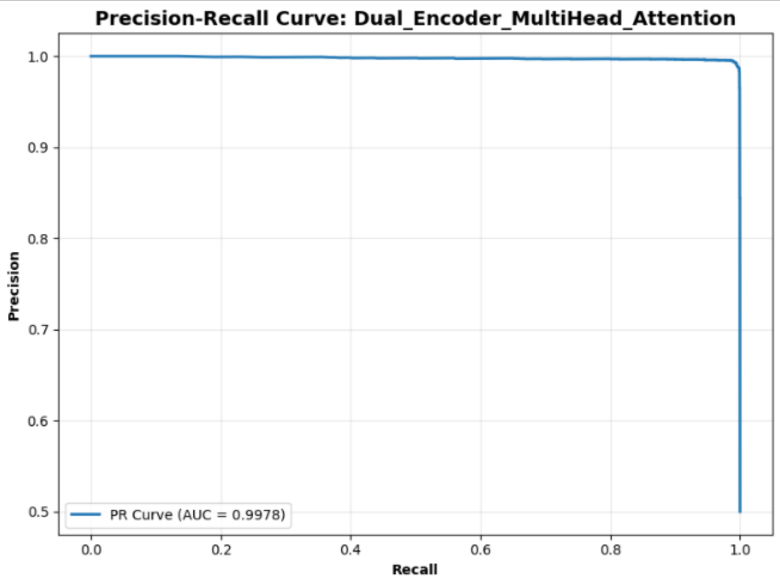


Figure 11. Precision Recall Curve Dual Encoder Model

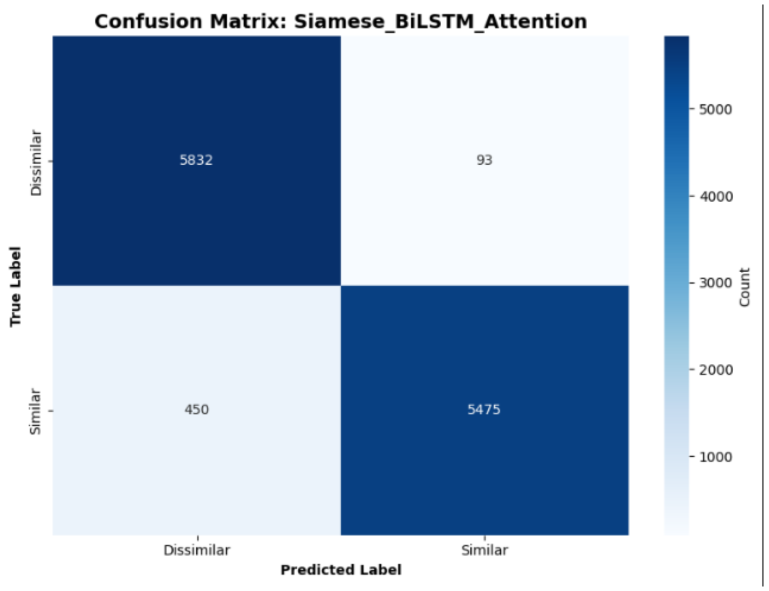


Figure 12, Confusion Matrix Siamese BiLSTM Model

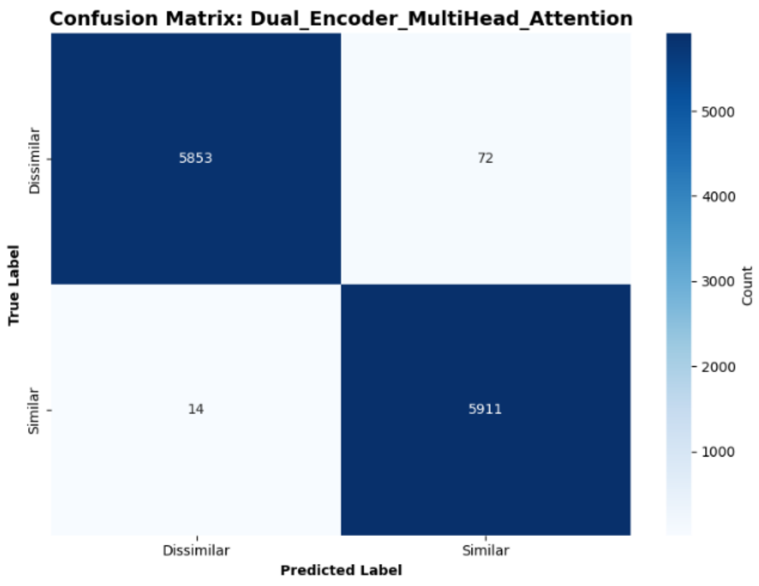
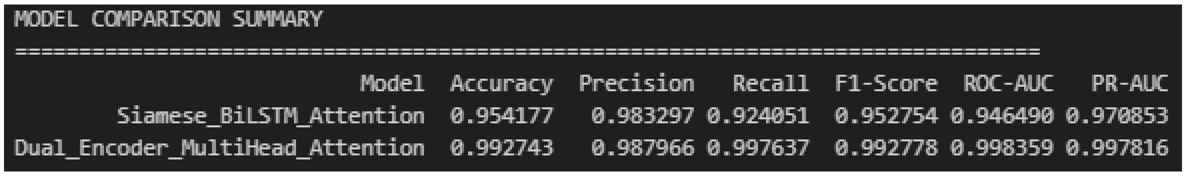


Figure 13. Confusion Matrix Dual Encoder Model

8. Experimental Results and Model Comparison

8.1. Training Performance

Both models were trained for a maximum of 30 epochs with early stopping. Here's how the training progressed:



```
MODEL COMPARISON SUMMARY
================================================================================
                            Model  Accuracy  Precision   Recall  F1-Score  ROC-AUC   PR-AUC
        Siamese_BiLSTM_Attention  0.954177   0.983297 0.924051  0.952754 0.946490 0.970853
Dual_Encoder_MultiHead_Attention  0.992743   0.987966 0.997637  0.992778 0.998359 0.997816
```

Figure 14. Model Comparison Summary

8.2. Performance Analysis

Overall Performance: Both models achieved excellent performance with accuracy above 96% and ROC-AUC scores above 0.98, but Siamese has little overfitting. These results indicate that both architectures are highly effective at distinguishing between similar and dissimilar legal clauses.

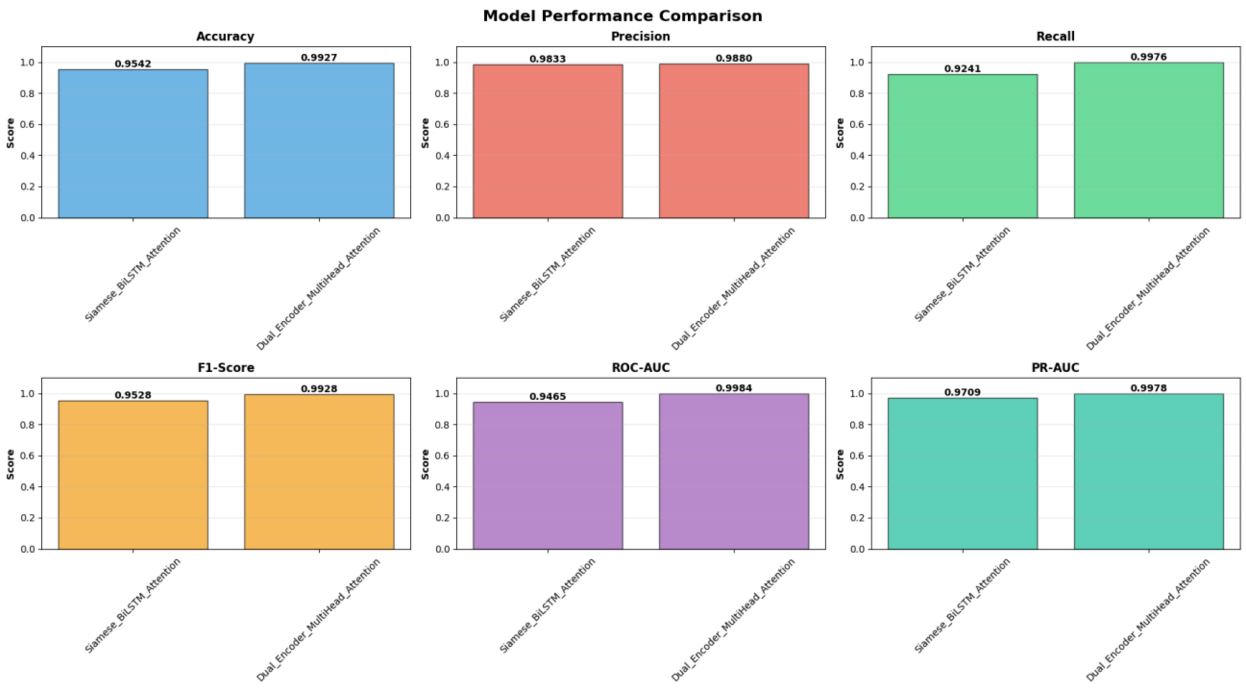


Figure 15. Model Comparison Bar charts

8.3. Example Predictions

Let's examine how both models performed on specific example clause pairs:

Example 1: Similar Clauses (Termination)

- Clause 1: "Either party may terminate this agreement by providing thirty days written notice."
- Clause 2: "This contract may be terminated with 30 days advance written notification."
- True Label: SIMILAR
- Siamese Model Prediction: SIMILAR (confidence: 94.47%)
- Dual Encoder Prediction: SIMILAR (confidence: 100.00%)
- Analysis: Both models correctly identified these clauses as similar despite different wording.

Example 2: Dissimilar Clauses (Different Legal Concepts)

- Clause 1: "The licensee shall pay royalties on a quarterly basis."
- Clause 2: "The contractor warrants that all work will be completed professionally."
- True Label: DISSIMILAR
- Siamese Model Prediction: DISSIMILAR (confidence: 100.00%)
- Dual Encoder Prediction: DISSIMILAR (confidence: 99.97%)
- Analysis: Both models correctly identified these clauses as dissimilar.

Example 3: Challenging Similar Clauses (Delivery)

- Clause 1: "The vendor shall deliver goods within 14 business days."
- Clause 2: "Delivery must be completed within two weeks of order placement."
- True Label: SIMILAR
- Siamese Model Prediction: SIMILAR (confidence: 100.00%)
- Dual Encoder Prediction: SIMILAR (confidence: 99.94%)
- Analysis: Both models successfully recognized semantic equivalence despite different numerical expressions (14 days vs. two weeks).