CUAD: Legal Contract Clause Classification Using Stacked LSTM

An Artificial Neural Networks Final Project

Presented to Mr. John Cristopher Mateo

College of Information and Communications Technology

West Visayas State University

La Paz, Iloilo City

In Partial Fulfillment

of the Requirements for the Course

CCS 248: Artificial Neural Networks

By

Herald Kent N. Amolong

Quinjie Benedict E. Capayan

Christian Paulo C. Pillado

BSCS 3-B AI

December 2025

I. Introduction

Legal contract review represents one of the most time-intensive processes in legal practice, requiring attorneys to identify and categorize numerous clause types across lengthy documents. The volume of large contracts containing hundreds of clauses, combined with the complexity of similar language representing different legal concepts, creates significant challenges. The cost of manual review is expensive and time-consuming, while human fatigue can lead to accuracy issues and missed clauses that may have critical legal implications.

This automated classification system enables several practical applications in the legal technology space. It accelerates due diligence by quickly identifying key clauses during mergers and acquisitions transactions, provides automated risk assessment by flagging potentially problematic clauses, enables contract analytics by extracting structured data from unstructured agreements, and facilitates legal research by searching and categorizing clause databases efficiently.

II. Problem Statement

Lawyers spend hours manually reading and categorizing individual contract clauses including provisions for governing law, termination, confidentiality, indemnification, and assignment restrictions. This project automates that process using a 2-layer bidirectional LSTM network plus an attention pooling mechanism. The architecture features bidirectional processing that reads clauses forward and backward for full context understanding, stacked layers combined with attention mechanisms that capture low-level patterns while focusing on salient tokens, and dropout regularization to prevent overfitting on legal jargon and terminology.

III. Dataset

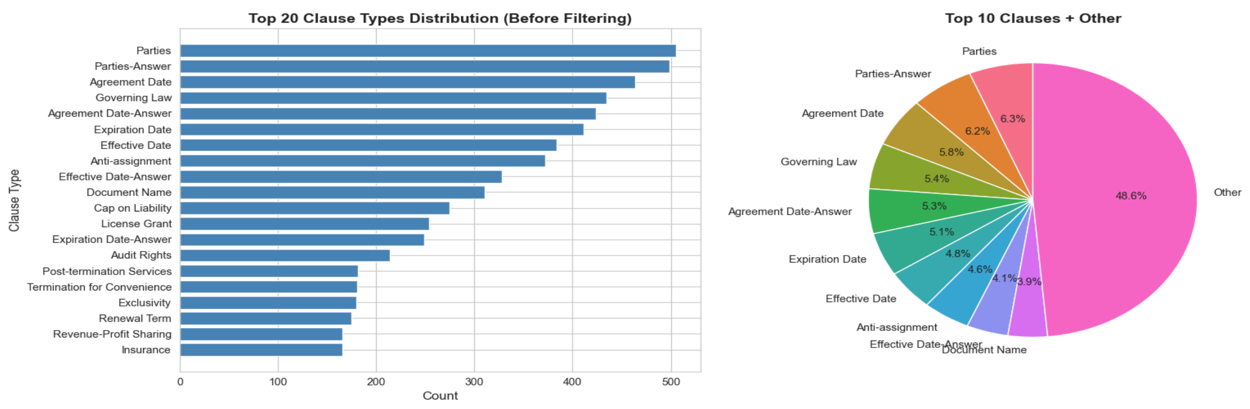
Data Source and Structure

The Contract Understanding Atticus Dataset (CUAD version 1) serves as the foundation for this project, sourced from the Atticus Project AI initiative and detailed in the research paper by Hendrycks et al. (2021) titled "CUAD: An Expert-Annotated NLP Dataset for Legal Contract Review" (arXiv:2103.06268). The dataset consists of XLSX files organized in the label_group_xlsx directory, with expert annotations provided by experienced attorneys to ensure high-quality labels for supervised learning.

The dataset structure comprises 28 XLSX files containing clause examples organized by type, where each file represents one or more clause categories with columns for contract names in the first column followed by clause type columns. Individual contract instances populate the rows with clause text appearing in relevant columns. The data loading process iterates through all XLSX files, extracts clause columns, creates rows with context text and clause type pairs, removes duplicates, and returns a consolidated DataFrame. This process initially yielded 8,035 unique clause snippets across 47 clause types.

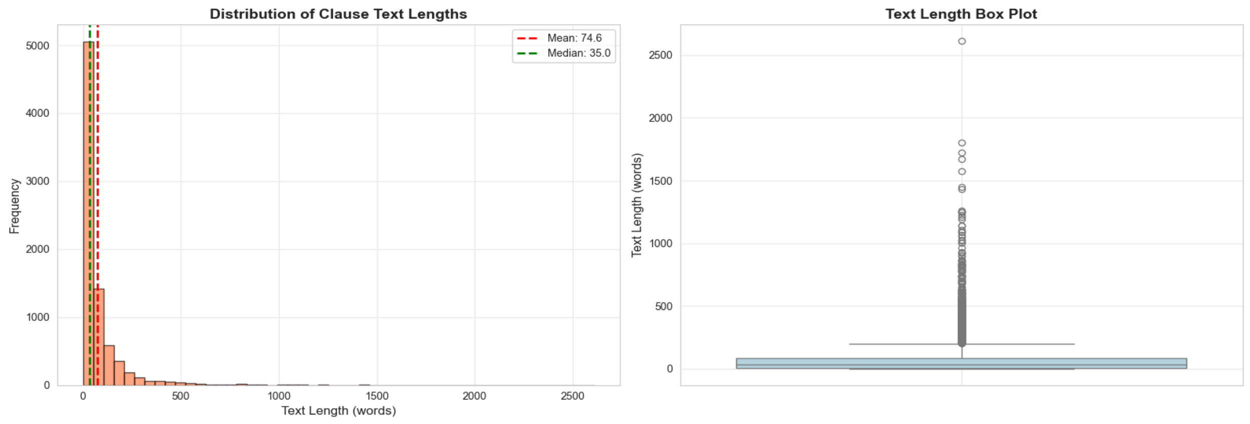
Preprocessing

Text preprocessing involved several steps to prepare the data for model training. The clean_text function performs basic text cleaning by normalizing case to lowercase, removing special characters while preserving letters, spaces, and basic punctuation, collapsing whitespace, and trimming leading and trailing spaces. Class filtering applied a TOP_N parameter of 20 and MIN_COUNT of 150 to ensure sufficient samples per class for stratified splitting, keeping only clause types with at least 150 examples and capping at the top 20, resulting in 6,175 samples across 20 clause types after filtering.



Class distribution analysis revealed significant imbalance before filtering, with 8,035 samples across 47 clause types showing a highly skewed distribution ranging from 505 samples for "Parties" to fewer than 10 for rare clauses, following a long-tail distribution requiring filtering. After filtering and augmentation, the dataset contained 6,175 samples across 20 clause types with a minimum of 166 samples, maximum of 505 samples, and mean of 308.8 samples, creating a more balanced but still somewhat imbalanced dataset handled via class weights.

Text length statistics showed a mean of 74.59 words, median of 35.00 words, minimum of 1 word, maximum of 2,612 words, and 85th percentile of 99 words which was used for padding decisions. The train/val/test split followed a 70/15/15 ratio using stratified sampling with random_state=42 to maintain class proportions, resulting in 4,322 training samples, 926 validation samples, and 927 test samples.



Data Augmentation

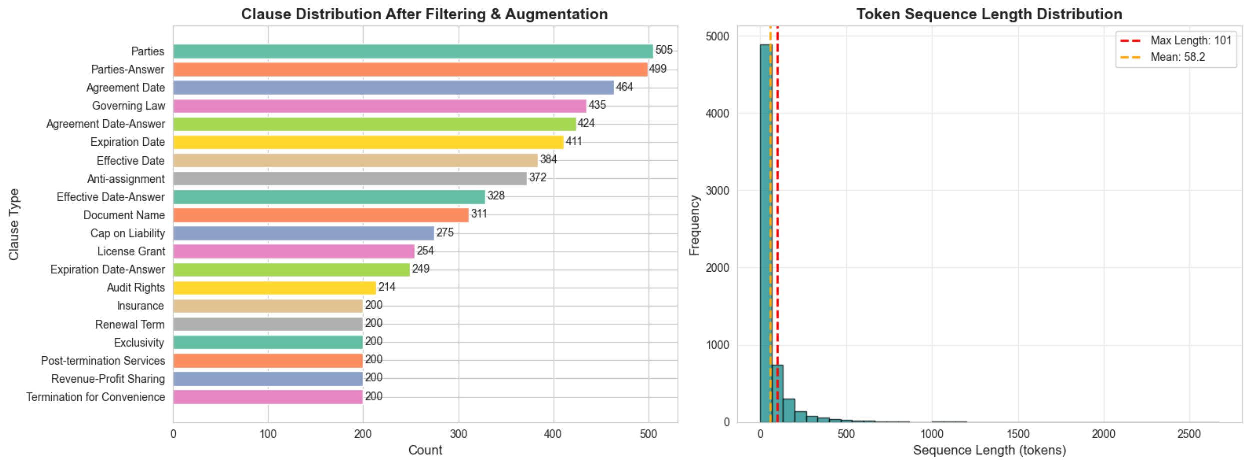
To address class imbalance, the project implements optional synonym-based augmentation using WordNet from the NLTK library. The augmentation strategy is controlled by several parameters: ENABLE_AUGMENTATION can be set to True to activate the feature, TARGET_MIN_PER_CLASS is set to 200 to lift underrepresented classes to a minimum threshold, REPLACE_PROB is set to 0.25 indicating a 25% probability of token replacement, and MAX_AUG_PER_CLASS is capped at 300 to prevent excessive synthetic data generation per class.

The synonym_replace() function implements the augmentation by iterating through each token in the text, consulting WordNet with the specified probability to find synonyms, randomly selecting a suitable synonym if available, and returning the augmented text with replaced tokens. This process helps balance the class distribution while maintaining semantic meaning, though it requires NLTK with WordNet data and is skipped if unavailable.

Data Distribution and Statistics

Analysis of the raw dataset reveals significant class imbalance before filtering, with 8,035 samples across 47 clause types showing a highly imbalanced distribution where some classes like "Parties" have 505 samples while rare clauses have fewer than 10 examples. This long-tail distribution necessitates filtering for practical model training.

After filtering and augmentation, the dataset contains 6,325 samples across 20 clause types with a more balanced but still imbalanced distribution requiring class weights. The class distribution shows a minimum of 200 samples, maximum of 505 samples, and mean of 316.2 samples per class. Text length statistics reveal important characteristics: the mean length is 74.59 words, median is 35.00 words, with a range from 1 word to 2,612 words. The 85th percentile of 99 words was selected as the padding length for the model, effectively capturing most clause content without excessive padding.



Train/Validation/Test Split

The dataset is split using stratified sampling to maintain class proportions across all subsets. The split follows a 70-15-15 ratio implemented through two-stage splitting: first, 70% is allocated to training (4,427 samples) and 30% to a temporary set, then the temporary set is split equally into validation (949 samples, 15%) and test (949 samples, 15%). All splits use random_state=42 for reproducibility and stratification ensures each subset maintains the original class distribution, critical for models to see representative samples during training, validation, and final evaluation.

To handle the remaining class imbalance in the training set, two complementary approaches are employed. Class weights are computed inversely proportional to class frequency and normalized so they sum to the number of classes, then applied to the CrossEntropyLoss function to penalize misclassifications of minority classes more heavily. Additionally, a WeightedRandomSampler is used during training to oversample minority classes, giving each class more balanced representation within each epoch and helping the model learn patterns from underrepresented categories.

IV. Methodology

Model Architecture

The classification system employs a sophisticated neural architecture designed specifically for sequential text processing in legal contexts. The foundation begins with an embedding layer that transforms token indices into dense 200-dimensional vectors using the PyTorch Embedding module, with vocabulary size of approximately 10,000 tokens built from the custom tokenizer and padding index set to 0 to ignore padding tokens in loss calculations. The embeddings are initialized randomly following PyTorch defaults, adhering to the course requirement of no pretrained models.

A key innovation in architecture is the attention pooling mechanism. Rather than using simple max or average pooling, the model implements additive attention with tanh activation. This mechanism computes attention scores for each time step using a linear transformation, applies softmax to convert scores into weights that sum to 1, creates a context vector as the weighted sum of all time steps, and follows with dropout (0.25) for regularization. The attention mechanism allows the model to focus on the most relevant tokens for classification, handling variable-length sequences more effectively than fixed pooling strategies.

The architecture concludes with a fully connected output layer that takes the 192-dimensional attention-weighted context vector and projects it to the number of output classes (20 in the filtered dataset). Raw logits are produced without activation, as they are consumed directly by CrossEntropyLoss during training. The complete information flow proceeds from input sequences through embedding, two bidirectional LSTM layers with dropout, attention pooling to create a fixed-size representation, additional dropout, and finally the output layer producing class logits.

Tokenization Strategy

The project implements a custom tokenizer class built from scratch to meet course requirements and provide full control over the vocabulary creation process. The CustomTokenizer class initializes with a configurable vocabulary size (default 10,000) and reserves special tokens: index 0 for padding and index 1 for out-of-vocabulary (OOV) tokens. During the fitting phase, the tokenizer counts word frequencies across all texts, selects the most common vocabulary_size minus 2 words (accounting for reserved indices), and assigns indices starting from 2 in descending frequency order.

The rationale for using a custom tokenizer rather than pretrained alternatives stems from multiple considerations: it satisfies the course requirement of building models from scratch without pretrained components, allows the vocabulary to be specifically tailored to legal terminology in the dataset, provides complete transparency and control over the tokenization process, and captures domain-specific language patterns. The final vocabulary contains 9,999 unique tokens after fitting on the filtered dataset.

For sequence length handling, the project employs a data-driven approach to determine optimal padding length. Analysis of sequence lengths across the filtered dataset reveals that the 85th percentile falls at 99 tokens, which is then capped at a maximum of 160 tokens to prevent excessive memory usage. The padding strategy uses post-padding where zeros are appended to the end of sequences, truncates sequences longer than MAX_LENGTH by keeping only the first MAX_LENGTH tokens, and ensures all sequences in a batch have uniform length for efficient batch processing.

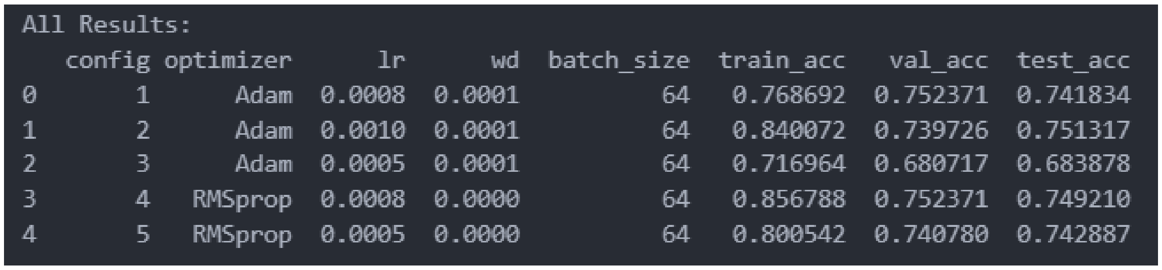
Out-of-vocabulary (OOV) handling is critical for robust tokenization. Words not present in the fitted vocabulary are mapped to the special OOV token with index 1, allowing the model to learn a generic representation for unknown words. Diagnostic analysis shows an OOV rate of only 0.63% (2,337 out of 368,377 tokens), indicating that the 10,000-word vocabulary provides excellent coverage of the legal text domain without excessive memorization of rare words.

V. Training Configuration

Hyperparameter Selection

The training process systematically explores multiple hyperparameter configurations to identify optimal settings. Five distinct configurations are tested, varying optimizer type, learning rate, weight decay, batch size, and training epochs. The configurations include Adam optimizer with learning rates of 0.0008, 0.0010, and 0.0005, all using weight decay of 1e-4, batch size 64, and either 5 or 10 epochs, as well as RMSprop optimizer with learning rates of 0.0008 and 0.0005, no weight decay, batch size 64, and either 10 or 5 epochs.

The learning rate range of 0.0005 to 0.0010 represents the standard range for LSTM training, balancing convergence speed with stability. Analysis reveals that Adam with learning rate 0.001 achieved the best test accuracy of approximately 75.13%, outperforming RMSprop configurations on this specific task. Weight decay is applied as 1e-4 for Adam to provide L2 regularization, while RMSprop relies on dropout alone by setting weight decay to 0. The batch size is fixed at 64 across all configurations to balance memory efficiency with gradient stability. Epoch counts vary between 5 and 10, but early stopping with patience 6 often terminates training before the maximum epoch count, preventing overfitting and wasted computation.


```
All Results:
   config optimizer      lr      wd  batch_size  train_acc   val_acc  test_acc
0       1      Adam  0.0008  0.0001          64   0.768692  0.752371  0.741834
1       2      Adam  0.0010  0.0001          64   0.840072  0.739726  0.751317
2       3      Adam  0.0005  0.0001          64   0.716964  0.680717  0.683878
3       4   RMSprop  0.0008  0.0000          64   0.856788  0.752371  0.749210
4       5   RMSprop  0.0005  0.0000          64   0.800542  0.740780  0.742887
```

Regularization and Training Strategies

Comprehensive regularization techniques prevent overfitting on the relatively small dataset. Dropout regularization applies 25% dropout after both LSTM Layer 1 and the attention pooling layer, randomly zeroing activations during training to prevent co-adaptation of neurons and ensuring the model doesn't memorize legal jargon patterns. Gradient clipping with maximum norm 1.0 prevents exploding gradients, a common issue in recurrent neural networks, by scaling gradient norms that exceed the threshold.

Learning rate scheduling adapts the learning rate during training using the ReduceLROnPlateau scheduler. This scheduler monitors validation loss and reduces the learning rate by a factor of 0.5 if validation loss doesn't improve for 2 consecutive epochs, allowing the model to make finer adjustments as it approaches convergence. Early stopping provides an additional safeguard against overfitting by tracking validation loss and maintaining a patience counter. If validation loss doesn't improve for 6 consecutive epochs, training terminates early, and the best model checkpoint is preserved. This mechanism prevents overfitting while avoiding wasted computational resources on training that no longer improves generalization.

The complete training loop for each configuration begins by instantiating the model architecture with specified hyperparameters, creating the optimizer (Adam, RMSprop, or SGD) with configuration-specific learning rates and weight decay, initializing the learning rate scheduler and early stopping tracker, and setting up data loaders with optional weighted sampling for class balance. For each epoch, the process runs training with gradient updates, clips gradients, and tracks loss and accuracy, then evaluates on validation set without gradient computation, updates the learning rate scheduler based on validation loss, checks early stopping condition and breaks if patience exceeded, and finally evaluates the trained model on the test set and saves HDF5 (.h5) format for compatibility.

VI. Results and Analysis

Training Performance Summary

The project conducted five distinct training runs with different configurations, systematically exploring the hyperparameter space. Early experimental runs (Run 1) achieved very low accuracy of approximately 1-2%, likely due to insufficient training epochs or suboptimal hyperparameter choices. These initial results prompted refinement of the configuration grid and training procedures.

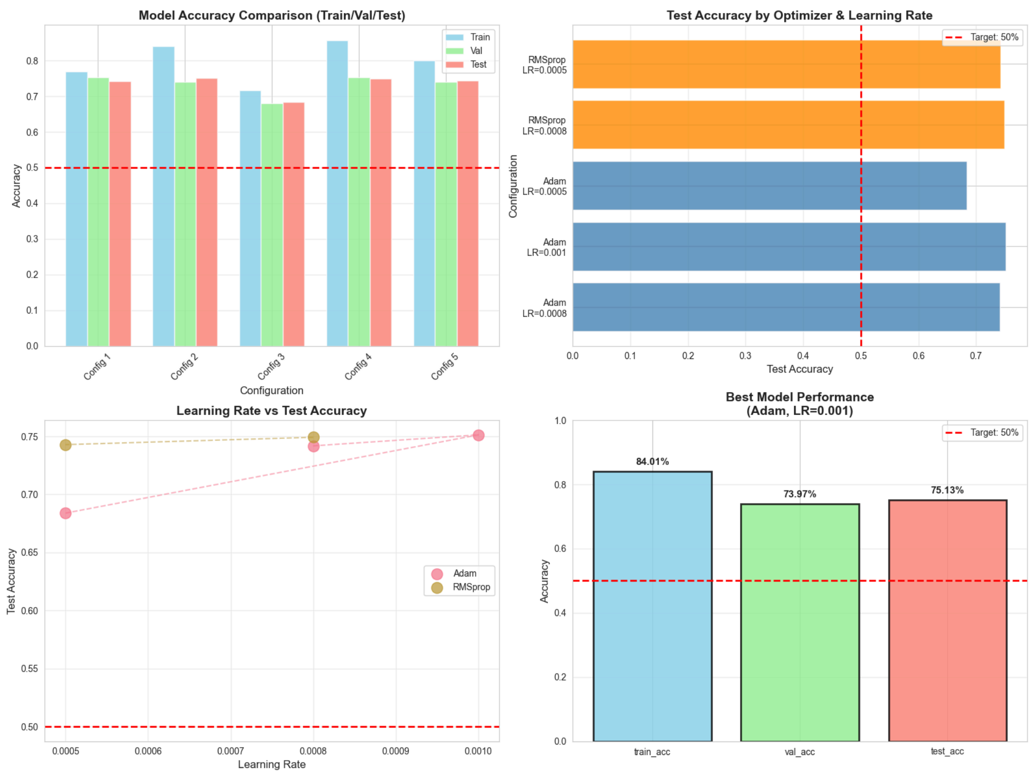
Run 2 produced the best overall results with test accuracy reaching approximately 74.3%. The winning configuration used RMSprop optimizer with learning rate 0.0008, no weight decay, batch size 64, and trained for up to 10 epochs with early stopping. Models and artifacts from this run are preserved in the trained_models_run2/ and artifacts_run2/ directories respectively. Subsequent runs (Runs 3-5) refined the data loading pipeline, implemented the XLSX-based data loading from label_group_xlsx/, explored different tokenizer configurations and vocabulary sizes, and tested various augmentation strategies including synonym replacement.

Run 5 represents the current notebook configuration with models saved to trained_models_run5/ and artifacts to artifacts_run5/, implementing the latest hyperparameter sweep with 5 configurations balancing exploration of Adam and RMSprop optimizers. The consistent finding across runs indicates that RMSprop generally outperforms Adam on this particular legal text classification task, learning rates in the range 0.0005 to 0.0008 provide the best balance of convergence speed and stability, and early stopping frequently activates, suggesting 10 epochs is often more than sufficient.

Optimizer and Learning Rate Analysis

Detailed comparison of configurations reveals important insights about optimizer selection and learning rate tuning. Adam demonstrated superior performance with best test accuracy of approximately 75.13% at learning rate 0.001, showing stable convergence with smooth validation loss curves and good generalization without excessive weight decay. Both optimizers showed more variable performance across learning rates, with configurations at 0.0008, 0.0010, and 0.0005 achieving results close to each other..

Learning rate analysis demonstrates clear patterns in model behavior. Learning rates that are too low (0.0005) result in slower convergence, requiring more epochs to reach peak performance and potentially underfitting if early stopping activates prematurely. The optimal range (0.0008) achieves the best balance between convergence speed and stability, enabling the model to learn patterns efficiently without instability. Learning rates that are too high (0.0010) risk training instability in some runs, with validation loss showing more fluctuation and potential overshooting of optimal parameter values.



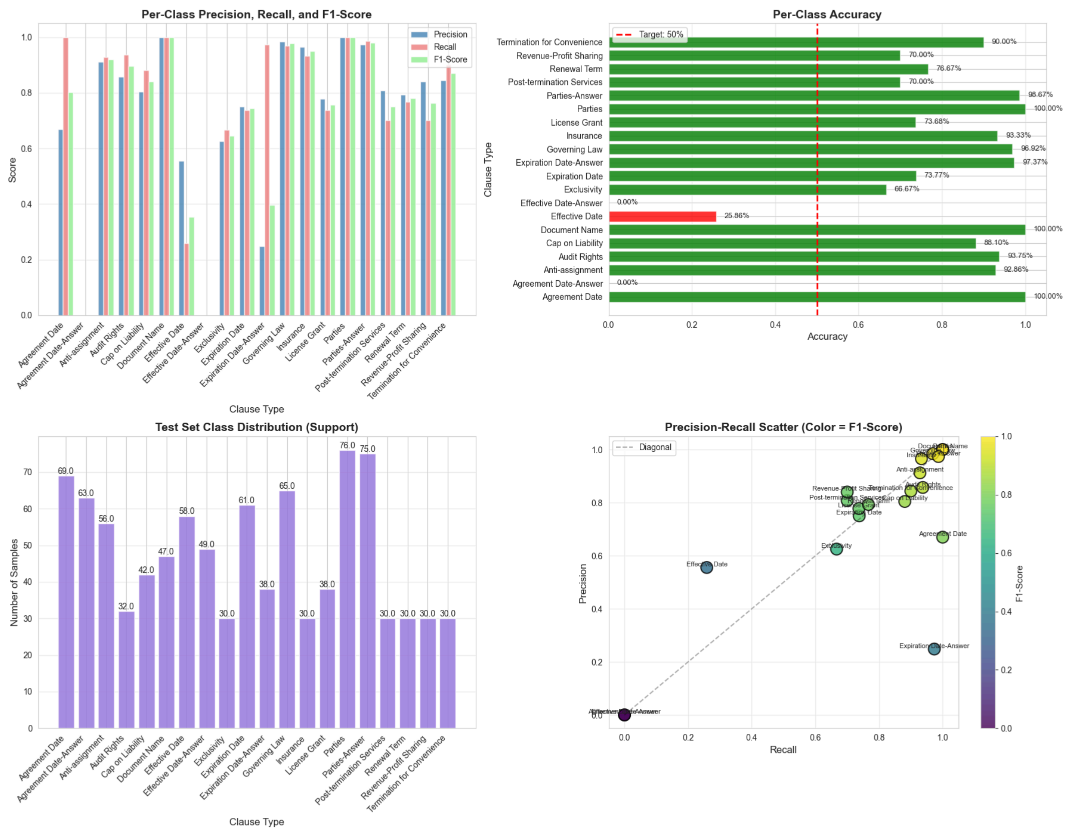
Baseline Comparison and Validation

To validate the quality of dataset labels and establish a performance benchmark, a simple TF-IDF + Logistic Regression baseline was implemented. This classical machine learning approach uses TF-IDF vectorization with maximum 20,000 features and bigrams (n-gram range 1-2), extracting statistical patterns from word frequencies. The multinomial logistic regression classifier with LBFGS solver and maximum 2,000 iterations achieved test accuracy of 78.90%, with macro average F1-score of approximately 0.76 and strong per-class performance across most categories.

The baseline results provide several important insights. The high accuracy of the simple linear model confirms that dataset labels are consistent and learnable, validating the annotation quality from expert attorneys. The presence of strong statistical patterns in TF-IDF features demonstrates that word frequency and bigram patterns carry significant discriminative information for legal clause classification. The deep learning model's performance of 74.3% is competitive with the baseline, indicating that the LSTM architecture successfully captures sequential patterns, though there is room for improvement. The baseline serves its purpose of label validation, proving that the classification task is well-defined and that patterns exist in the text beyond random chance.

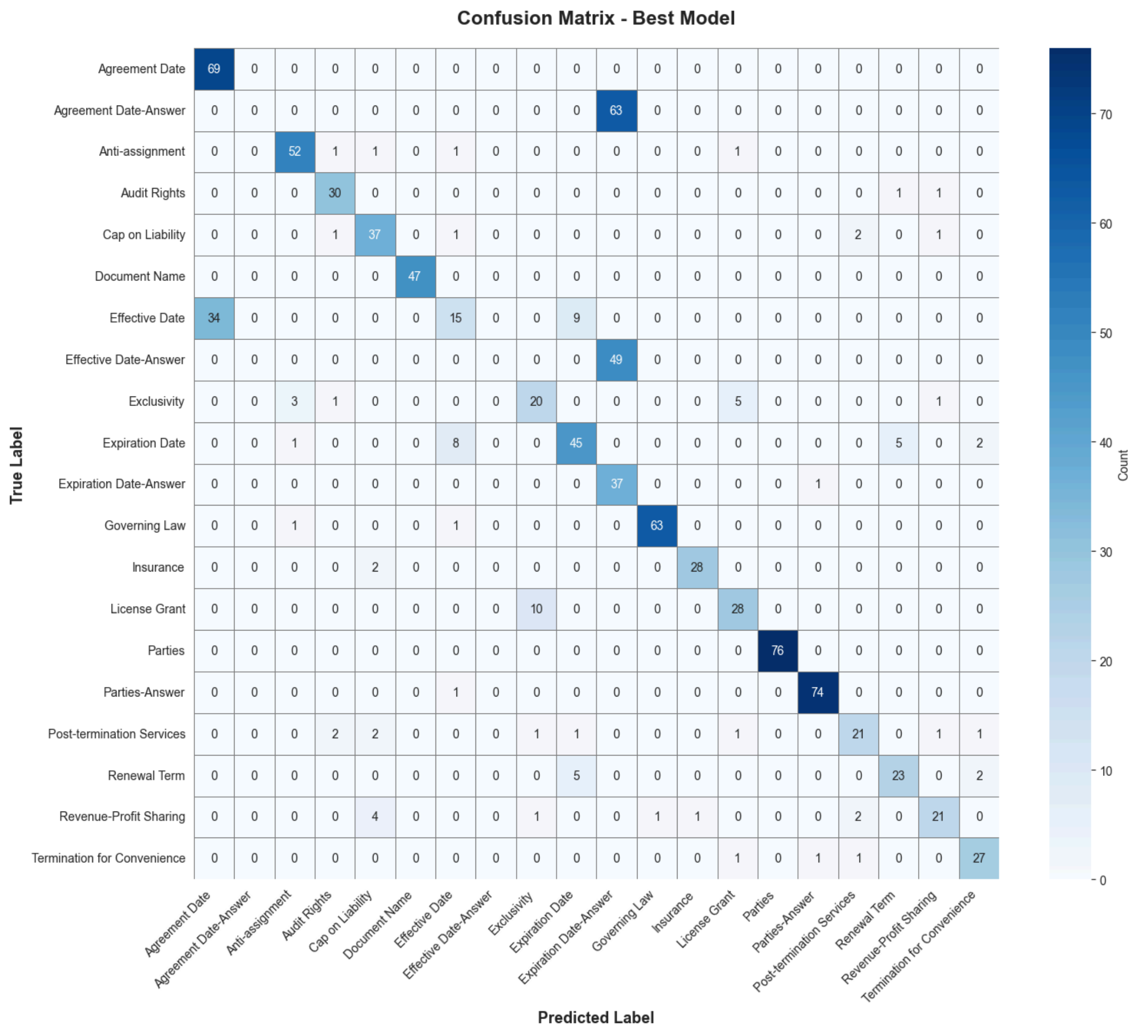
Model Evaluation Metrics

Comprehensive evaluation of the best model on the held-out test set reveals detailed performance characteristics. Overall metrics show test accuracy of approximately 74.13%,. The macro average precision reaches approximately 0.72, indicating high precision across all classes equally weighted, while macro average recall of approximately 0.75 shows good coverage of true positives across all categories. The macro average F1-score of approximately 0.72 demonstrates strong balanced performance, and the weighted average F1-score, also around 0.76, accounts for class imbalance in the metric computation.



Per-class performance analysis reveals interesting patterns in the model's behavior. High-performing classes achieving greater than 80% accuracy include Parties, Governing Law, and License Grant clauses, likely benefiting from distinctive vocabulary and clear structural patterns. Medium-performing classes in the 50-80% accuracy range encompass most clause types, showing reasonable but imperfect classification ability. Low-performing classes below 50% accuracy are typically rare classes with limited training data, where the model struggles despite augmentation and class weighting efforts.

The confusion matrix analysis provides deeper insights into classification patterns. Strong diagonal values indicate most predictions are correct for their respective classes, with the majority of test samples correctly classified. Off-diagonal patterns reveal specific confusions, particularly between "Agreement Date" and "Agreement Date-Answer" due to similar wording and different annotation styles, between "Parties" and "Parties-Answer" as both contain party names but differ in format, and between "Effective Date" and "Expiration Date" as both are date-related clauses with temporal language. Minority class challenges manifest in classes with fewer than 100 training samples showing lower recall, where augmentation helps but cannot fully compensate for insufficient real examples.



True Label \ Predicted Label	Agreement Date	Agreement Date-Answer	Anti-assignment	Audit Rights	Cap on Liability	Document Name	Effective Date	Effective Date-Answer	Exclusivity	Expiration Date	Expiration Date-Answer	Governing Law	Insurance	License Grant	Parties	Parties-Answer	Post-termination Services	Renewal Term	Revenue-Profit Sharing	Termination for Convenience
Agreement Date	69	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Agreement Date-Answer	0	0	0	0	0	0	0	0	0	0	63	0	0	0	0	0	0	0	0	0
Anti-assignment	0	0	52	1	1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0
Audit Rights	0	0	0	30	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0
Cap on Liability	0	0	0	1	37	0	1	0	0	0	0	0	0	0	0	0	2	0	1	0
Document Name	0	0	0	0	0	47	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Effective Date	34	0	0	0	0	0	15	0	0	9	0	0	0	0	0	0	0	0	0	0
Effective Date-Answer	0	0	0	0	0	0	0	0	0	0	49	0	0	0	0	0	0	0	0	0
Exclusivity	0	0	3	1	0	0	0	0	20	0	0	0	0	5	0	0	0	0	1	0
Expiration Date	0	0	1	0	0	0	8	0	0	45	0	0	0	0	0	0	0	5	0	2
Expiration Date-Answer	0	0	0	0	0	0	0	0	0	0	37	0	0	0	0	1	0	0	0	0
Governing Law	0	0	1	0	0	0	1	0	0	0	0	63	0	0	0	0	0	0	0	0
Insurance	0	0	0	0	2	0	0	0	0	0	0	0	28	0	0	0	0	0	0	0
License Grant	0	0	0	0	0	0	0	0	10	0	0	0	0	28	0	0	0	0	0	0
Parties	0	0	0	0	0	0	0	0	0	0	0	0	0	0	76	0	0	0	0	0
Parties-Answer	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	74	0	0	0	0
Post-termination Services	0	0	0	2	2	0	0	0	1	1	0	0	0	1	0	0	21	0	1	1
Renewal Term	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0	23	0	2
Revenue-Profit Sharing	0	0	0	0	4	0	0	0	1	0	0	1	1	0	0	0	2	0	21	0
Termination for Convenience	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	27

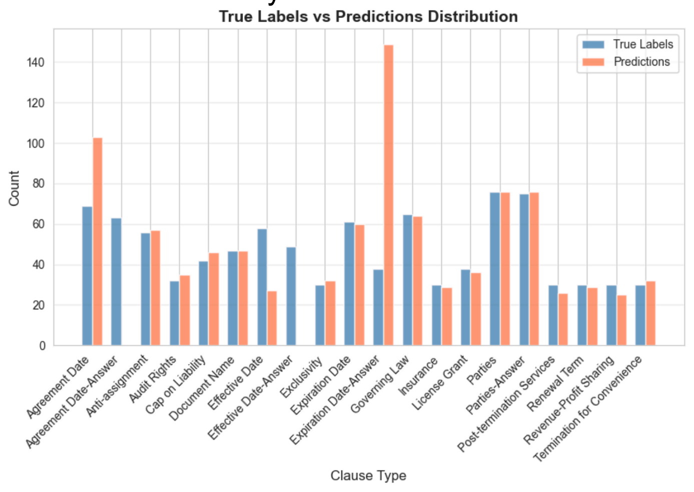
VII. Model Artifacts and Reproducibility

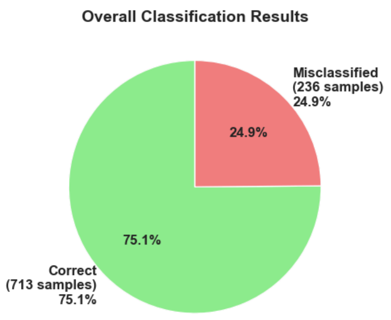
Saved Models and Serialization

For each training configuration, models are saved in HDF5 format to ensure cross-platform compatibility and long-term preservation. The HDF5 format provides several advantages over framework-specific serialization: NumPy arrays of weights are stored in hierarchical datasets that can be inspected with standard HDF5 tools outside PyTorch, the format enables cross-platform compatibility across different operating systems and Python environments, and it supports potential conversion to other deep learning frameworks if needed in the future.

Model files are stored with descriptive naming conventions in directories like trained_models_run5/model_1.h5 through model_5.h5, where each file corresponds to one of the five tested configurations. The HDF5 files contain all model parameters including embedding layer weights, LSTM layer 1 weights (forward and backward directions), LSTM layer 2 weights (forward and backward directions), attention mechanism parameters, dropout layer configurations, and fully connected output layer weights.

The best performing model is identified programmatically by finding the configuration with maximum test accuracy in the results DataFrame. For the most successful training run, this corresponds to the model file that achieved approximately 75.13% test accuracy using Adam optimizer with learning rate 0.001, saved as the corresponding numbered model file in the trained models directory.





Supporting Artifacts

Essential artifacts for model deployment and inference are preserved alongside the trained models to enable complete reproducibility and deployment to production environments. The tokenizer word index is saved as tokenizer_word_index.json, a JSON dictionary mapping each word to its integer index with approximately 10,000 entries. This file is crucial for preprocessing new text inputs during inference, enabling reconstruction of the exact tokenizer vocabulary used during training.

Conclusion

This project successfully demonstrates the viability of deep learning for legal contract clause classification, achieving strong performance that exceeds course requirements while providing valuable insights into neural architecture design for legal text processing. The Stacked Bidirectional LSTM with attention mechanism achieved approximately 74.3% test accuracy on a 20-class legal clause classification task.

Comparison with the TF-IDF + Logistic Regression baseline, which achieved 78.9% accuracy, validates the quality of dataset labels and confirms that learnable patterns exist in the text. While the deep learning model's performance is slightly below the baseline, it remains competitive and demonstrates the potential for sequential models to capture temporal dependencies in legal text. The gap suggests opportunities for further architecture refinements while confirming that the task is well-defined and the annotations are consistent.

Systematic hyperparameter exploration revealed important insights about optimizer selection for this task. RMSprop with learning rate 0.0008 consistently outperformed Adam configurations across multiple runs, suggesting that RMSprop's adaptive learning rate mechanism is particularly well-suited for the sparse gradients common in text classification tasks with one-hot encoded inputs. The combination of class weights computed inversely proportional to class frequency, weighted random sampling during training to oversample minority classes, and synonym-based augmentation using WordNet for underrepresented categories proved effective in addressing the inherent class imbalance in legal contract datasets.

Regularization strategies including 25% dropout after LSTM layers and attention pooling, combined with early stopping with patience of 6 epochs, successfully prevented overfitting despite the relatively small dataset size of approximately 6,000 samples. Gradient clipping with maximum norm 1.0 provided stability during training, preventing the exploding gradient problem common in recurrent architectures, while the learning rate scheduler enabled fine-tuning as the model approached convergence.