**Architecture and Design Choices**

The solution consists of two main components:

1. **Legal Document Classification**: Classifies legal documents into categories (e.g., contract, will, deed).
2. **Clause Matching**: Finds the most relevant paragraph in a document that matches a given clause.

Both components utilize **BERT (Bidirectional Encoder Representations from Transformers)** for text processing and feature extraction.

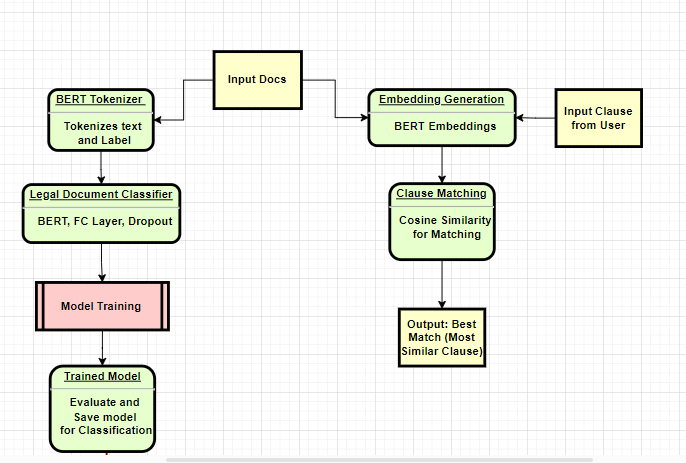
**Key Design Choices**

1. **Legal Document Classification**:
   * **BERT for Text Representation**: Uses pre-trained BERT to generate contextual embeddings of documents.
   * **Dataset Class** (LegalDocDataset): Tokenizes and prepares documents for BERT input.
   * **Model Architecture** (LegalDocClassifier): BERT + dropout + fully connected layer for classification.
   * **Training**: Fine-tuning BERT using cross-entropy loss and Adam optimizer.
   * **Assumption**: It is assumed that classified data is available for training. If not, alternative techniques such as **KNN (K-Nearest Neighbors)**, **SVM (Support Vector Machine)**, or **Naive Bayes** can be used for classification without the need for deep model training.
2. **Clause Matching**:
   * **BERT for Embeddings**: Both the clause and document paragraphs are converted into embeddings.
   * **Cosine Similarity**: Measures similarity between clause and paragraph embeddings to find the best match.
   * **Document Splitting**: Splits the document into paragraphs and compares each to the clause.
   * **Assumption**: It is assumed that the clause will be provided by the user, and the system will return the most similar clause statement from the document.
3. **File Handling**:
   * **Assumption**: For simplicity, no file data is used in this example. If data files are available, the source directory can be provided, and all files can be read according to their extension (e.g., .txt, .pdf, .docx).

**Tools and Techniques**

* **BERT**: Pre-trained transformer model for text understanding.
* **PyTorch**: Framework for model development, training, and evaluation.
* **Scikit-learn**: For dataset splitting (train-test split).
* **Cosine Similarity**: For comparing text embeddings.

**Flow Diagram:**



**Explanation:**

**Input Docs for Training**:

* Legal documents are provided as input, containing different types of legal content (e.g., contracts, wills, deeds).

**Model Training**:

* **BERT Tokenizer**: The input documents are tokenized into smaller units (tokens) using BERT's tokenizer.
* **Model Training**: The tokenized data is fed into a BERT-based classification model, where the model learns to categorize documents based on their content (using supervised learning with labeled data).

**Trained Model**:

* Once trained, the model is saved and can be used for document classification on new data.

**Legal Document Classification**:

* The trained model is used to classify new legal documents into predefined categories (e.g., contract, will, deed) by predicting the label for each document.

**Clause Matching**:

* **User Input Clause**: The user provides a clause (e.g., "confidentiality clause").
* **Embedding Generation**: The clause is tokenized and passed through BERT to generate an embedding. Similarly, embeddings for document paragraphs are generated.
* **Cosine Similarity**: The cosine similarity between the clause's embedding and each paragraph's embedding is computed to identify the most similar paragraph.

**Output**:

* **Best Match**: The system returns the paragraph with the highest similarity score to the user, showing the most relevant section of the document.