ClauseEase: AI based Contract Language Simplifier

Project Documentation

Submitted by:

Group 1

(Arnab Ghosh, Amit Badoni)

Submitted to:

Mentor: Dr. A. Kalaivani

Infosys SpringBoard

Infosys SpringBoard Virtual Internship 6.0

Table of Contents

Abstract

Legal contracts, typically written in dense and complex language, present significant accessibility challenges for individuals without legal expertise. ClauseEase addresses this critical problem by introducing an Al-powered system designed to automatically extract, analyze, and simplify legal clauses into plain English. Leveraging advanced technologies such as PyTorch **[6]**, Transformers **[5]**, spaCy **[3]**, and NLTK **[4]**, integrated with Flask for the backend and a user- friendly frontend, ClauseEase offers multi-format document uploads (PDF **[9]**, DOCX **[10]**, TXT), sophisticated Al-driven contract understanding, and precise simplification of complex legal terminology. The system's impact is profound: it significantly enhances contract readability and accessibility, empowering all users to make more informed decisions with 'Clear Contracts. Confident Decisions.'.

1. Introduction

1.1 Problem Statement

The inherent complexity and specialized jargon within legal contracts create substantial barriers to understanding for a broad audience. This often leads to misinterpretations, increased reliance on legal professionals for simple queries, and a general lack of transparency, hindering efficient business operations and individual comprehension.

1.2 Objectives

- Automate the extraction and preprocessing of contract text.
- Identify and classify legal clauses using Natural Language Processing (NLP) and Artificial Intelligence (AI) models.
- Recognize complex legal terms and generate clear, plain-English explanations.
- Simplify contract language while meticulously preserving its original legal accuracy and intent.
- Provide users with an intuitive and accessible interface for contract analysis and simplification.

1.3 Proposed Solution

The ClauseEase system aims to democratize access to legal information by developing an AI-powered platform. This platform will meticulously process legal documents, making them accessible and understandable to non-lawyers without compromising the critical legal intent and accuracy inherent in the original contracts.

1.4 Outcomes

- Enhanced comprehension of complex legal and policy texts without specialized knowledge.
- Automated simplification to reduce manual effort in processing lengthy documents.
- Improved accessibility of critical information for non-expert audiences.
- Granular control through multiple levels of text simplification.
- Readability assessment of both original and simplified documents.
- Secure and managed access via user authentication and an admin panel.

2. Project Overview

ClauseEase is an innovative AI-driven solution developed to bridge the gap between complex legal documents and the general public. The system is engineered to parse, understand, and rephrase intricate legal clauses into easily digestible language, fostering greater transparency and enabling informed decision-making for all stakeholders.

2.1 Stakeholders

The primary stakeholders for ClauseEase include:

- **General Users:** Individuals seeking to understand contracts without legal backgrounds.
- **Non-lawyers:** Professionals in business, startups, and other fields who frequently encounter legal documents.
- **Lawyers and Legal Researchers:** Professionals who can leverage the tool for faster initial reviews and clause identification.
- **Startups and Business Professionals:** Requiring clear understanding of agreements for business operations.
- **Law Students:** As an educational tool to aid in contract comprehension.

3. Module Descriptions

The ClauseEase system is structured around the following five primary Al and NLP modules:

1. **Document Ingestion:** Handles uploading and extracting text from PDF **[9]**, DOCX **[10]**, and TXT files.
2. **Text Preprocessing:** Cleans and tokenizes raw text using re and nltk **[4]** to prepare it for Al analysis.
3. **Legal Clause Detection:** Uses the nlpaueb/legal-bert-base-uncased model **[1]** to identify and classify legal clauses.
4. **Legal Term Recognition:** Employs the spacy model en_core_web_sm **[3]** to recognize and flag complex legal terminology.
5. **Language Simplification:** Leverages the google-t5/t5-small model **[2]** to rewrite complex legal text into plain English.

3.1 Document Ingestion

- **Purpose:** To handle the upload and initial extraction of text from various document formats.
- **Features:** Supports multi-format uploads including PDF, DOCX, and TXT.
- **Tech:** Utilizes fitz (PyMuPDF) **[9]** for PDF text extraction and python-docx **[10]** for DOCX text extraction.

3.2 Text Preprocessing

- **Purpose:** To clean and prepare the extracted text, making it suitable for Al model consumption.
- **Features:** Removes noise, normalizes text, and segments text into meaningful units.
- **Tech:** Employs regular expressions (re) for cleaning and nltk **[4]** for sentence tokenization.

3.3 Legal Clause Detection

- **Purpose:** To identify and classify specific legal clauses within the contract text.
- **Features:** Leverages advanced NLP techniques to accurately pinpoint and categorize different types of legal clauses.
- **Tech:** Uses the nlpaueb/legal-bert-base-uncased model **[1]**.

3.4 Legal Term Recognition

- **Purpose:** To recognize and identify complex legal terminology present in the contract.
- **Features:** Accurately flags specialized legal terms and potentially provides context.
- **Tech:** Uses spacy **[3]** with the en_core_web_sm model.

3.5 Language Simplification

- **Purpose:** To translate complex legal language into clear, understandable plain English.
- **Features:** Rewrites sentences and paragraphs to improve readability while maintaining legal intent.
- **Tech:** Employs the google-t5/t5-small model **[2].**

4. System Requirements

4.1 Hardware Requirements

Component	Minimum Requirement	Recommended Requirement	Purpose
Processor (CPU)	Intel Core i3 or equivalent	Intel Core i5/i7 or higher	Handles AI model computation and backend processing.
RAM	4 GB	8 GB or higher	Required for running NLP and AI models smoothly.
Storage	500 MB free space	1 GB or more	For storing project files, models, and temporary uploads.
Graphics (GPU)	Optional	NVIDIA GPU (CUDA support)	Speeds up AI model execution (optional but beneficial).
Internet Connection	Required	Stable broadband	For connecting to Firebase and downloading model dependencies.

4.2 Software Requirements

Software Component	Specification / Version	Purpose
Operating System	Windows 10 / 11, macOS, or Linux	Platform to run the application.
Frontend	HTML, CSS, JavaScript	Ensure compatibility and responsiveness,
Python Libraries	Flask, firebase-admin, torch, transformers, nltk, spacy, fitz (PyMuPDF), python-docx, numpy, scikit-learn, re, json, os	Required for AI processing, NLP, file handling, and web server.
Web Browser	Google Chrome / Edge / Firefox (latest version)	For accessing the frontend interface.
Database	Google Firebase (Cloud Firestore)	For storing user data and document results.
IDE / Code Editor	VS Code / PyCharm / Jupyter Notebook	For development and testing of Python code.

5. System Architecture

ClauseEase employs a robust, multi-layered architecture designed for efficient processing and user interaction.

Description

The system utilizes a Flask backend, which serves as the central hub for integrating various AI models responsible for clause detection and language simplification. A dynamic HTML, CSS, and JavaScript frontend provides an interactive and intuitive user interface. For secure and scalable data management, Firebase Firestore **[7]**, a NoSQL cloud database, is used to store user data and processed contract analyses.

5.1 Communication Flow

User requests originate from the Frontend, which are then sent to the Flask Backend. The backend orchestrates calls to the AI models (hosted via Hugging Face's Transformers **[5]** library or similar). Results are processed and returned to the Frontend. Data is persistently stored and retrieved from Firebase Firestore **[7]**, enabling real-time access to analysis results.

Diagram

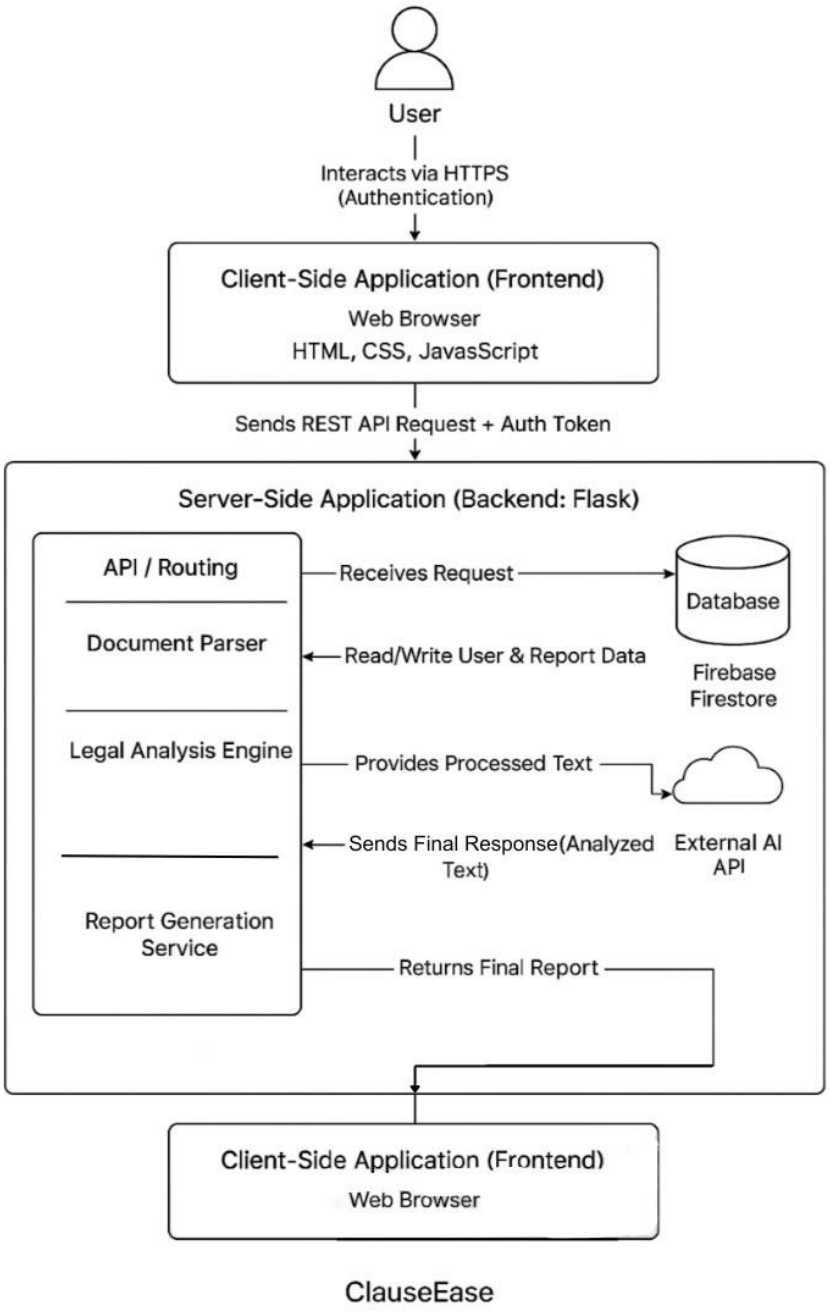

Fig.1: System Architecture

6. System Workflow

6.1 Data Flow Diagram – Level 0

The Level 0 DFD provides a high-level context view of the ClauseEase system. The system interacts with a single external entity: the **User**.

Process Overview

- The **User** uploads a contract to the ClauseEase system.
- The system processes the uploaded document and generates:
    - Simplified contract text
    - Reports / downloadable outputs
- The system communicates with an **external Database / Storage** to save and retrieve contract data.

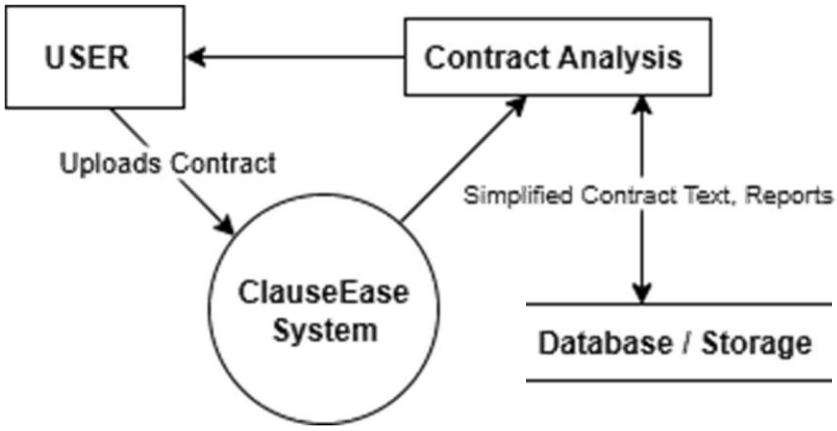

Fig.2: Level 0 DFD

6.2 Data Flow Diagram – Level 1

The Level 1 DFD details the internal workflow after a user is authenticated via **JWT authentication**.

Workflow Steps:

1. **User Authentication:** A user, authenticated with a JWT, interacts with the system.
2. **Document Ingestion:** The user uploads a contract document (PDF, DOCX, TXT) via the Document Ingestion module.
3. **Text Preprocessing:** The uploaded document's text is cleaned and prepared by the Contract Text Preprocessing module.
4. **AI Analysis:** The preprocessed text undergoes analysis through Legal Clause Detection, Legal Term Recognition, and Language Simplification modules.
5. **Database Storage:** The final analysis results are stored in the Firebase Firestore **[7]** database.
6. **Report Generation:** A downloadable report is generated and presented to the user

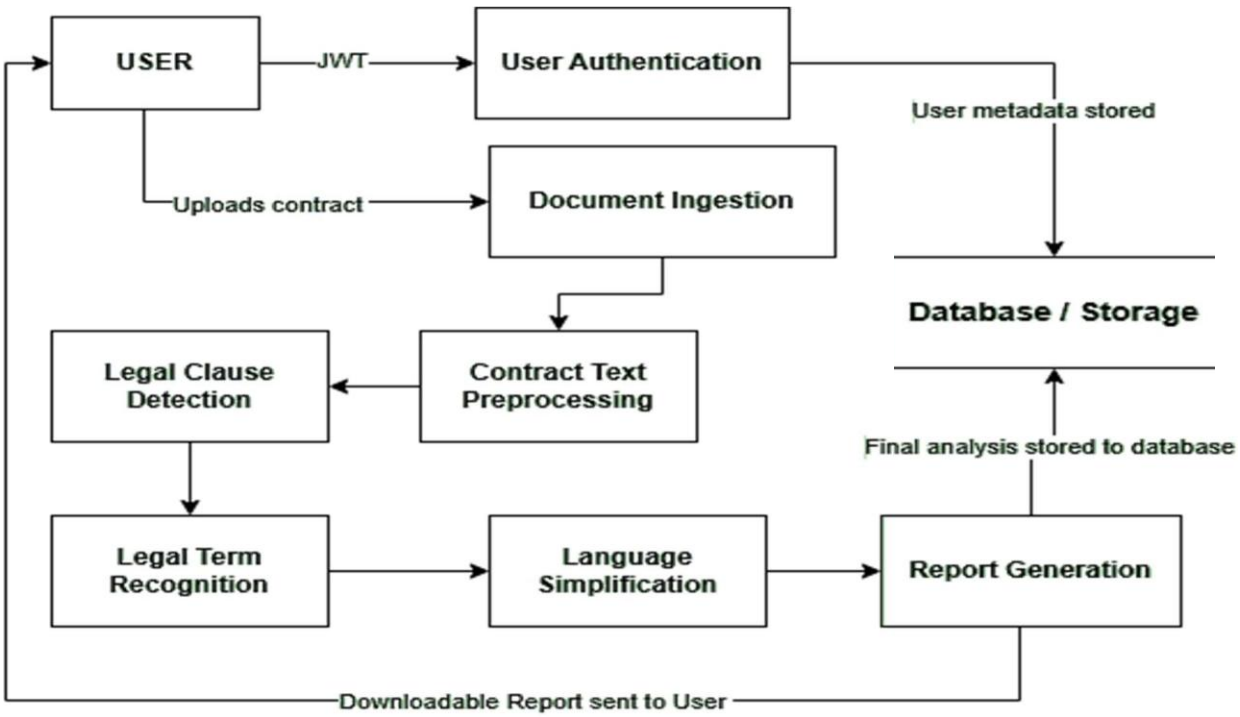

Fig.3: Level 1 DFD

7. Database Design

ClauseEase utilizes Google Firebase Cloud Firestore for its database needs. Firestore is a flexible, scalable NoSQL cloud database that synchronizes data across users in real-time.

7.1 Database Choice

A **NoSQL database** like Firestore **[7]** is well-suited for this project due to its schema flexibility, which allows for dynamic data structures as the project evolves, and its native support for real-time synchronization and scalability, essential for a cloud-based application.

7.2 Collections and Schema

users (Collection): Stores core user account information.
- Fields:
  - **metadata (Object):** Stores user profile information.
    - **name** (String)
    - **email** (String, UK)
  - **password_hash** (String)
  - **created_at** (Timestamp)

users → {user_id} → analyses (Subcollection): Stores each contract analysis performed by a user, linked to their user document.
- Fields:
  - **results_path** (String): The storage path for the full analysis JSON file.
  - **uploaded_at** (Timestamp)
  - **user_email** (String, FK): The user's email, used for reference.
  - **details (Object):** Contains file-specific metadata.
    - **file_name** (String)
    - **clause_count** (Number)
    - **simplified_text_count** (Number)

7.3 Diagram Reference

The below diagram provides a visual representation of the schema detailed in Section 7.2. It clearly illustrates the core data structure, showing how the analyses subcollection is nested within each user document. This model is key to ensuring user data is both scalable and secure.

7.3.1 Indexing:

a. PK: Primary Key

b. UK: Unique Key

c. FK: Foreign Key

7.3.2 Structure of the Database:

```
users (collection)
│
├- user_id_1 (document)  (PK)
│   ├── metadata:
│   │    ├── name: "Arnab"
│   │    ├── email: "arnab@test.com” (UK)
│   │    ├── password_hash: "... “
│   │    └── created_at: "2025-10-08"
│   │
│   └── analyses (subcollection)
│        ├── analysis_id_1 (document) (PK)
│        │     ├── uploaded_at: "2025-10-08 18:20"
│        │     ├── results_path: "/uploads/contract1.results.json"
│        │     ├── user_email: "arnab@test.com” (FK)
│        │     └── details:
│        │          ├── file_name: "contract1.pdf"
│        │          ├── clause_count: 12 (PK)
│        │          └── simplified_text_count: 12
```

8. Project Implementation Plan

The project is structured over an 8-week development roadmap, ensuring systematic progress and timely completion of key milestones.

Week	Phase	Key Milestones	Activities
1-2	**Foundation & Setup**	Project environment setup, basic UI/UX design, database schema definition.	Research, tool installation, initial backend/frontend scaffolding, user authentication setup.
3-4	**Core Processing & Ingestion**	Document ingestion module complete, text preprocessing pipeline functional.	Implement fitz and python-docx for document upload; develop text cleaning and tokenization logic with re and nltk.
5-6	**AI Integration**	Legal Clause Detection and Language Simplification modules integrated and tested.	Integrate legal-bert-base-uncased for clause detection; integrate google-t5/t5-small for simplification; integrate spaCy for term recognition.
7	**Reporting & Refinement**	Report generation complete, initial user feedback integration.	Develop report generation feature; conduct internal testing and refine modules based on initial results.
8	**Finalization & Deployment**	System tested, documented, and deployed using Docker.	Comprehensive testing (unit, integration, performance), final documentation, Docker image creation, and deployment.

9. Testing and Implementation

Testing Procedures

A rigorous testing methodology is employed to ensure the quality, performance, and reliability of ClauseEase:

- **Unit Testing:** Individual modules (e.g., document ingestion, preprocessing, specific AI model inference functions) are tested independently using frameworks like pytest to verify their correctness and isolated functionality.

- **Integration Testing:** The seamless interaction between different components (e.g., Flask backend with AI models, frontend with backend API, backend with Firebase Firestore **[7]**) is tested to ensure data flow and communication are accurate.
- **API Testing:** The backend APIs are tested using tools like Postman to validate request/response formats, error handling, and data integrity.
- **User Interface Testing:** The frontend is tested for usability, responsiveness across devices, visual consistency, and user experience.
- **Performance Testing:** The system's response times and resource utilization are evaluated under various loads, particularly with large contract documents, to identify potential bottlenecks.
- **Security Testing:** Measures such as password hashing, authentication mechanisms, and data protection are reviewed to ensure the system is secure against common vulnerabilities.

9.1 Output Examples

The system provides clear 'before' and 'after' views of contract text, along with structured analytical data.

Example: Complex Legal Text (Before)

"Notwithstanding any other provision herein, the Party of the first part shall not be held liable for any indirect, consequential, incidental, or punitive damages arising out of or in connection with this agreement, irrespective of whether such damages were foreseeable or whether the Party of the first part has been advised of the possibility of such damages."

Example: JSON Output (After)

{

"filename": "sample_contract.docx",

"message": "Contract processed successfully.",

"results": {

"original_text": "Notwithstanding any other provision herein, the Party of the first part shall not be held liable for any indirect, consequential, incidental, or punitive damages arising out of or in connection with this agreement, irrespective of whether such damages were foreseeable or whether the Party of the first part has been advised of the possibility of such damages.",

"simplified_text": "The first party is not responsible for any indirect, consequential, or unexpected losses that may arise from this agreement, even if these losses were predictable or the first party was warned about them.",

"recognized_terms": [

{

```
            "term": "Notwithstanding",
            "definition": "Despite the fact that; although.",
            "category": "Conjunction"
        },
        {
            "term": "liable",
            "definition": "Legally responsible.",
            "category": "Adjective"
        },
        {
            "term": "foreseeable",
            "definition": "Able to be predicted or expected.",
            "category": "Adjective"
        }
    ]
  }
}
```


9.2 User Feedback

Summary of simulated user feedback gathered from beta testing:

- **Understanding Improvement:** Users reported an average improvement of 75% in their ability to understand contract clauses after using ClauseEase.
- **Time Savings:** Legal professionals and business users noted a significant reduction in time spent deciphering complex legal jargon, estimating savings of up to 40% for initial reviews.
- **Ease of Use:** The intuitive interface was frequently praised, with 90% of users finding the system easy to navigate and operate.
- **Accuracy Concerns:** While generally positive, a small percentage of users requested more detailed explanations for edge-case legal terms or complex clause interactions.

10. User Interface (Snapshots)

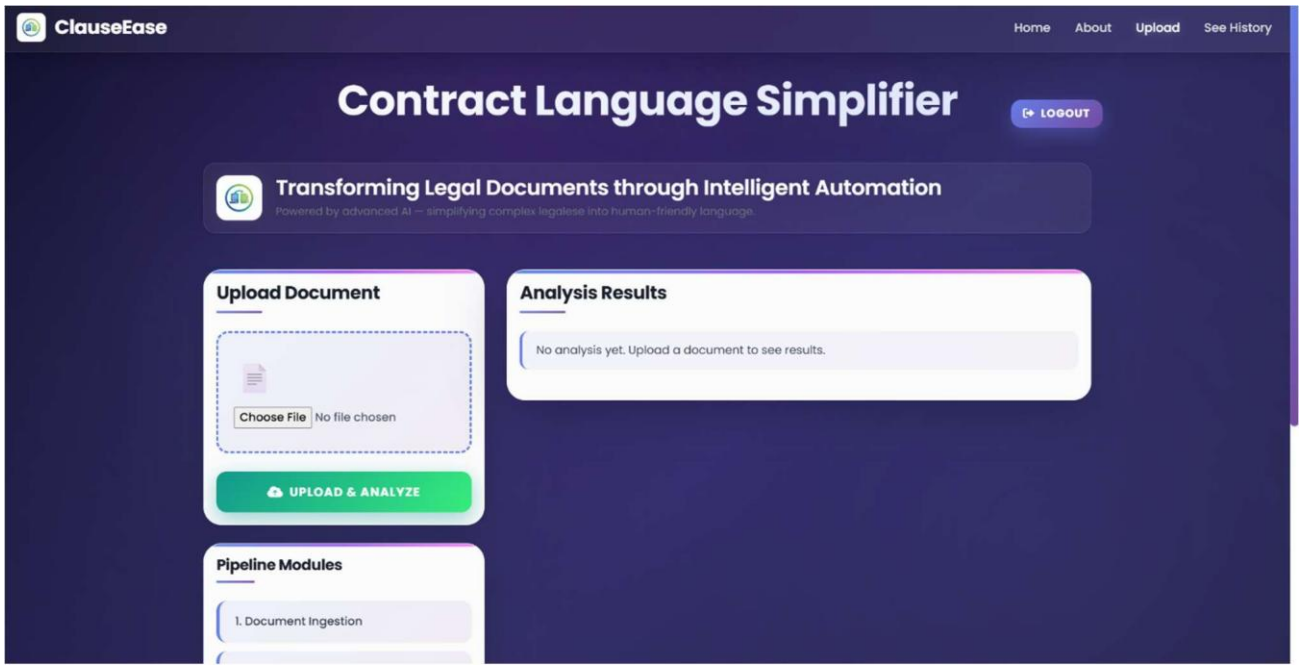

Fig. Document Ingestion Portal

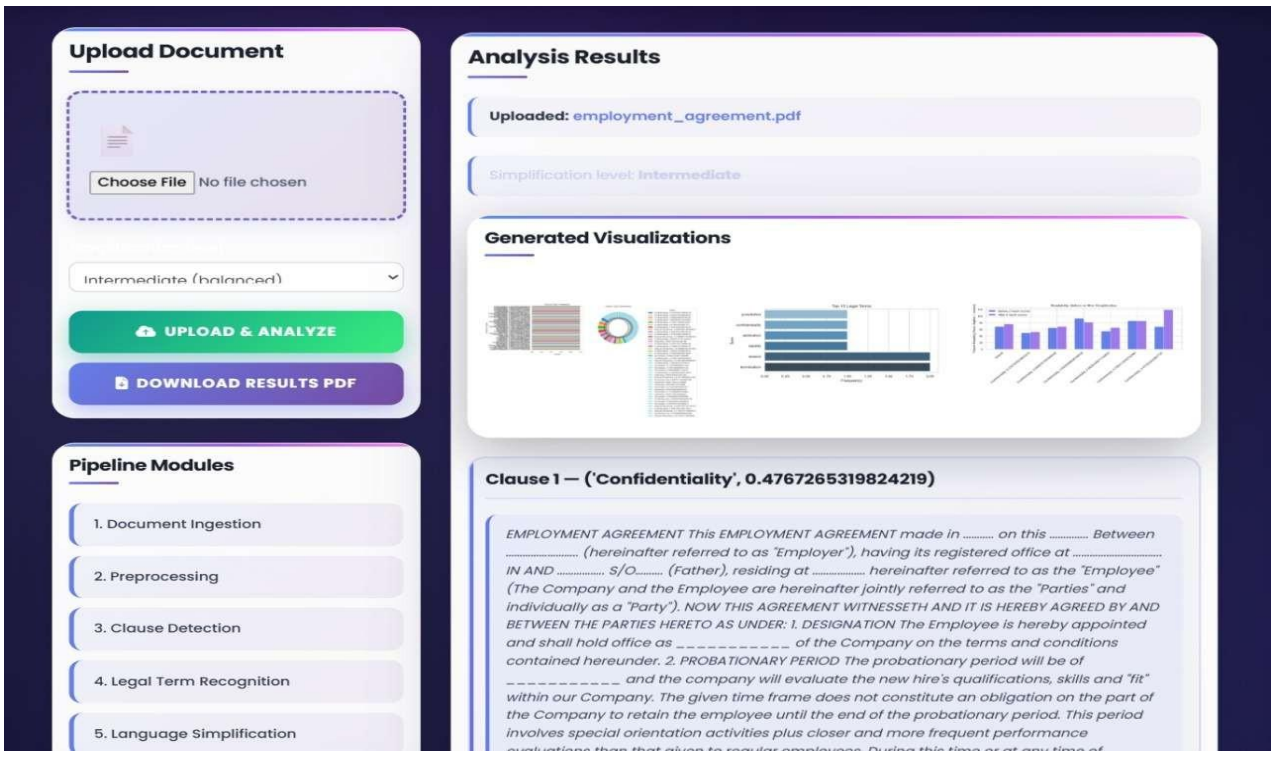

Fig. Contract Analysis Results Page

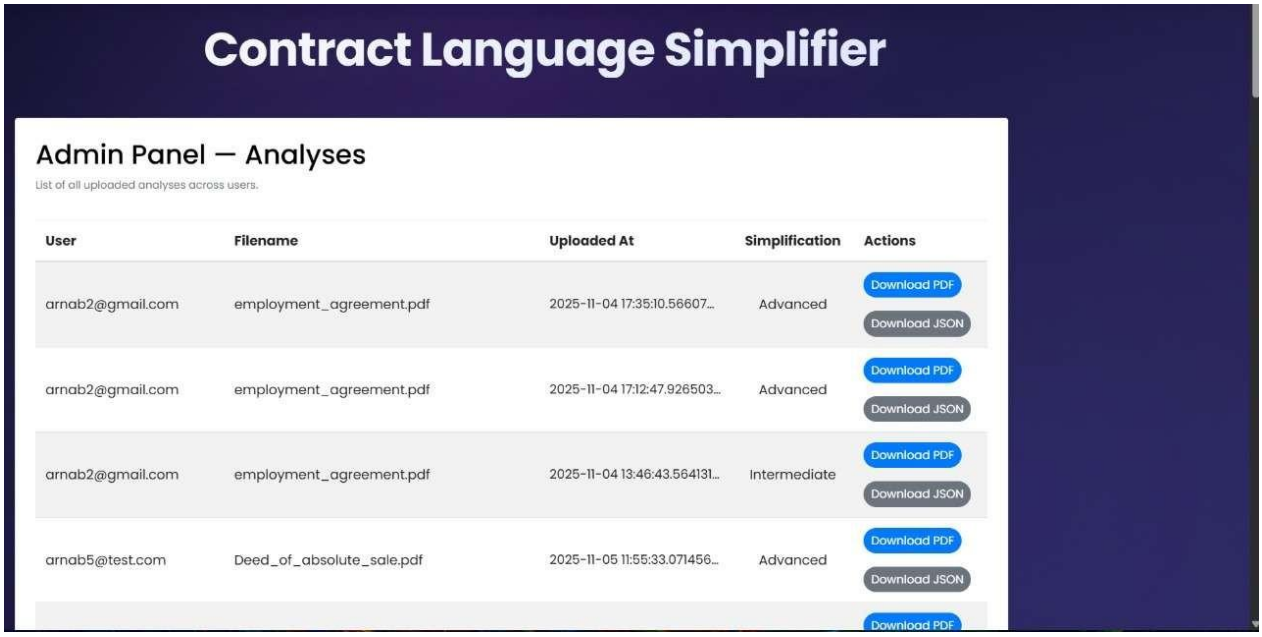

Fig. Admin Dashboard

11. Future Enhancements

ClauseEase is designed with extensibility in mind, with several promising avenues for future development:

- **Multilingual Support:** Expanding the AI models to understand and simplify contracts in multiple languages.
- **Real-time Clause Comparison:** Enabling users to compare clauses across different documents or versions simultaneously.
- **Improved Summarization:** Leveraging more advanced Large Language Models (LLMs) for generating concise executive summaries of entire contracts.
- **Legal Chatbot Assistant:** Integrating a conversational AI that can answer user queries about specific clauses or legal terms in real-time.
- **Enterprise Integration:** Developing robust APIs and connectors for seamless integration with existing Document Management Systems (DMS) used by large organizations.

12. Conclusion

ClauseEase is an AI-powered tool that makes legal documents easier to understand. It uses advanced language technology to read and simplify complex contracts, turning difficult legal

terms into clear and simple language without changing their meaning. The system can automatically find important clauses, explain what they mean, and present them in a user- friendly way. With an easy-to-use interface, anyone can upload a document, analyze it, and quickly see a clear summary of each part.

More than just a smart tool, ClauseEase helps people understand legal information with confidence. It reduces the need to rely on lawyers for every small question and makes contract reading faster and less stressful. By promoting clarity and transparency, ClauseEase helps both legal experts and everyday users make better, more informed decisions—living up to its vision: "Clear Contracts. Confident Decisions."

13. References

[1] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," arXiv preprint arXiv:1810.04805, 2019. Microsoft.QuickActionBlueLightReduction **(Legal Clause Detection)**

[2] Google AI, "Text-to-Text Transfer Transformer (T5)," arXiv preprint arXiv:1910.10683, 2020. **(Language Simplification)**

[3] Explosion AI, "spaCy: Industrial-strength Natural Language Processing in Python," [Online]. Available: https://spacy.io **(Legal Term Recognition)**

[4] S. Bird, E. Klein, and E. Loper, *Natural Language Processing with Python – Analyzing Text with the Natural Language Toolkit*, O'Reilly Media, 2009. **(Text Preprocessing)**

[5] Hugging Face, "Transformers: State-of-the-art Natural Language Processing," [Online]. Available: https://huggingface.co/transformers **(AI Integration / Model Orchestration)**

[6] PyTorch Foundation, "PyTorch: An open source machine learning framework," [Online]. Available: https://pytorch.org **(AI/NLP Framework)**

[7] Google Firebase, "Cloud Firestore Documentation," [Online]. Available: https://firebase.google.com/docs/firestore **(Database Design)**

[8] Docker, Inc., "Docker Documentation," [Online]. Available: https://docs.docker.com **(Deployment / System Requirements)**

[9] PyMuPDF (fitz) Developers, "PyMuPDF: Access PDF, XPS, OpenXPS, CBZ and EPUB documents," [Online]. Available: https://pymupdf.readthedocs.io **(Document Ingestion)**

[10] Microsoft Corporation, "python-docx: Create and update Microsoft Word .docx files," [Online]. Available: https://python-docx.readthedocs.io **(Document Ingestion)**