# AI-Based Contract Review

Developer Manual

By:

Jarrett Hill

Mattea Petrillo

Submitted in partial fulfillment of the requirements for COMP4710 Senior Design to the Department of Computer Science and Software Engineering, Samuel Ginn College of Engineering, Auburn University

Auburn, Alabama
December 4, 2023

# TABLE OF CONTENTS

# 1. Overview

AI-Based-Contract-Review is a program created with the express purpose of providing Auburn's Office of Sponsored Program's (OSP) contract negotiators with a tool that increases their ability of processing contracts. This tool was designed with the idea that an OSP contract negotiator will upload a contract document into the program and the program will be able to determine all instances of problematic contract language within the uploaded contract. The output of the program should be a contract with all instances of problematic language highlighted and flagged, with the program providing/generating suitable alternative language and a justification for changing the initial language.

The main control flow for the program is:

- Take in a given target contract input by the user (OSP contract negotiator)
    - Discern the file's type (pdf or docx) and write the contract text to a .txt file for easier data manipulation
- The program also reads in the FAR Clause and AU Ts&Cs Matrices
    - storing the data within inside of an appropriate data structure (FAR Clauses = numpy array, Ts&Cs = python dict) that enables the content to be easily manipulated
- The program will then scan the entire contract's text and flag matching clauses/problem language.
    - On a matching FAR clause string, the program will collect the matched term, then after the entire contract has been scanned, the program will highlight the text according to the FAR clause acceptance status maintained within the excel matrix.
    - For problem language, the program will read in a paragraph at a time, tokenize each sentence, and then compare the tokenized sentence to a list of common problems that reside within the AU Ts&Cs matrix. If a sentence from the contract matches a sentence within the matrix above a certain adjustable threshold (based on jaccard similarity), the program will throw flag the sentence and write some key details for the flagged language
    - The program will create two helper .txt files to assist in it's problem language flagging.
- The program will also extract key details about the contract's sponsor and provide the extracted details inside of a text window within the GUI.
- The program's main sequence is facilitated by a minimalistic GUI.

# 2. Structure

**Overview** The AI-Based Contract Review system is designed with a function-based architecture. Each file or script in the system is designed to address a specific aspect of the contract review process, meaning each component is focused and narrow in scope.

**Component Integration**

- The user_interface.py script is the main driver function for the whole program.
- The interaction between the scripts is sequential. Once the user provides the input contract and specifies the export destination, the user_interface.py script manages the following tasks:
    1. Identifying the contract's file format and converting it to text for processing (contract_to_txt.py).
    2. Scanning the contract text for FAR clauses (flag_FAR_clauses.py) and problematic language (flag_problem_language.py).
    3. Extracting company details from the contract text (extract_company_details.py).
    4. Outputting the annotated and flagged contract to the specified destination.

**Single-Task Focus**

- The current design of the system is currently suited for handling one contract at a time. The functionality could definitely be expanded to incorporate uploading and scanning multiple documents at a time.
- The system is capable of handling a complete cycle of contract review without user interaction past the initial setup. Once the target contract is input and the "Scan" button is pressed, the system produces the output.

**Internal Data Handling**

- All data processing and management occur internally within the system. After the initial input of the target contract and setting the export path, the system handles all subsequent data manipulations. This internal handling of data simplifies the user experience.

# 3. Program Components Overview

**user_interface.py:** The main driver function for the program.

**flag_FAR_clauses.py:** Scans and highlights the target contract for matching FAR clauses.

**flag_problem_language.py:** Scans and flags the target contract for matching “Common Problems”.

**extract_company_details.py:** Scans contract and collects relevant contract sponsor details.

**contract_to_txt.py:** Helper function designed to assist in flagging FAR Clauses and “Common Problems”.

# 4. Components Details

**user_interface.py:**

- See GUI overview

**flag_FAR_clauses.py:**

- **Purpose**: This script is responsible for annotating contract documents based on Federal Acquisition Regulation (FAR) clauses. It flags clauses in the contract text that match those listed in an external FAR Matrix.
- **Key Functions**:
    - annotate_contract: The main function, which takes inputs for the FAR Matrix (Excel file), the contract file, and a save path for the annotated document. It processes the input files and applies the annotation logic.
    - read_far_matrix: Reads and cleans the FAR Matrix data, storing it in a format suitable for processing. Likely, this function converts the Excel file into a DataFrame using pandas.
    - read_docx: Reads the contract file in DOCX format and extracts the text for analysis.
- **Process Flow**:
    1. **Read FAR Matrix**: The script first reads the FAR Matrix Excel file, cleaning and preparing the data for analysis.
    2. **Extract Clauses**: Clauses and their acceptance statuses are extracted from the FAR Matrix and stored in a pandas datafame.
    3. **Read Contract Text**: The contract text is read from a DOCX file.
    4. **Flagging Clauses**: The script then scans the contract text, flagging any clauses that match those in the FAR Matrix.
    5. **Highlighting Clauses**: Matched clauses are highlighted in the contract document according to their acceptance status as specified in the FAR Matrix.
- **Libraries and Modules**:
    - pandas: Used for reading and processing Excel files.
    - re: Utilized for regular expression operations.
    - docx: For reading and modifying DOCX files.
    - WD_COLOR_INDEX: From docx.enum.text, used for text highlighting in the DOCX document.
- **Input/Output**:
    - **Input**: FAR Matrix (Excel file), Contract file (DOCX).
    - **Output**: Annotated contract file with highlighted FAR clauses.

**flag_problem_language.py:**

- **Purpose**: This script identifies and flags problematic language in contract texts. It uses natural language toolkit (NLTK) to compare sentences from the contract with a list of common problems.
- **Key Functions**:
    - extract_sheet_data: Extracts data from a given Excel sheet, storing the data into a structure python dictionary.

    - NLTK-related functions for tokenizing sentences and comparing them against common problems.
- **Process Flow**:
    1. **Data Extraction**: Extracts data on common contract problems from an Excel sheet.
    2. **Text Analysis**: Analyzes contract text to identify problematic language, using tokenization and comparison algorithms.
    3. **Flagging**: Flags sentences in the contract that match problem language contained in "Common Problems" cells in the AU Ts&Cs Matrix.
- **Libraries and Modules**:
    - nltk: For natural sentence tokenizing.
    - re: For regular expression operations.
    - Numpy, sklearn, and gensim: For numerical operations and machine learning operations for advanced text matching operations. (Note the currently implemented nearest neighbor functionality is less effective than using Jaccard similarity.)

**extract_company_details.py:**

- **Purpose**: Extracts key details about the company involved in a contract, such as the name, location, and type of organization.
- **Key Functions**:
    - extract_company_details: Extracts company details from the contract text using regular expressions.
- **Process Flow**:
    1. **Text Analysis**: Analyzes the contract text to identify relevant company details.
    2. **Data Extraction**: Uses regular expressions to extract details like company name, location, and organization type.
- **Libraries and Modules**:
    - re: For regular expression operations.

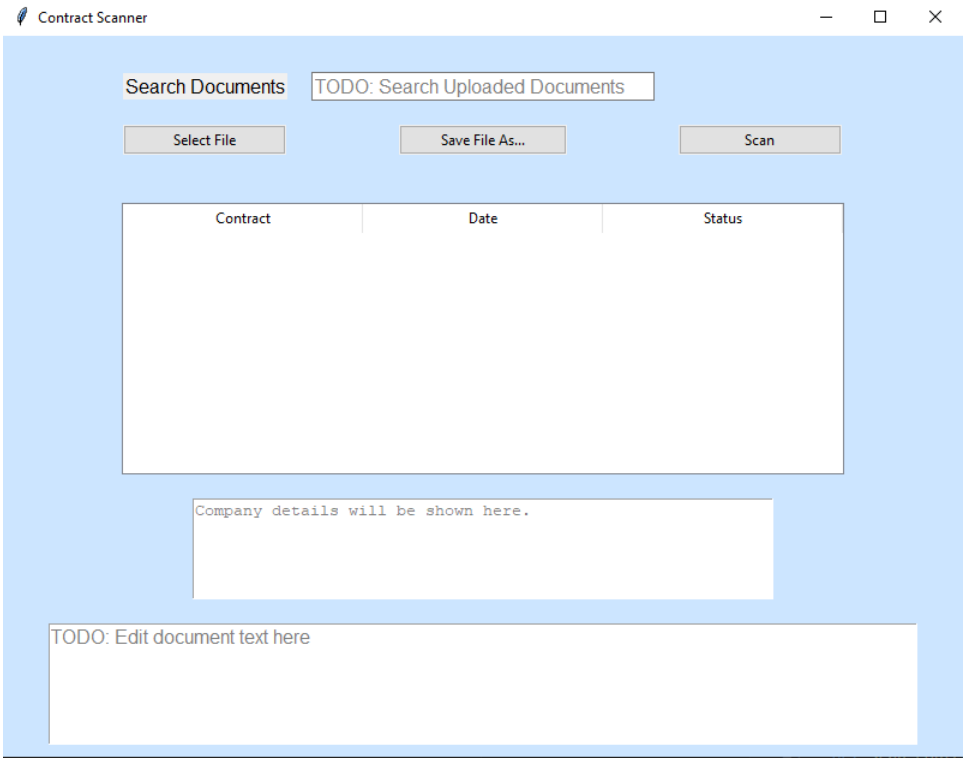
**contract_to_txt.py:**

- **Purpose**: Converts contract documents from DOCX or PDF formats into a text format for easier processing.
- **Key Functions**:
    - _clean_docx: Processes and cleans the content of a DOCX file.
    - _clean_pdf: Processes and cleans the content of a PDF file.
    - convert_to_txt: Driver function that should be called by other files if txt conversion is needed.
- **Process Flow**:
    1. **File Reading**: Reads the contract file in either DOCX or PDF format.
    2. **Text Extraction**: Extracts the text content from the file.
    3. **Cleaning**: Cleans and prepares the extracted text for further processing.

- **Libraries and Modules**:
    - docx2txt and PyPDF2: For handling DOCX and PDF files, respectively.
    - docx: For operations related to DOCX files.

# 5. GUI Overview

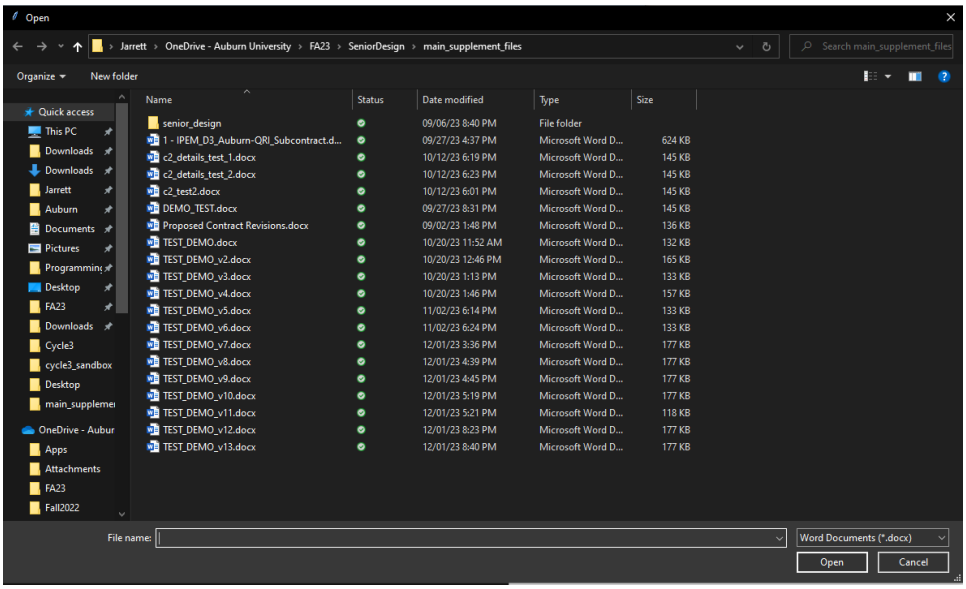
**The opening screen:**



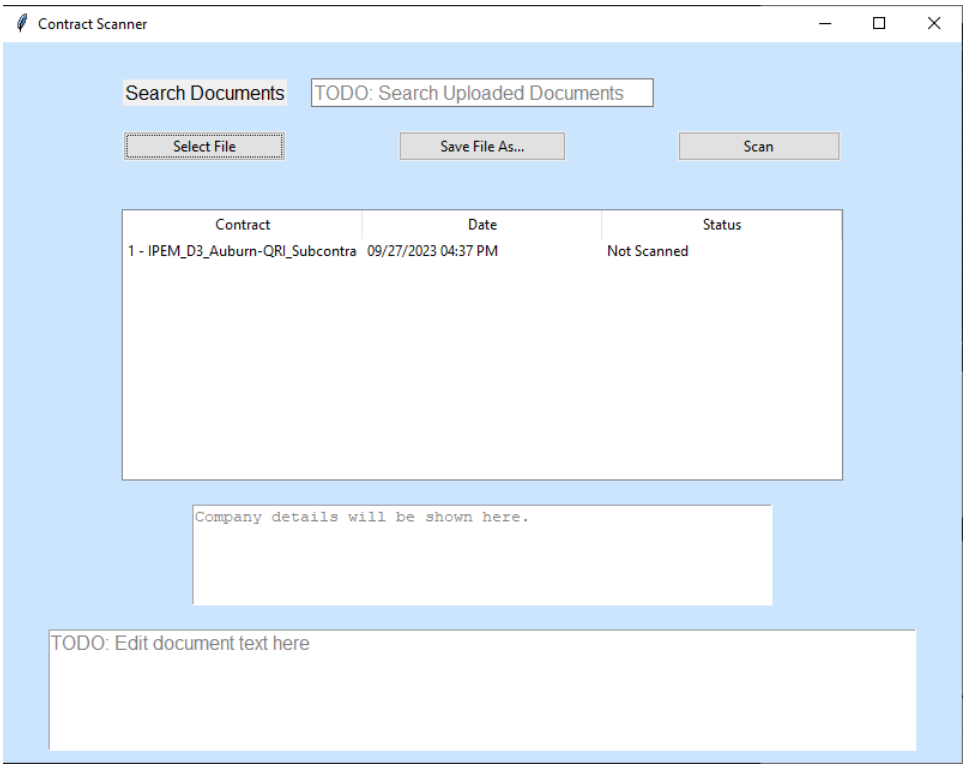

- The user is meant to use the current program after launch with the flow of

“Select File” -> “Save File As...”-> “Scan”

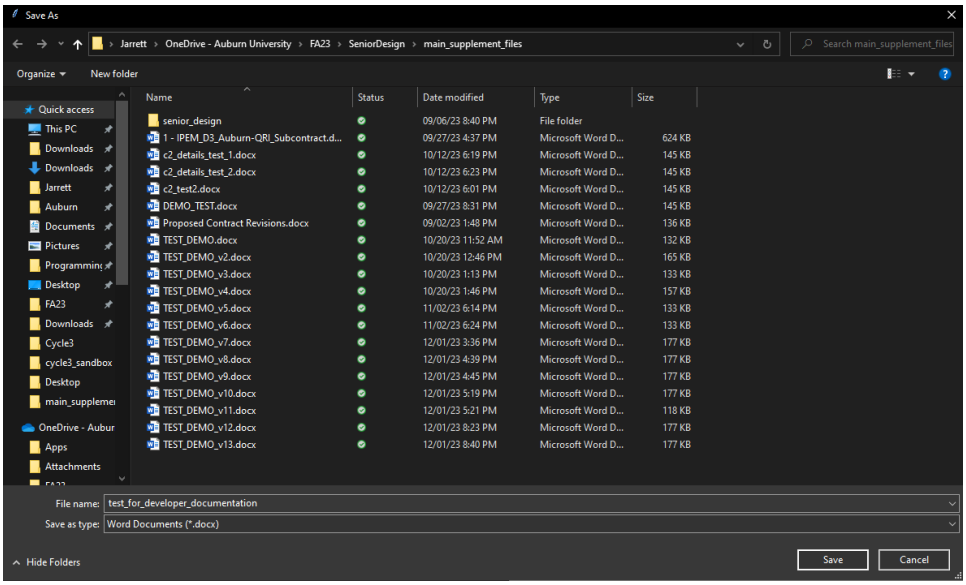
**Select File:**



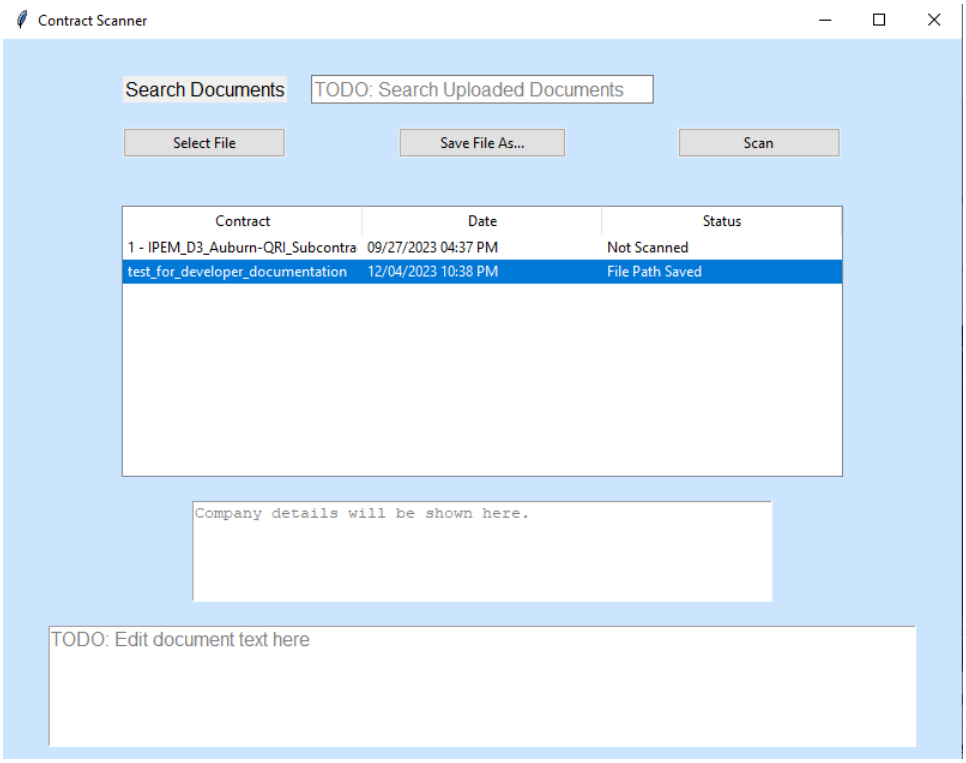

- Will open a native Windows file browser prompting the user for the target contract. The file browser has an option for DOCX or PDF files
- After selecting the target contract the screen updates the file preview window




**Save File As...:**

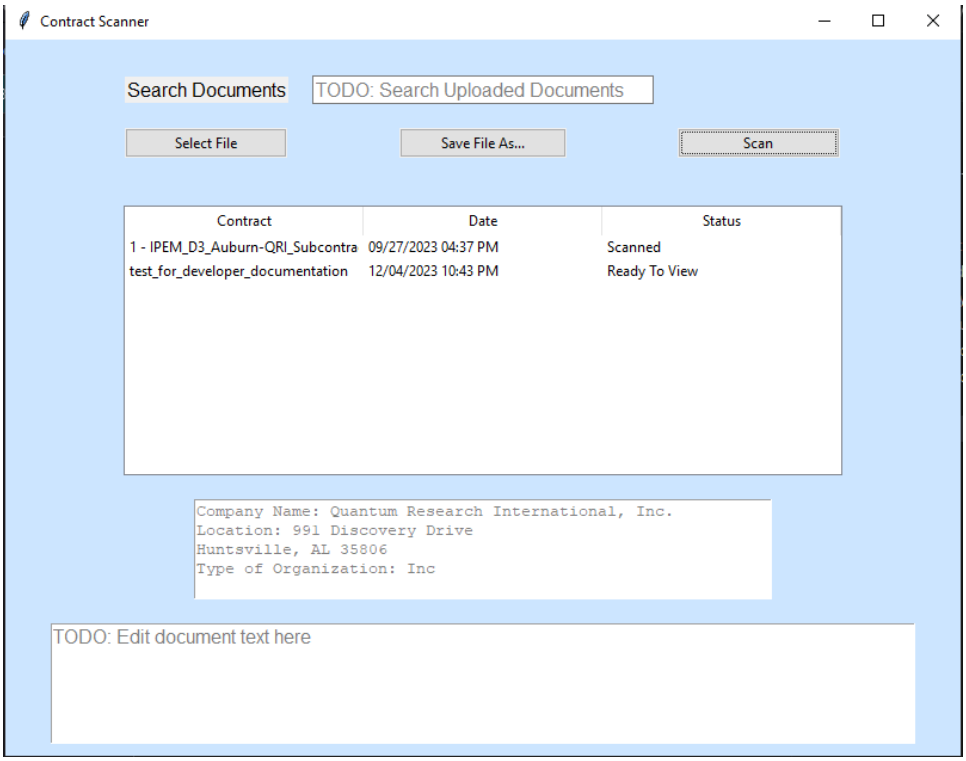



- Again, opens a native Windows file browser prompting user to type in a file save name and select their desired save destination. Defaults to where the target contract was opened from.
- After specifying save path the screen updates the file preview window




**Scan:**

- After the save destination is specified and the “Scan” button is clicked, the file preview window will update accordingly




- Alongside the company details viewer window being populated with the relevant company details.

# 6. Program Installation and Configuration

**Overview:**

- At the time of writing this, the program was 100% written in VSCode in Python version 3.11.6. There is no necessary VSCode configuration utilized for this project.
- The program should be able to run in any IDE that supports Python 3.11.6
- The only necessary supplement files beyond what are uploaded to the GitHub repo are the FAR clause matrix, Ts&Cs Matrix, and any target contract file.
- The helper text files 'contract_to_txt.txt' and 'flagged_contract_to_txt.txt' are generated on program execution.

**External Libraries:**

1. **docx2txt**
    - **General Purpose**: Used for extracting text from Microsoft Word .docx files.
    - **Installation**: pip install docx2txt
    - **Documentation**: docx2txt on PyPI
2. **PyPDF2**
    - **General Purpose**: A library for handling PDF files, including reading text from PDFs.
    - **Installation**: pip install PyPDF2
    - **Documentation**: PyPDF2 on PyPI

3. **python-docx (docx)**
    - **General Purpose**: Used to create, modify, and extract information from Word documents.
    - **Installation**: pip install python-docx
    - **Documentation**: python-docx on Read the Docs
4. **pandas**
    - **General Purpose**: Provides data structures and data analysis tools.
    - **Installation**: pip install pandas
    - **Documentation**: pandas on PyPI
5. **openpyxl**
    - **General Purpose**: A library to read/write Excel 2010 xlsx/xlsm files.
    - **Installation**: pip install openpyxl
    - **Documentation**: openpyxl on Read the Docs
6. **NLTK (Natural Language Toolkit)**
    - **General Purpose**: A leading platform for building Python programs to work with human language data.
    - **Installation**: pip install nltk
    - **Documentation**: NLTK on PyPI
7. **gensim**
    - **General Purpose**: Used for unsupervised topic modeling and natural language processing.
    - **Installation**: pip install gensim
    - **Documentation**: gensim on PyPI
8. **NumPy**
    - **General Purpose**: A fundamental package for scientific computing with Python.
    - **Installation**: pip install numpy
    - **Documentation**: NumPy on PyPI
9. **scikit-learn (sklearn)**
    - **General Purpose**: Simple and efficient tools for predictive data analysis.
    - **Installation**: pip install scikit-learn
    - **Documentation**: scikit-learn on PyPI

Side note on installing these external libraries: About half of these did not install correctly with the `pip install {library}` command and instead I had to use `pip3 install {library}`. This may just be a personal issue, but I figured I would include this if I could save a headache.

**Where to find the code/files:**
**https://github.com/J-Hill22/main/tree/Cycle3**
**(This does not contain the necessary FAR Clause and Ts&Cs Matrix at the time of writing this. I plan to upload those files into the repo in the near future, but if I don't you can get them from Mr. May at the OSP or I can send them to you jwh0100@auburn.edu)**

# 7. Assumptions and Dependencies

**Assumptions:**

- The user will execute this program on a Windows machine.
- The user only inputs and scans a single target contract at a time.
- The target contracts will always be of type DOCX or PDF.
- The output contract does not need to specifically be the input format, as the program defaults the output contract to DOCX to utilize the FAR highlighting feature.

**Dependencies:**

- **docx2txt**
- **PyPDF2**
- **python-docx (docx)**
- **pandas**
- **openpyxl**
- **NLTK (Natural Language Toolkit)**
- **gensim**
- **NumPy**
- **scikit-learn (sklearn)**

# 8. Current Limitations / Future Work

**1. Accuracy and Precision in Identifying Problem Language**

- **Current Limitation**: The program's current text-matching algorithm, based on Jaccard similarity, does not achieve a high level of accuracy or precision in identifying problem language within contracts. This limitation is the most important factor to consider when moving forward with the project. Because of the nature of the work conducted at the OSP, our program we are providing for them needs to inspire confidence that ALL potential problematic language is being flagged.
- **Future Work**: Implementing a more sophisticated natural language processing (NLP) algorithm, potentially integrating machine learning models, could significantly enhance the program's ability to accurately and precisely flag problem language. Research into advanced text analysis methods should be prioritized to improve this functionality.

**2. Automated Generation of Alternative Language**

- **Current Limitation**: The program does not currently offer a feature to generate and suggest alternative language for flagged problematic sections in contracts. This was just a feature that I never got around to addressing in my development cycle.

- **Future Work**: Developing a feature to automatically suggest approved alternative language when problem language is detected. This would require implementing advanced NLP techniques and possibly training a machine learning model on a dataset of contract language modifications.

**3. Feedback System for Generated Responses**

- **Sponsor's Requirement**: Alongside generating alternative language, the sponsor has expressed interest in a native feedback system to judge (like/dislike button) the quality of the program's generated alternatives.
- **Future Work**: Implementing a user feedback mechanism within the program, allowing users to rate the suggested alternative language. This feedback can be used to continuously improve the algorithm behind the generation of alternative language.

**4. PDF File Handling**

- **Current Limitation**: The program currently struggles with processing PDF files, often resulting in output documents with excessive and randomly dispersed whitespace.
- **Future Work**: Enhancing the PDF processing capabilities to handle various PDF types more effectively. This might involve refining the text extraction process and implementing better algorithms for dealing with PDF formatting nuances.

**5. Output Format Consistency**

- **Current Limitation**: The program does not currently output scanned, flagged, and highlighted documents in the same format as the input. This inconsistency can be problematic for users expecting a seamless transition between input and output formats. Mr. May and I had repeated discussions about this detail. While he said he could live without it, I could tell this feature was pretty important to the end result.
- **Future Work**: Developing functionality to maintain the format consistency between input and output files. This will likely require advanced handling of document formatting and styles, especially for diverse file formats like DOCX and PDF.