Option 1: Data Extraction and Classification

Objective:

Develop a pipeline to automatically **extract** specific **property values** from tender documents using a pre-trained language model (e.g., BERT, LLaMA, GPT). This task aims to demonstrate your ability to handle unstructured data, preprocess it, and use modern NLP techniques for information extraction.

Example of Property-Value Pairs:

Property	**Value**
publication_date	12.03.2019
external_id	00987456-2345
cpv_code	12340000; 56789100

Input:

Two tender documents will be provided (see attached files) in HTML format. The documents should be stored in a database (e.g., SQLite) with two columns:

- **ID**: A unique identifier for each tender document
- **HTML Text**: The **raw** HTML content of the tender document.

Output:

The pipeline should produce a structured output in **JSON** format. Each document will be represented by a JSON object that contains key-value pairs. A key may have multiple values, for example:

{" publication_date ": ["12.03.2019"],

"external_id": ["00987456-2345"],

"cpv_code": ["12340000", "56789100"]}

The output should include the following properties for each document:

- cpv_code: The CPV code(s) associated with the tender.
- publication_date: The date when the tender was published.
- title: The title of the tender.
- journal_number: The journal number of the tender (e.g., "*174/2024*"), but ensure that this is extracted dynamically, not hardcoded.
- external_id: The Provider ID, which is a numeric value and the year (e.g., " *00534910-2024*"), but this pattern should be extracted dynamically, not hardcoded.

Pipeline:

The solution should be modular, allowing it to **extract data by providing the document ID**. Based on the ID, the pipeline should retrieve the corresponding HTML text from the SQLite database, clean it to create raw text, and then apply an NLP model to extract the relevant properties.

Tasks:

1. **Extract HTML Text**

- Given a tender ID, retrieve the HTML text from the database.
- Clean and preprocess the HTML content to convert it into raw text (e.g., remove HTML tags).

2. **Use a Pre-trained Language Model (No Regex)**

- Utilize a pre-trained NLP model (e.g., BERT, LLaMA, GPT) to extract the properties from the cleaned text.
- Note: Do not use regular expressions (regex) for extracting these properties. The extraction should rely entirely on the language model's capabilities.

3. **Store Results in JSON Format**

- For each document, store the extracted key-value pairs in a JSON file.
- Ensure that the values are correctly represented as lists (even if there is only one value per key).

4. **Repository**

- Upload the complete solution to a GitHub repository.
- The repository should contain all the necessary files

Evaluation Metrics:

1. **Creativity and Problem-Solving**

- Creativity in how you approach the extraction and classification of tender data.
- The ability to design a pipeline that is flexible, scalable, and maintainable.

2. **Code Quality**

- Clean, modular, reusable, and well-documented code.
- Adherence to Python best practices.
- Proper repository.

3. **Use of NLP Model**

- Effective use of a pre-trained language model to extract the required information.
- Ensure that the model is used appropriately for this task.
- Prompt engineering utilization.

Additional Notes:

- There is **no need to provide a** fully functional **solution or** an **accurate output**. However, the approach and the **idea for solving the problem** should be clearly stated/**explained**.
- If you choose to use GPT, here is an API key that you can use. Be aware of the cost associated with using the API. If you exceed the cost limit, halt the project there.
- **Use "gpt-4o-mini" ONLY, otherwise the key will be blocked.**
- KEY: [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]
- Key is just available till 12.12.2024
- Add to the GitHub repo: SebastianArchila

Attached Files (HTML input data):

- 01_tender_doc.txt
- 02_tender_doc.txt

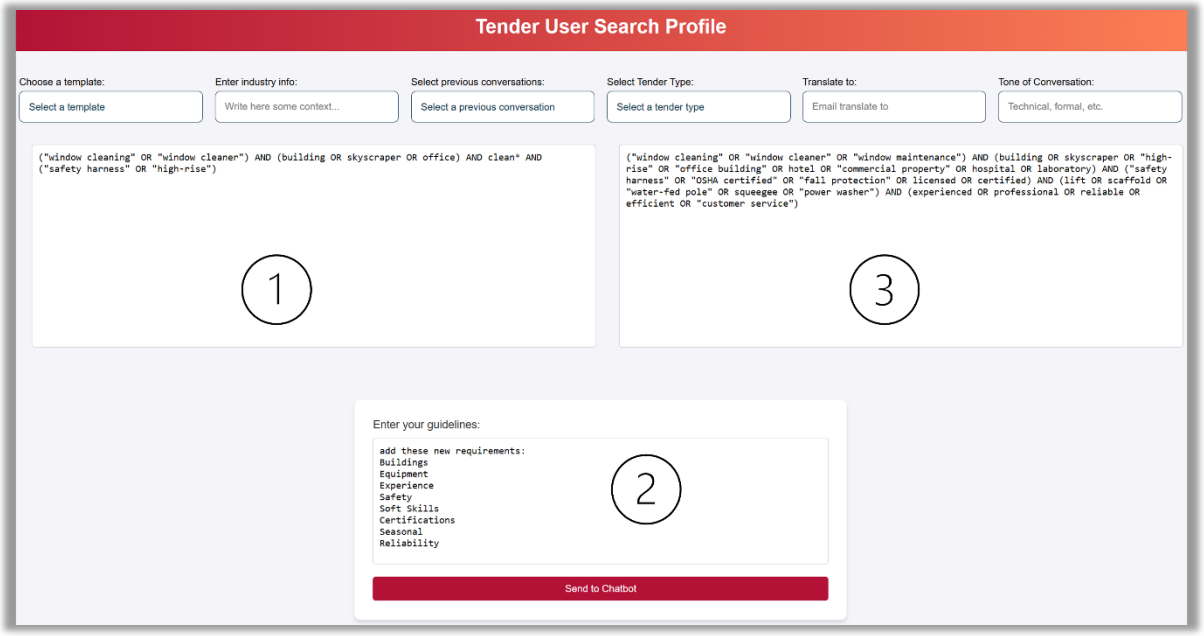
If you have any questions, please feel free to contact **sebastian.archila@tender-service.com**

Option 2: Frontend App for Improving Boolean Queries

Objective:

Create a frontend app with a “chatbot” interface that helps users craft and refine Boolean queries efficiently.

Example of frontend:



Goal:

The app should help users create robust Boolean queries by engaging in a focused **dialogue**.

Functional Requirements:

1. **Frontend**

- Create a simple web interface (e.g., using Flask, NodeJS).
- The chatbot interface should include the following functionalities:
    - **Boolean Query Display (Point 1 in the example image)**: Display any Boolean query, which the user can modify or improve.
    - **User Instruction Input (Point 2 in the example image)**: Allow the user to input new instructions or modifications to the query.
    - **LLM Query Generation (Point 3 in the example image)**: The LLM (Large Language Model) should process the user's input and generate an updated Boolean query based on the new instructions, incorporating any modifications or additions.
    - **Conversation Memory**: Maintain a conversation history, where the system remembers previous interactions. For example, if the user adds more instructions after the initial query generation, the LLM should recall the previous query and adjust it accordingly. This ensures that the conversation is dynamic and that the system can build upon prior inputs to refine the query over time.
- **It is not necessary to display the entire conversation history in the front end**

2. **Core Functionality**

- Implement a backend chatbot logic (e.g., using GPT, LLaMA, Mistral) that:
    - Accepts user inputs about desired or excluded keywords.
    - Suggests terms or structures for the Boolean query.
    - Retains memory of the query during the session to provide coherent suggestions.

3. **Repository**

- Upload the complete solution to a GitHub repository.
- The repository should contain all the necessary files

Evaluation Metrics:

1. **Creativity and Problem-Solving**

- Creativity in how you approach this task.
- The ability to design an app (frontend and backend) that is flexible, scalable, and maintainable.

2. **Code Quality**

- Clean, modular, reusable, and well-documented code.
- Adherence to Python/JS best practices.
- Proper repository.

3. **Prompt engineering**

- Effectiveness in designing prompts for the LLM to generate meaningful, accurate, and relevant queries.
- Ability to refine the prompts so that the model responds appropriately to user instructions and maintains consistency over multiple interactions.

Additional Notes:

- There is **no need to provide a** fully functional **solution or** an **accurate output**. However, the approach and the **idea for solving the problem** should be clearly stated/**explained**.
- If you choose to use GPT, here is an API key that you can use. Be aware of the cost associated with using the API. If you exceed the cost limit, halt the project there.
- **Use "gpt-4o-mini" ONLY, otherwise the key will be blocked.**
- KEY: [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]
- Key is just available till 12.12.2024
- Add to the GitHub repo: SebastianArchila

If you have any questions, please feel free to contact **sebastian.archila@tender-service.com**