Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica

Smart Extraction of Structured Data from Unstructured Documents

Relatore:
Prof.
Federico Montori

Presentata da:
Elia Friberg

Sessione 2
Anno Accademico 2024

Abstract

This thesis presents DataDig, a mobile application designed to address the growing need for efficient and accurate data extraction from documents and images.
DataDig uses a combination of Optical Character Recognition (OCR) and Large Language Models (LLMs) to automate the extraction process, leaving the users the ability to define fully customizable templates that retrieve specific information without the need for model training or example documents.

The app's key features include support for various data types, dynamic table extraction, context-aware interpretation and direct integration.
Through testing and user feedback, DataDig demonstrates its effectiveness in accurately extracting information from diverse document types.
The mobile-first design, coupled with the number of options available for both the process and output format, as well as the added cost transparency, makes it a useful tool for individuals, small businesses, and professionals looking to streamline document processing and data management workflows.

This research contributes to the field of data extraction by providing a practical solution that utilizes the power of LLMs to automate this previously time-consuming and error-prone process.

The thesis also explores the broader context of today's AI-powered data extraction, analyzing ethical implications and the current market landscape, comparing DataDig with its main competitors, and discussing potential future developments.

Sommario (ITA)

Questa tesi presenta DataDig, un'applicazione mobile progettata per affrontare la crescente necessità di estrarre dati efficientemente ed accuratamente da documenti e immagini.
DataDig utilizza una combinazione di riconoscimento ottico dei caratteri (OCR) e Intelligenza Articiale (LLM) per automatizzare il processo di estrazione, consentendo agli utenti di definire modelli completamente personalizzabili che recuperano informazioni specifiche in base alle loro esigenze, senza la necessità di addestrare un AI o fornire documenti di esempio.

Le caratteristiche principali dell'app includono: Supporto per vari tipi di dati, l'estrazione dinamica di tabelle, l'interpretazione basata sul contesto e l'integrazione diretta con le aziende.
Attraverso test e feedback degli utenti, DataDig dimostra la sua efficacia nell'estrarre accuratamente le informazioni da diversi tipi di documenti.
Il design "mobile-first", insieme alle numerose opzioni disponibili sia per il processo di estrazione che per il formato di output, nonché la trasparenza dei costi,fanno di DataDig uno strumento utile per individui, piccole imprese e professionisti che desiderano semplificare i flussi di lavoro di elaborazione dei documenti e di gestione dei dati.

Questa ricerca contribuisce al campo dell'estrazione dei dati fornendo una soluzione pratica, che utilizza la potenza degli LLM per automatizzare questo processo, precedentemente dispendioso in termini di tempo e soggetto a errori.

La tesi esplora anche il contesto più ampio dell'estrazione dei dati basata sull'intelligenza artificiale, analizzando le implicazioni etiche e il panorama attuale del mercato, confrontando DataDig con i suoi principali concorrenti e discutendo i potenziali sviluppi futuri.

Introduction

In an increasingly data-driven world, the ability to extract data from physical or unstructured sources is crucial for accessing the whole potential of your data. And organizing this data in an efficient and usable way is useful and important for both businesses and individuals alike.
To achieve this, businesses today face the challenge of not only transitioning to fully autonomous database systems but also integrating vast amounts of legacy data.
This legacy is often trapped within custom-made or unstructured documents, leaving the only source of truth of the data understanding the context and the elements and text of the document.

The advent of AI-powered parsing techniques relying on neural networks and Large Language Models (LLMs) has revolutionized data extraction.
These technologies can now mimic and substitute that long and boring work that an unlucky employee or intern had to do, allowing for rapid and accurate data retrieval at a fraction of the cost and time, while still significantly lowering error rates compared to human alternatives, as we see in [4] .

But, even if less profitable, the benefits of data extraction extend far beyond large enterprises, where data extraction is most present [12].
Professionals, small businesses, and individuals can all leverage this technology to streamline their work and make life easier.
From organizing personal documents like prescriptions and invoices to extracting contact information from business cards, the applications are diverse and impactful.

My interest in this field stemmed from my experience during an internship at Prometeia™, where I participated in the development of a Python-based tool for extracting data from semi-flexible templates.
Having achieved promising results in terms of extraction accuracy and recognizing the potential of this technology, I was motivated to develop a customizable mobile application to make it available and free to everyone.
The design and development of DataDig was also influenced by the concepts and ideas presented in [3], an essential reading for those interested in the application of Large Language Models for text analysis.

Based on these experiences and research findings, I want to propose DataDig, a free, customizable, and user-friendly mobile app that addresses the challenges of document digitalization and data extraction.

DataDig allows users to extract key-value pairs from documents and images, offering precise control over the extraction process, and most importantly, flexibility over the document.
Users can define custom templates, specifying data types, default values, and extraction rules based on both textual content and contextual understanding.
The app also supports fixed, semi and fully dynamic table extraction, allowing users to filter and select specific rows, columns and cells.

Crucially, DataDig does not require training or example documents, nor does it rely on fixed document templates.
Its AI-powered engine utilizes the textual content and context of the documents themselves, letting it adapt to a wide range of document types and layouts.
Unlike other solutions that require example documents or pre-defined templates, DataDig allows users to define extraction rules based solely on the desired data and the document's context.

Furthermore, DataDig is a free application that utilizes the user's own API keys for OpenAI and Azure services, ensuring transparency and cost control.
DataDig also does not collect any of the user's information, as the keys and sensitive extractions data are kept encrypted on the device and never shared.

Contents

List of Figures

List of Tables

Chapter 1

State of the Art

1.1 Market and Placement

Today, the extraction and management of information is more than indispensable.

With the world generating an estimated 2.5 quintillion bytes of data daily, much of which is stored in unstructured formats like PDFs, the need for automated solutions is evident.

Traditional manual document processing is not only costly, but also vulnerable, with over 70% of businesses estimated to fail within three weeks if they lost their paper-based records in a fire or flood [14].
This highlights the critical need for companies for reliable and automated data extraction tools to facilitate and mitigate the risks of manual data management.

In response to this need, the field of automated data extraction has evolved significantly, from rigid OCR machines and position-based analyzers, to now, when the rise of machine learning and Large Language Models (LLMs) has opened up new possibilities in this field.

Several AI-based data extraction services have emerged in recent years, mostly catering to large enterprises.

This focus is understandable, as high-volume extractions and workflow integration are essential for businesses, and with consume based costs, the profit is nearly all where the big companies are.

While the presence of major players like Google and Amazon might seem daunting, DataDig's unique features and target audience allow it to carve out a distinct niche.

Existing solutions often involve complex setups, require model training, and come with subscription-based pricing models, typically in the hundreds of dollars per month.
This creates a barrier for individuals, small businesses, and professionals who need a more accessible and flexible solution.

DataDig addresses this gap by offering a free, user-friendly mobile app that instead uses the user's own API keys for cost transparency and eliminates the need for long training and example documents.
Its custom template system allows users to define extraction rules based on both textual content and contextual understanding, adapting to various document types and variabilities.

Additionally, DataDig's mobile-first approach allows for on-the-go document capture and data extraction, allowing the user to use the camera to directly scan the documents.

1.1.1 Comparison with Competitors

Let's explore some big key players in the data extraction market and how DataDig compares: (data was taken by direct exploration of the platforms and from [14])

- **Google Document AI:** Google's solution offers a wide range of pre-trained parsers and integrates with the Google Cloud ecosystem.
  However, it is both less flexible for custom extractions, needing to provide document examples, and pretty complex to setup, as it is very developer driven.
- **Amazon Textract:** This AWS service gives access to advanced features for specific industries and experts, but is much less accessible for non-technical users.
- **Nanonets:** This platform uses continuous learning and advanced table extraction, but requires training for custom models and has a high subscription cost.
- **Docparser:** This cloud-based solution offers automated workflows and supports various document formats.
  However, it has limitations in its customizability and only utilizes zonal OCR.

1.1.2 DataDig's Differentiators:

DataDig stands out due to:

- **No Training Required:** Custom templates can be created without the need for example documents.
- **Mobile-Native:** Provides a seamless mobile experience with on-device processing and camera integration
- **Default Values:** Handles missing fields by allowing users to specify default values.
- **Context-driven extraction**: The extractor understands the context of your input and of the document, and can better recognize and even infer what you ask from the text.
- **Cost Transparency:** Users control costs by using their own API keys.
- **Data Privacy:** Sensitive data and keys are stored locally on the device.

Table 1.1: Table showing a direct comparison with the main competitors

Feature	Google Document AI	Amazon Textract	Nanonets	Docparser	DataDig
Pricing Model	Pay-per-use	Pay-per-use	Freemium (starting at **$500**/month)	Freemium (starting at **$25**/month)	Free, user pays for API usage
Flexibility	Pre-trained parsers, limited trainable custom models	Advanced features, limited custom models	Continuous learning, custom models with training	Customizable position-based parsing rules	High flexibility with fully customizable templates
Contextual Extraction	Limited	Limited	Limited	not available	Supports context-based extraction
Default Values	Not available	Not available	Not available	Not available	Available
Ease of Use	Requires Google Cloud familiarity	Requires technical skills and AWS familiarity	Simple Web-Based UI	Complex Web UI	Intuitive UI, requires key setup
Supported Inputs	Wide range of formats	Primarily PDFs	PDFs, images, and other formats	PDFs, Word documents, and images	Direct support for images and PDFs
Data Privacy	Data processed in Google Cloud	Data processed in Amazon Cloud	Data processed in Nanonets Cloud	Data processed in Docparser Cloud	Kept on-device, keys are encrypted
Target Audience	Enterprises	Enterprises	Enterprises	Businesses of all sizes	Individuals, small businesses, professionals
Model Customization	Requires manual annotation and model training	Requires model training	Requires example documents or zero-shot learning	Customizable parsing rules, potential training	Custom templates based on descriptions, no examples needed

Chapter 2

Overview of the App

2.1 Main Flow and Use Cases

Here follows an explanation of what a user can expect when using the app.
A rough flow is also provided to understand the use case.

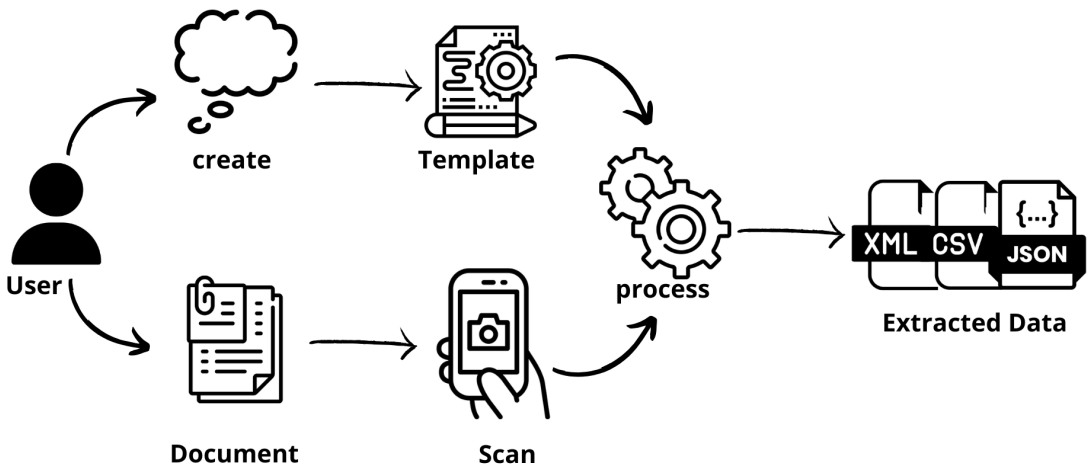


Figure 2.1: Flow of the app

First Launch and Setup:

- The user is guided through an initial tutorial and explanation of the app and its components.
- In the settings, the user enters the necessary API keys
  (OpenAI is mandatory, Azure is optional for tables).
  The app provides links and instructions on how to obtain these keys.

Template Creation/Selection:

- The user can choose from predefined (customizable) templates or create new ones.
- Templates can be customized by defining **extraction fields** (title, description, data type, mandatory status, defaults, etc.) and **tables** (titles, descriptions, keywords, structure).
  It's possible to extract entire tables or only specific parts of them.

Document Acquisition:

The user can take a photo, select an image from the gallery, or upload a PDF.

Option Selection:

1. **Template selection** (if not already selected).
2. **Choose extraction options**:
   Language (auto-detect or manual), Output format (JSON, CSV, XML, TXT), OpenAI model (GPT-3.5-turbo,GPT-4, Smart mix), enable Azure OCR (which offers better results than the basic one).

Extraction:

- The extraction is performed in the background so that the user can exit the app while it is being performed.
- A notification signals the completion of the extraction when the app is in the background.

Data Access:

- **Storage Screen:**
  Displays all previous extractions in detail and gives the user the ability to access, filter, sort and delete them.
- **Extraction Screen:**
  Lets you look at, copy and modify the extracted fields and tables.
  It also lets you add additional images (for logos or company crests) and informs you about costs and eventual errors.
- **Extraction File:**
  Contains the extracted data and the images used, it can be downloaded, shared, or opened with other apps through implicit intents to allow for direct integration with company workflows.

2.2 Screens

2.2.1 Template Screen

The main "Template" screen serves as the central hub for managing document templates within the app.
As illustrated in the screenshot, it presents a simple interface with the following elements:

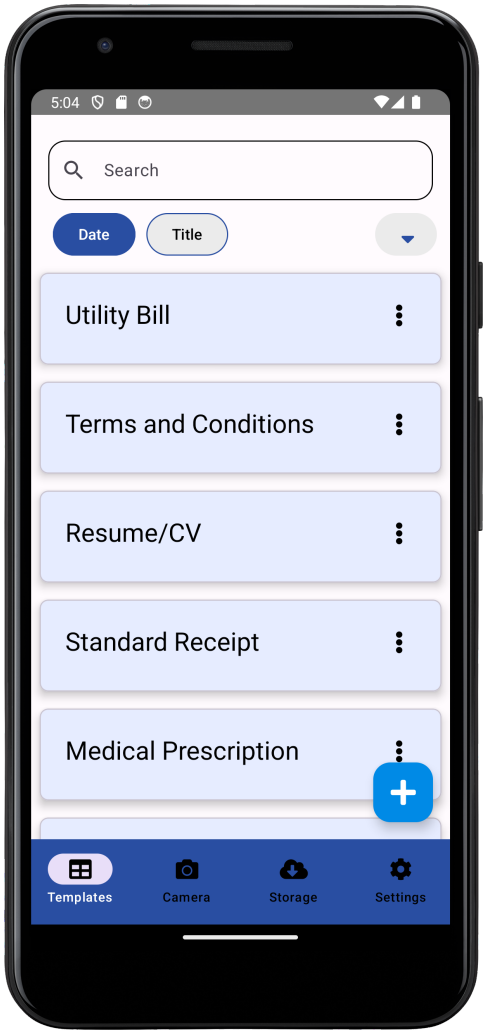


Figure 2.2: Template Screen

- **Search Bar:**
  Located at the top, allows to quickly search for a specific template
- **Sorting Options:**
  Sort the displayed templates by "Date" or "Title" in either ascending or descending order.
- **Template List:**
  List of available templates, both pre-made and user-created. Each template is represented by a card containing its title and a three-dot dropdown.
- **Dropdown Menu (Three-dot icon):**
  The menu offers options to "Edit", "Use", or "Delete" the selected template.
- **"Add Template" Button:**
  The plus button in the bottom right corner enables users to create a new customizable template.

2.2.2 Field Editing

This screen allows user to modify or add data extraction fields.

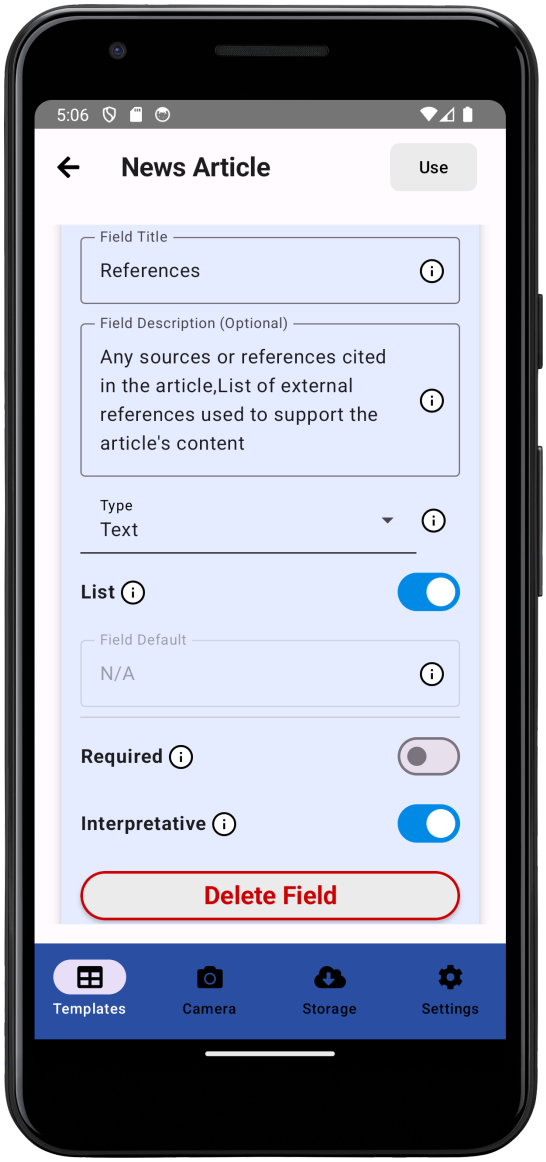


Figure 2.3: Field Editing

- **Title:** The field name and main searching criterion.
- **Description:** Most used for contextual help, optional but recommended if the title is insufficient.
- **Type:**
  The data type the user wants returned.
  The available data types are:
  Text, Number, Date, Float, Boolean.
- **List:** If multiple values are possible.
- **Default:**
  The value to be returned if the field is not found.
- **Required/Optional:**
  Whether the value must exist within the document.
- **Interpretative:**
  Whether the field should be extracted directly from the document's text or inferred from its overall context.
  Enabling this allows for the extraction of information that is not explicitly stated, but can be derived through interpretation, such as summaries, sentiment analysis, or meta-information

2.2.3 Table Editing

Settings to modify the tables that need to be extracted.

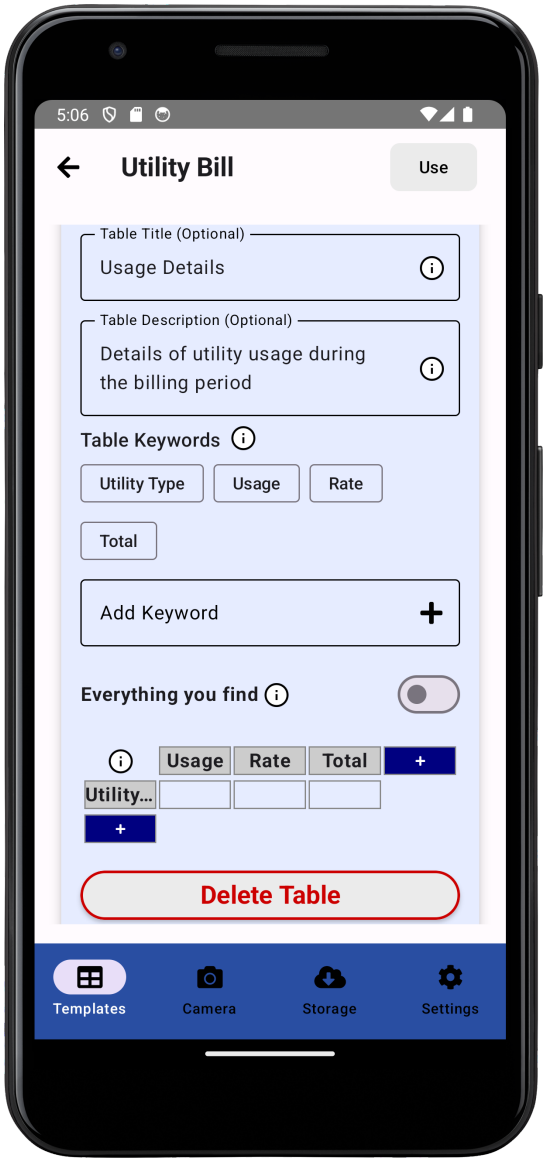


Figure 2.4: Table Editing

- **Title (Optional)**
  Not used in the extraction process
- **Description (Optional)**
  Not used in the extraction process
- **Keywords**
  Words present in the table, used to find the table in the document
- **Structure**
  Specify the cells or rows/columns you want to extract.
  You can specify the data type of each row or column and decide whether to extract through rows, columns, or cells.
- **Everything you find**
  Trumps the exact structure and returns the entire table found, dynamically.

2.2.4 Post Photo and Template Selection Screen

This screen is displayed after the user has selected an image or PDF and a template.

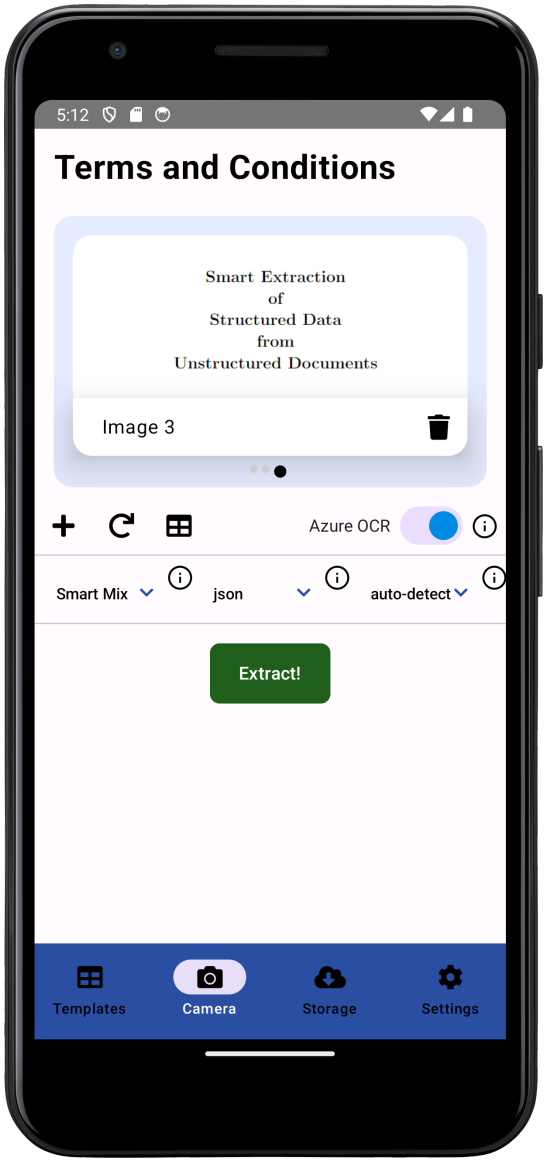
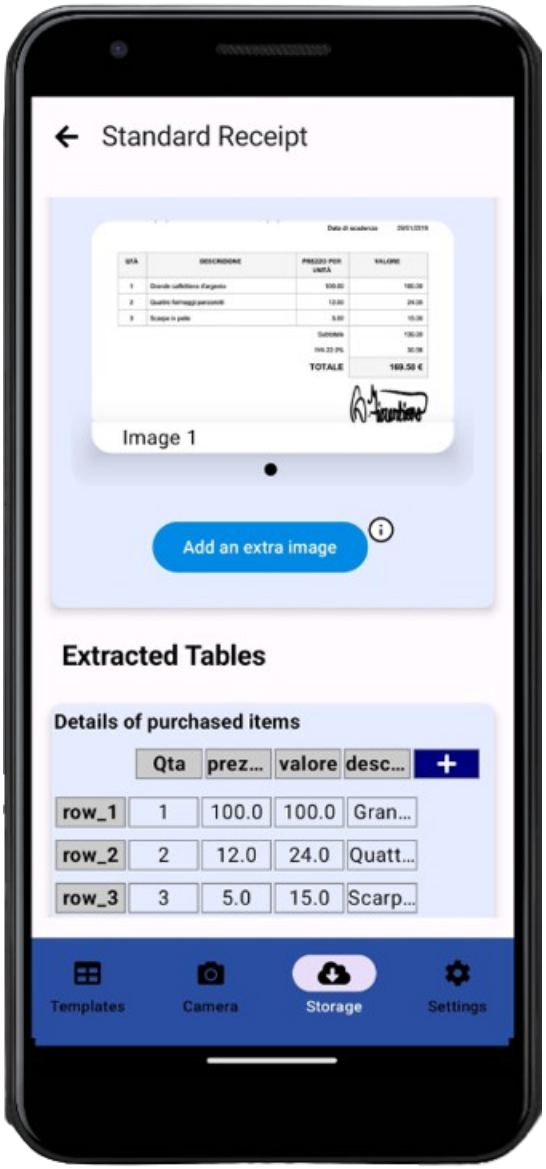


Figure 2.5: Post Photo and Template Selection Screen

- **Selected Image:**
  The selected image is displayed as a preview that can be expanded for a more detailed view.
- **Action Buttons:**
  - +: Re-opens the camera, to add other documents.
  - **Redo**: Clears all selected images and allows you to retake them.
  - **Edit**: Allows you to select a different template.
  - **Trash**: Deletes the current image.
- **Extraction Options:**
  - **Model:**
    Selection of the OpenAI model to use (Smart Mix, GPT-3.5 Turbo, GPT-4).
  - **Format:**
    Choice of output format for the extracted data
    (JSON, CSV, XML, TXT).
  - **Language:**
    Selection of the document language (auto-detect or manual: en, it, es, de, fr).
  - **Azure OCR:**
    Whether to activate Azure OCR.
- **"Extract!" Button:**
  Starts the extraction process.

2.2.5 Extraction Screen

This screen shows the data extracted from the document.



- **Document Image:**
  The uploaded image or PDF is displayed as a reference.
- **"Add an extra image" Button:**
  Allows adding logos or crests to the result.
- **Extracted Tables:**
  Extracted tables with headers and editable values.
- **Extracted Fields (not visible in the screenshot):**
  Editable fields extracted based on the selected template, with their found values and respecting data types.
- **Extraction File:**
  - The user can change the output file format as desired (e.g., JSON, CSV, XML).
  - The file can be downloaded, shared, or opened with other apps.

Figure 2.6: Extraction Screen

2.2.6 Settings Screen

This screen allows the user to configure the app's settings, including API keys.

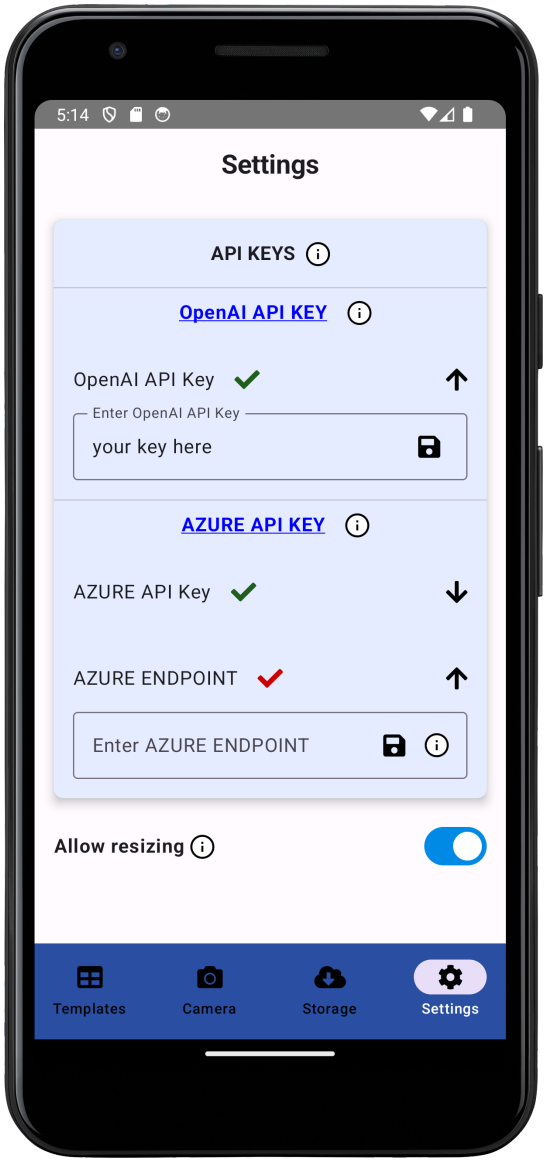


Figure 2.7: Settings Screen

- **API KEYS:**
  Section dedicated to entering the API keys required to use the external services.
  - **OPENAI API KEY:**
    Field to enter the OpenAI API key, mandatory for the app to function.
  - **AZURE API KEY:**
    Field to enter the Azure API key, optional and only necessary for table extraction or to use Azure OCR.
  - **AZURE ENDPOINT:**
    Field to enter the Azure service endpoint, required if using the Azure API key.
- **Allow resizing:**
  Switch to enable or disable automatic resizing of documents and images to 4MB.
  This option is useful for users who are not using the free Azure plan, which has file size limits.

Chapter 3

Implementation

The application was developed using Kotlin and the Jetpack Compose framework for declarative UI creation.
The architecture follows the Model-View-ViewModel (MVVM) pattern, separating logic from presentation and data.

3.0.1 Tech Stack

Architecture and Technological Choices

- **Kotlin Compose:**
  Compose permits the creation of a modern, reactive, and easily maintainable user interface, leveraging my past experience with component-based infrastructure.
- **Navigation:**
  Navigation between screens is handled using `NavHost` and `NavController`, which I found to be smooth and intuitive.
- **State Management:**
  `ViewModels` are used to manage the application's state, using `StateFlow` for reactive communication with Composable functions.
- **Dependency Injection:**
  Hilt was employed to simplify dependency injection, ensuring a cleaner and more testable code structure.

- **Local Database:**
  Realm was chosen as the local database for its ease of use, high performance, automatic data synchronization, and, most importantly, its object-oriented structure instead of table-based.

- **Extraction Service:**
  A foreground service was implemented to handle data extraction in the background, allowing the user to continue using the app or even keep it in background during processing.

- **Service-UI Communication:**
  A dedicated repository acts as an intermediary between the extraction service and the ViewModels, solving the problem of direct communication and facilitating shared state management.

- **Security:**
  API keys are securely stored using `EncryptedSharedPreferences` and `MasterKey`, ensuring the protection of sensitive data.

- **Multilingual Support:**
  DataDig supports five languages (English, Italian, Spanish, German, and French), with translated strings, base templates, and document extraction capabilities.

- **OCR:**
  The `TextRecognition` library was used for the in-house OCR implementation, to extract text from images and PDFs for processing.

Organization of Composable Functions

Composable functions were organized primarily by screen, to promote modularity and code re-usability.
Some generic components, such as buttons, input fields, and cards, were centralized in the `Misc` file to avoid duplication and simplify maintenance.
The extensive use of `Modifier` allowed for flexible and efficient customization of the appearance and behavior of the Composable functions.

3.1 Frontend and Design

The frontend of the DataDig app is implemented using Kotlin and the Jetpack Compose framework, adopting a Model-View-ViewModel (MVVM) architecture for a clear separation of concerns. The app comprises a single Activity and multiple screens navigated using `NavHost` and `NavController`.
The UI design emphasizes simplicity and usability, making extensive use of `Cards`, `Scaffolds`, and `LazyColumns` to present the information.
Icons are primarily sourced from FontAwesome.

3.1.1 ViewModels and Data Management

Three ViewModels are employed:

1. **TemplatesViewModel**:
   Manages the state and interactions related to templates (creation, editing, deletion).

2. **ExtractionsViewModel**:
   Handles the state and display of the extractions, including modifications and file generation.

3. **ServiceViewModel**:
   Facilitates communication between the UI and the extraction service, manages API keys, and handles all other global app settings.

A `Repository` class acts as a bridge between the extraction service and the `ExtractionsViewModel`, allowing for efficient state sharing and updates, and also gives access to the API keys, which are securely stored using `EncryptedSharedPreferences` and `MasterKey`, ensuring data protection.

3.1.2 Additional Components

- **File Creation Classes:**
  A set of classes is responsible for generating output files in various formats (CSV, JSON, XML, TXT) from the extracted data.

- **ImageOCR Class:**
  Implements OCR functionality using the `TextRecognition` library to extract text from images and PDFs.

- **Realm Models:**
  Two Realm model classes define the structure for storing extraction results and template data in the local database.

- **AppModule:**
  Provides dependency injection for `EncryptedSharedPreferences` used by the `KeyManager` for secure API key storage.

- **Multilingual Support:**
  String resources and base templates are translated into 5 languages, English, Italian, French, German and Spanish

3.2 Backend Implementation

DataDig's extraction process is handled within a Foreground Service, ensuring no effect on the user interface by performing the computationally intensive extraction tasks in a separate thread.

3.2.1 Integration of Python with Chaquopy

The core extraction logic actually resides within a Python script, accessed via the Chaquopy library.
This integration was motivated by three key factors:

- **Previous Experience:**
  Making use of my prior experience with Python libraries and architectures for similar projects allowed for faster development and higher code quality.

- **Frontend-Backend Separation:**
  Maintaining a clear separation between frontend (Kotlin) and backend (Python) facilitates a more modular and testable codebase.
  This also enabled direct testing of the Python backend within a separate IDE, making development and debugging much faster.

- **Educational and Novelty Aspects:**
  The integration of two distinct languages, Kotlin and Python, adds a unique dimension to the project and gives me an opportunity to explore cross-language aspects within the Android ecosystem.

3.2.2 Python Backend Structure

The Python backend is structured with a main entry point function that receives arguments such as chosen options, the template, the image or PDF, and a callback to retrieve encrypted API keys.
To optimize extraction time, the script employs lots of asynchronous calls to execute the main extractor functions concurrently.

Key components of the backend include:

- **Main Extractor Class Initializer:**
  This includes language detection functionality.
- **Text and Table Finder:**
  Identifies and retrieves text and tabular data from the input document.
- **Main Text-Based Extractor:**
  Performs the core extraction of key-value pairs from the text.
- **Table-Based Extractor:**
  Handles the extraction of data from the found tables.
- **Extraction Creator:**
  Generates the final extraction output in the desired format.

3.2.3 LLM Usage

The extraction process utilizes Large Language Models (LLMs) in a two-step approach:

1. **Initial Extraction:**
   The document's text and context are embedded in a type-specific prompt and sent to the LLM, which returns a raw draft extraction of the data specified in the template.
2. **Refined Extraction:**
   A dynamic pydantic class is generated from the template and used, along with the prievous draft extraction, as input to the LLM.
   This refined extraction ensures that the extracted values adhere to the data types and structures defined in the user's template.

The LLM function relies on a singleton class that holds the connection instances to communicate with GPT.
Of the two connecting function I have the first one is `extract`, the function that takes care of the initial extraction and returns a text result.
The following code snippet contains said function.

```
from langchain.chat_models import ChatOpenAI
from langchain.prompts import PromptTemplate
from openai import AuthenticationError, RateLimitError
from langchain.chains import LLMChain

    @classmethod
    def extract(cls, file_id, model, prompt: PromptTemplate,
    pages, template, temperature: float = 0.0) -> str:
        llm: ChatOpenAI = cls(model, temperature)._models[model]
        [temperature][hash(os.getenv("OPENAI_API_KEY"))]
        # Get the singleton instance
        try:
            chain = LLMChain(llm=llm, prompt=prompt)
            output = chain.run(context=pages, template=template)
        except AuthenticationError as auth_err:
            print("Authentication Error (Invalid API key?):",
            auth_err)
            raise ValueError("invalid OpenAI key")
        except Exception as e:
            print("Error in extract:", e)
            raise
        cls.calc_costs(file_id, model, inputs=[pages,
        str(prompt)], outputs=[output])
        return output
```

While the second and most interesting function is the `tag` function, responsible for generating a pydantic object from the initial extraction, refining it.
This is used both after calling `extract` and on its own when extracting tables, which are already compact enough.
The `tag` function:

```python
from openai import OpenAI, AuthenticationError, RateLimitError
from langchain.prompts import PromptTemplate
import instructor
from pydantic_core import ValidationError

    @classmethod
    def tag(
        cls, prompt: PromptTemplate, schema, file_id: str,
        model: str = "gpt-4", temperature: float = 0.0
    ) -> tuple[Any, Optional[Exception]]:

        error_occurred: Optional[Exception] = None

        if cls._openAI_instance is None:
            cls._openAI_instance = OpenAI()
        try:
            client= instructor.from_openai(
                cls._openAI_instance)
            output = client.chat.completions.create(
                model=model,
                messages=word_prompt,
                max_retries=3,  # Retry up to 3 times
                #if validation fails
                response_model=schema
            )
        except ValidationError as e:
            print("Validation Error:", e)
            output = schema()  # empty instance dummy output
            error_occurred = e
        except Exception as e:
            print("Error in tag (GPT-4):", e)
            output = schema()  # empty instance dummy output
            error_occurred = e

        cls.calc_costs(file_id, model, inputs=[str(prompt)],
        outputs=[str(output)])
        return output, error_occurred
```

Note: Due to a dependency incompatibility with Chaquopy, the `instructor` library could not be used for pydantic tagging.
This necessitated a workaround where the template is passed as text instead of pydantic class in the second prompt and a parser is introduced, potentially impacting accuracy and introducing the possibility of random parsing errors with big inputs.

```
from langchain.output_parsers import PydanticOutputParser
    ...
        output_parser = PydanticOutputParser(pydantic_object=schema)
        error_occurred: Optional[Exception] = None
        try:
            chain = LLMChain(llm=llm, prompt=prompt, output_parser=outpu
            output = chain.run(schema=schema.schema_json())
```

This is only a temporary limitation and is going to be addressed with future Chaquopy update, when this happens, a DataDig update will follow with the original already tested direct pydantic tagging.

3.2.4 Table Extraction

- **Azure Document Intelligence:**
  Azure's Document Intelligence service is used to identify and extract the tables from PDF pages or images.
  The service also offers inbuilt OCR capabilities in the same call, which have been found to outperform DataDig OCR engine by a good margin.
  Therefore, Azure OCR is automatically used whenever a page is scanned to retrieve a table.

- **Keyword-Based Filtering:**
  A keyword system allows the algorithm to recognize where and what tables are asked, optimizing costs and processing time by focusing on pages containing these relevant tables.

- **Table Processing:**
  Extracted tables are converted into LLM-readable text and processed using pydantic tagging.
  The extraction approach varies based on the table structure: cell-by-cell for fixed tables, lists for dynamic tables, and general matrices for fully dynamic tables.

It is important to note that the documents are resized to a maximum dimension of 4200px by 4200px or 4 MB because of the limit on the free version of Azure Document Intelligence.
While this can be a problem with extreme aspect ratios PDF pages, it is not expected to significantly impact OCR accuracy.
As demonstrated in [11], OCR technology can effectively extract text from images even at very low resolutions, maintaining high precision even with a 200x100 pixel image.

More experimenting with lower sizes and application or alternative preprocessing filters, such as noise reduction or skew correction techniques proposed in [5], could benefit the accuracy of the photos, and could be a focus of future developments.

3.2.5 Overall Workflow

The overall extraction workflow involves the following steps:

1. Language Detection
2. Table and Text Recognition (using Azure)
3. Main Text-Based Extraction
4. Table Extraction
5. Extraction-Class Creation

Here is a rough flowchart of the process, you can find the full link to it in A:

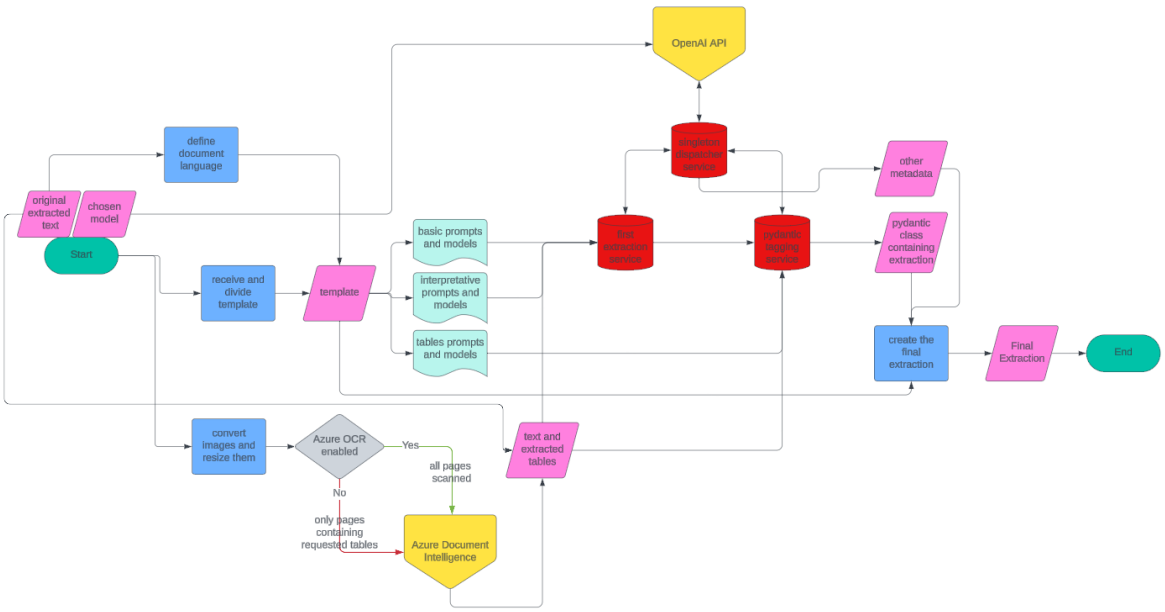


Figure 3.1: Diagram showing the extraction pipeline

3.2.6 Prompt and Template engineering

Crafting good prompts is fundamental in ensuring good accuracy and results, and because prompt engineering is a subjective field with objective results, the routes you can take and the philosophies you can employ are multiple.
Knowing how the LLM inner workings function, where and how the prompt is used greatly helps in determining what you should write.

In designing them I decided to go with a context focused prompt, subdividing it into context, instructions and usable text and template, so that the LLM could distinguish and vectorize them separately, not confusing the sections.
Prompts are also tailored to the specific extraction task (normal, interpretable, or table extraction) and are available in all supported languages.

While studies like [6] have demonstrated that longer prompts (up to 40 lines) can improve accuracy in certain tasks, it's crucial to acknowledge that prompt length is not the sole determinant of effectiveness.
In the context of DataDig, where prompts may significantly exceed 40 lines due to the inclusion of the template and document text, brevity and clarity become most important. Concise and well-structured prompts not only contribute to cost efficiency by reducing the number of tokens processed but also prevent overwhelming the LLM with excessive information, which could potentially hinder its ability to discern relevant patterns and extract accurate data.

The prompts I created and that I suggest define clearly and briefly the extraction goals, expected output, and the relevant data sources, following a structure and workflow similar to the one suggested by [3].
As I said, I believe the essence or undercurrent of a prompt to be what is most important, as that is how the LLM will analyze it trough contextual embeddings.

Template engineering is also a way of prompt engineering in my app, as the template is injected into the prompt, which means that every user will have a part in it, and I found understanding what gives the best performances is a developed skill that you acquire though experience with the app.

I interestingly discovered that short or lazy titles and descriptions work best, and adding informal requests in the description is not off the table, especially in interpretable fields, varying what you can get the LLM to output and giving it a multipurpose role.
I also found that while generalizing titles can be scary, it hardly makes a dent in the accuracy, while opening up for highly diverse documents.

3.2.7 Scaring the LLM

I noticed a phenomenon while prompt and template engineering and while working with interpretable fields.
During the development process, I often observed a failure to extract information when being too precise or giving too many instructions.
While it might seem intuitive that more detailed and rigid prompts would yield better results, overly specific instructions can sometimes have the opposite effect, and imposing strict protocols and procedures "scares" the LLM into avoiding anything it considers remotely risky.

Especially when providing overly specific instructions and demanding high precision while offering a "way out" in the form of a default value, the LLM will prefer the default even when the correct value is present in the text.
This was actually a big problem in the interpretable fields: even if the LLM was asked to interpret the context for the response, while still prioritizing extracting directly if the data was in the document, the LLM was so scared of interpreting the text and returning something that was not present in it that it would default, even if marked as highly discouraged.

Because the default value was poisoning the results, and any encouragement on improvising or discouragement on using the default was not working, I decided to cut it completely for the interpretable fields, disabling it.
This solved the problem, and the LLM was forced to "risk" giving a subjective response, which is exactly was I want to be returned in an Embedding-based retrieval, as the system's flexibility in giving an "acceptable" answer contributes to its overall performance, like discusses in [7].

A similar effect I noticed you can achieve by over-engineering the template and exaggerating with the descriptions and titles, this gives the impression that you ask for extreme specificity, and any small divergence with the document, and sometimes even if there are no divergences, scares the LLM into marking it as non present.
This suggests a need for careful balancing of prompt specificity and flexibility.

3.3 Security

This section addresses the cybersecurity considerations relevant to DataDig.
As an installed mobile application that does not collect any of the user information, the app inherently starts with a less vulnerable base than web or cloud-based alternatives. It does however handle sensitive information, including private user-provided API keys and documents that could be considered confidential.

3.3.1 Keys security

All keys are secured into an encrypted AES-256-GCM Shared Preference, which stores them securely and protects them even if the device has been compromised, as the keys for the encryption are kept in Android Keystore under root access.
API keys are also passed from the Kotlin application to the Python script via callbacks and environment variables, making sure that even if there was a way to check system logs or messages sent to the script the keys would not be accessible.
The keys are not even available to the user, having a write only access mode like most other platforms do.

Memory attacks are probably where the keys can be found, either in the environment or during callbacks, clipboard attacks can be employed too if the user has copied recently the key to insert it.
While theoretically possible, all these attacks would require heavy physical access to the device with specialized tools, and mitigating this is beyond the scope of my project, especially considering these keys are generally not valuable enough for the effort.

OpenAI and Azure keys also have inherit high security, with online scans, controlled access, easy replace-ability and limits to the payments.

3.3.2 Document security

The scanned documents and the extracted information can be considered confidential and private, this is why the extractions are never shared, even for analytics, and are kept locally on the device.

Contrary to the keys, past extractions are not kept encrypted, but simply saved into a local realm database, which still keeps security as it can only be accessed by the application due to how Android Permissions works.
Images and PDFs on the other hand are not saved as a copy, but only referenced, which means if a user decides to delete a confidential document, this will also be reflected in the app, keeping control of ones documents.

Still, if an attacker got root or direct access to the device memory, the data would be compromised, but this is the case for every other app.

3.3.3 Data privacy

DataDig collects no personal information from its users, as no login or email is needed for the app.
The app is also GDPR compliant, complete of Privacy Notice,

The only forms of data collection employed are:

- The optional anonymous extraction ratings, which are sent to an external database, they contain only a vote and a description, and are not connected to the extraction or the user.
- Google Firebase Crashlytics for monitoring production crashes.

3.3.4 Prompt injection

A rising discussion in the recent use of LLM in web pages and applications, especially those where input from the user is directed directly towards the LLM is Prompt injection.

It is in fact true that anyone could insert malicious text to try to break the prompt or change the focus of the LLM by just adding it in the template, and while this could potentially break the extraction process, it is not possible to use this maliciously to access sensitive information.

Security measures are in place to minimally pad some prompt injection, like clear division in the prompt between what is described user input and the system instructions, but these methods are unreliable and can be worked around with prompt engineering by the attacker, so it is a waste of prompt space to add more meta information to help the LLM detect fraud.

That said I do not believe this to be a serious problem, data needs to get parsed to an object and any prompt injection will just make the extraction fail.
It is also virtually impossible for it to lead to the exposure of sensitive information, as API keys and user data are not accessible to the LLM.
However, further research is needed to develop robust defenses against this emerging threat.

The only thing I can think you could gain by prompt injection is a vague understanding of the architecture, by following the errors that are given in making the parsing fail or by inserting problematic control characters in the data.
This is also a non-problem, given the whitebox-like programming I used, and could be done better by just reading this thesis.

3.4 The AI discussion

3.4.1 Why is AI needed?

The increasing use of AI technology has raised a lot of concern about its appropriate application (see [9]).
This section justifies the critical role of LLMs in DataDig's data extraction process.

Typically Data Extraction can and is being done without Artificial Intelligence, by using position based extractors that utilize OCR to extract the key-value pairs based on a positional template, as demonstrated in [11].
Extracting from text however is much more difficult, as you have to reference keywords or titles, making the templates very susceptible to variations in document structure or formatting.
This means context-less extractors, while fast, cheap and good at large scale extractions, not only need ad-hoc templates which are difficult to create and maintain, especially for individual users, but are very fragile, as minor changes to the model would invalidate such template.

3.4.2 Ethical Implications

While AI-powered data extraction offers significant benefits in terms of efficiency and accuracy, it also raises ethical considerations that must be addressed [10].
One major concern is the potential for profiling, data hoarding, and privacy violation.
It is in fact easy with this kind of technology to scan personal data without losing on context and understanding of the individual.
It would be possible in fact to extract sensitive personal data even where it wasn't possible before, this includes social media posts, text messages, medical records, government documents, and anything containing a part of your life or information about you, creating a profile and understanding of your character.
This information could be used for targeted advertising, discriminatory practices, or even identity theft.

While DataDig focuses on small single extractions, the field of data extraction already encompasses larger and trainable models, which are likely to be used or are already being used in this manner.
It is important for tools of this kind to be aware of the risks and work against the misuse of the application.

The broader issue of AI regulation and compliance should also be considered.
As the European Union's GDPR and similar data privacy laws continue to evolve, it will be crucial for DataDig's to remain compliant with future regulations on AI use, especially as it deals with sensitive personal information.

To mitigate these ethical risks, it's crucial to prioritize data privacy and security through access control [13], to protect the sensitive private information.
Considering the ethical risks, it's important to adapt to a world where information and now understanding becomes cheaper by the day.

Chapter 4

Testing, Accuracy, and Results

The performance of the application has been thoroughly tested both by myself and by external testers.

4.0.1 Manual Testing Methodology

Establishing appropriate parameters for evaluation was very challenging, as it required a well-defined standard for document and template quality and consideration of numerous influencing factors.

Factors Affecting Extraction:

There is no way around it, the extraction is a child of the environment, context, options, text and prompt that gets fed into the LLM, and everything influences it.
That said, some factors have a way more significant impact than others.

I believe the following factors to be the ones that can actually impact the accuracy of data extraction:

- **Document Quality**
  How comprehensible, well written and coherent the document is.

- **Representation Quality**
  The quality of the document's visual representation, including image quality, skew, graininess, and dimensions (especially for photographed documents).

- **Document Complexity**
  The complexity of the language and subject matter, including the use of metaphors, complex or archaic language, and unconventional phrasing.
  This factor is also closely related to document quality.

- **Document Length**
  How many words and so tokens is the document .

- **Document Language**
  The language of the document, including potential challenges posed by archaic language, low-resources or mixed-language content.

- **OCR Engine Used**
  If the user opts to use the in-house free OCR or permits a whole scan by the Azure OCR Service.

- **OCR Result Quality**
  The accuracy of the OCR process is a critical factor, as errors in text detection or recognition can significantly impact the quality of the extracted data and often do.

- **Template Complexity**
  If what you are asking is simple and straightforwards or very complex and context focused.

- **Template Vicinity to Document**
  How closely the template structure matches the document layout, if the words chosen are the same or if it needs to rely on general meaning.

- **Template Quality**
  How well you explain what you need, using descriptions or adding context.
- **Template Length**
  How many fields you are asking to extract.
- **Table Keyword Correctness**
  The presence of the keywords provided for table identification.
  Incorrect or missing keywords can lead to the failure to identify the table completely.
- **Table Vicinity to Template**
  How similar the columns or rows you gave are to the ones in the document.
- **Table Size**
  How many cells, columns and rows the table contains.
- **Table Dynamism**
  If you are using a fixed, dynamic rows/columns or fully dynamic table in the template.
- **Table Representation in Document**
  How the table is displayed in the document, missing lines, unaligned rows, non traditional shape, weird representation are very common and pose a big problem.
- **Model Used**
  What LLM model the user opted for, GPT-3.5 Turbo, GPT-4, or Smart Mix.
- **Randomness**
  The inherent variability and potential for unpredictable outputs from LLMs.

Testing Focus

In my testing, I separated the evaluation of tables from the classic fields, as their extraction processes are independent and so the result would be independent too.

Of the millions of attributes listed I primarily focused on:

- Template and Document Complexity
- Template and Document Length
- Model Used
- Table Dynamism
- Template Vicinity to Document

To isolate the impact of the factors under investigation, certain variables were controlled. PDF documents were primarily used, and Azure OCR was employed to free the extraction of any OCR issues, as they are very unpredictable and random, compound on LLM errors, are difficult to distinguish, and have been already largely tested in other studies such as [5].
Language was not extensively tested due to my linguistic limitations, and table keyword correctness was not directly evaluated, as it's not related to LLM performance, and can be predicted easily .

Test Setup

I created a 2-page, 560-word document and accompanying template as a base to use for the tests.
It was created trying to achieve a very standard complexity, quality, vicinity to the document and length, to simulate a common use case.

Extraction Fields:

- 2 text fields
- 1 list of 3 text items
- 1 float (temperature)
- 1 number (year)
- 1 date
- 1 boolean interpretable field
- 1 text interpretable field

The template was designed to cover all possible facets and features of the extraction process.

4.0.2 Text Attributes Testing

Document Complexity:

The base document just described was subjected to modifications to introduce varying levels of complexity:

- **Normal**: Formal language and straightforward sentence structures.
  This represents the original, unmodified base document.
- **Hard**: Increased use of metaphors, adjectives, and complex phrasing, still readable but excessively complex.
- **Extreme**: Extensive use of metaphors, old speak and almost poetic style and writing, making the text challenging both to understand and also read.

While each increase in complexity resulted in a minor increase in document length, I do not expected this to significantly influence the results, as I will demonstrate in the subsequent document length testing.

Results (Document Complexity)

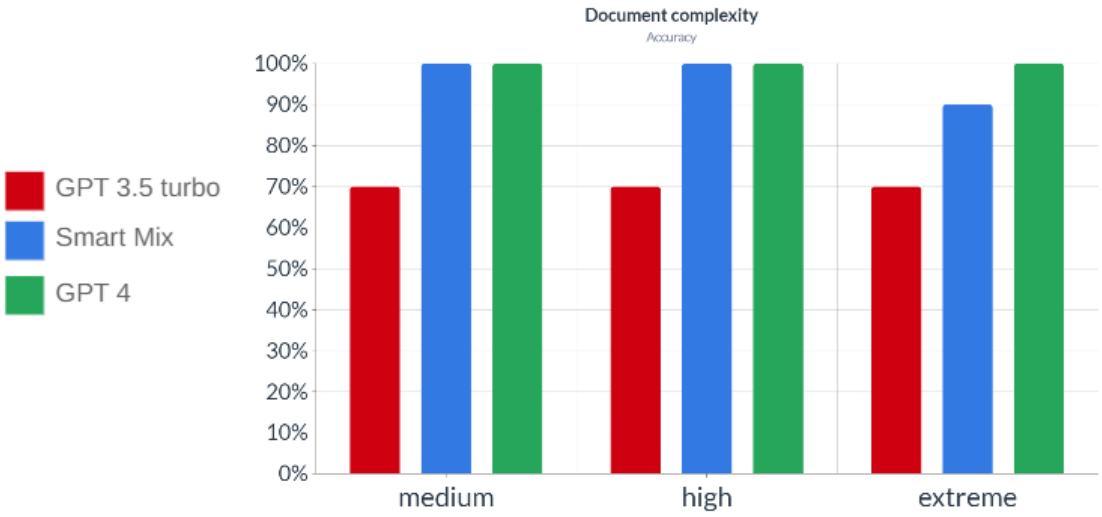


Figure 4.1: Average accuracy for different document complexities

- **LLM Resilience to Complexity:**
  The LLM demonstrated remarkable resilience to increasing text complexity.
  This robustness is likely attributed to its capacity for contextual understanding, and its ability to filter out irrelevant information.

- **Impact on Interpretable Text:**
  The quality of the subjective overview generated for the interpretable text field decreased slightly with higher complexities, consistent with the increased complexity of the documents.
  Notably, the length of these responses showed significant variability in repeated tests with the highest complexity, ranging from approximately 15 to 65 words.

- **Model Performance:**
  GPT-4 consistently achieved the best performance, while GPT-3.5 struggled, particularly with the date and two text fields.
  The "Smart Mix" model aimed to balance performance and cost by using GPT-4 only when necessary performed just as well, with a little drop in the higher difficulty.

Extraction Speed (Document Complexity)

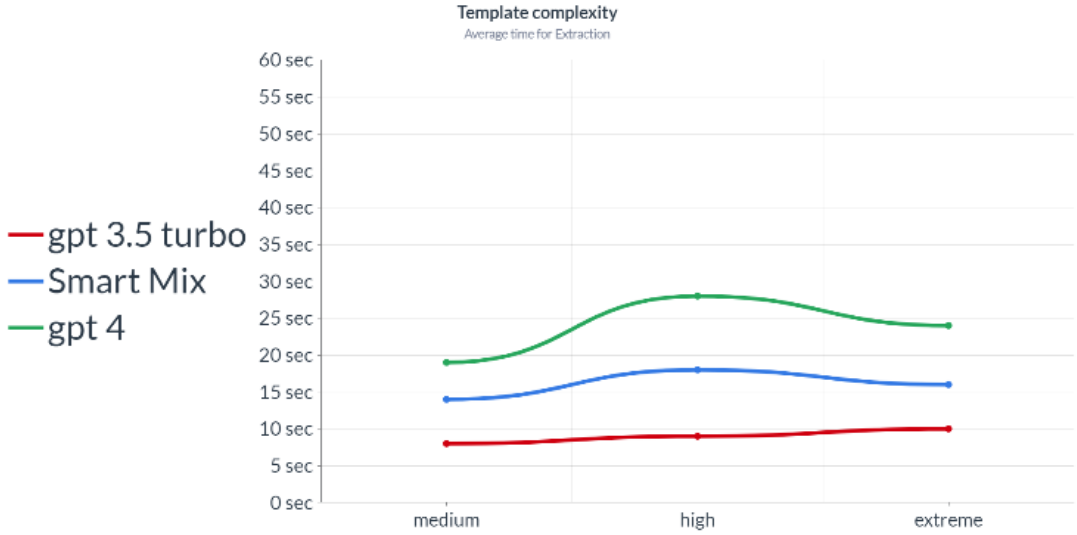


Figure 4.2: Average speed for different document complexities

- **Extraction Time:**
  GPT-3.5 Turbo as expected was the fastest, followed by the Smart Mix, with GPT-4 being the slowest.
  It is safe to say that text complexity did not significantly impact extraction time, with the variations being likely due to other factors like the increased text length.

  This is probably the case because of context vectors, as I doubt that bigger worded sentences take much longer to digest than a normal paragraph, and the context and meaning remains the same, maintaining good understanding and with it extraction accuracy.

Document Length Testing

To assess the impact of document length on extraction performance, the complexity of the base document was held constant, mainly by adding similar paragraphs in between the existing ones.
The following document lengths were tested:

- **2** pages, **560** words, **3500** characters (the base document as established before)
- **5** pages, **2000** words, **12500** characters
- **10** pages, **3000** words, **18000** characters

Excursus on Maximum Document Size

The maximum document size is currently limited by the API constraints, which allow for approximately 6000 words for GPT-4 and 3000 words for GPT-3.5 Turbo.

Considering the overhead introduced by the prompt, template, and LLM output, the practical limit for the input text is estimated to be around 2500 words, after which it returns an error, regardless of the chosen model.
This means it is possible to reach it with a combination of a long and lengthy text or/and template, although difficult to achieve, this stunts extractions on huge PDFs, and is not ideal.

With better support and future development, batch extractions of smaller text chunks could potentially address this limitation, but it would require significant redesign and careful management of API rate limits, as it would need to account for various factors.

Results (Document Length)

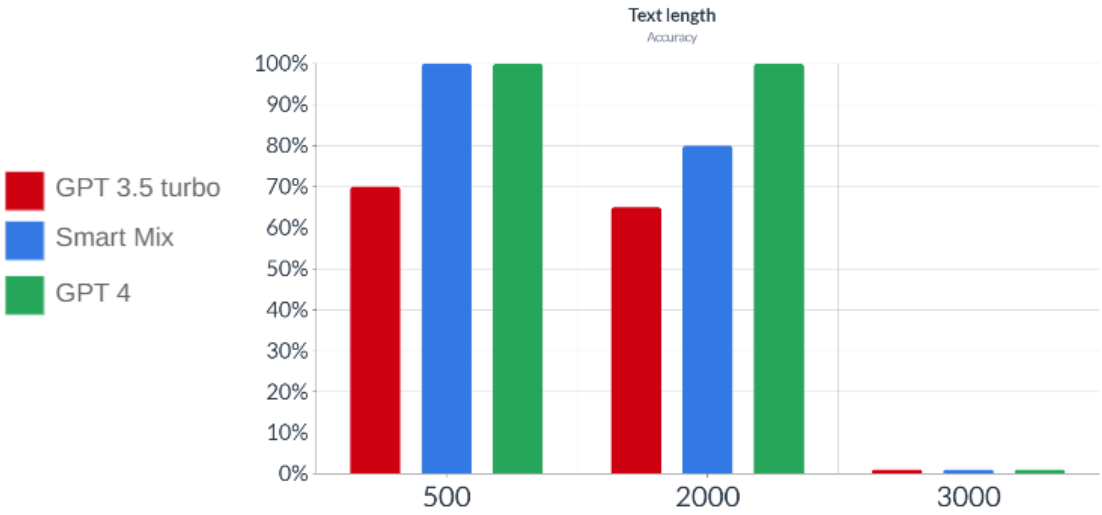


Figure 4.3: Average accuracy for different document length

- **API Limits:** The 3000-word document exceeded the API token limit and failed to produce any extraction, failing completely.
- **Model Performance:** GPT-4 maintained high performance even with the longer document. The Smart Mix missed a couple values, and GPT-3.5 continued to struggle, losing quality in an interpretable field, that was only partially credited.

Extraction Speed (Document Length)

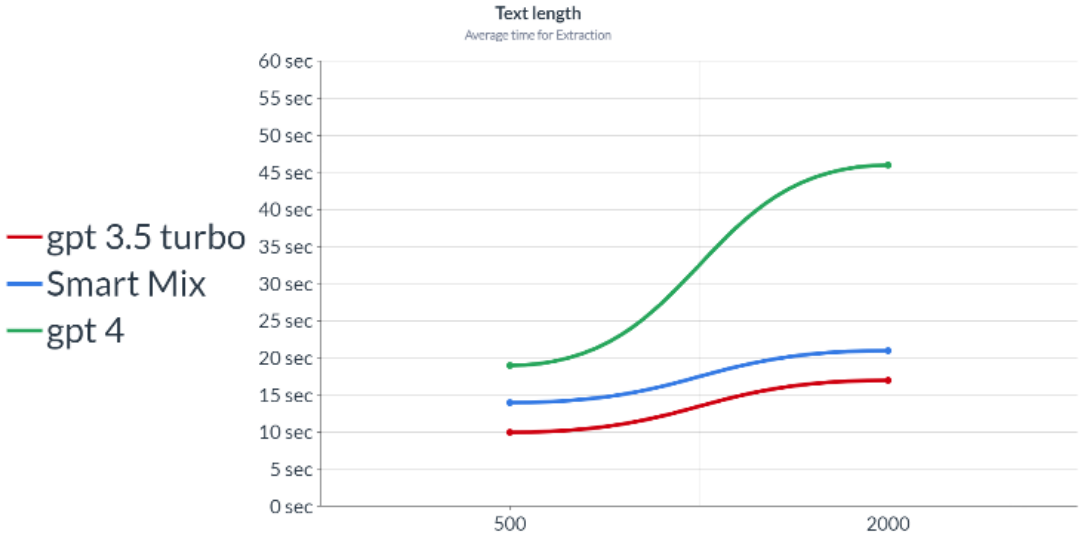


Figure 4.4: Average speed for different document length

- **Extraction Time:** Extraction time increased predictably with document length, impacting GPT-4 the most.
  This clearly shows the strength of the mixed model, that uses GPT-3.5 turbo for the scalable text extraction, limiting cost and time

4.0.3 Template Attributes Testing

Template Clarity

Using our original battle-tested base document, the clarity of the template was varied, but what does clarity mean?

The clarity or quality of a template refers to the extent to which the defined fields are well-articulated, unambiguous, and aligned with the structure and content of the document.
A template is clearer than another if the descriptions are better suited, the titles are closer to the ones in the document, and more care and consideration is put into the template engineering (see 3.2.6)

- **Clear Template:**
  Well-crafted high quality template with clear descriptions and close alignment with the document structure.
- **Lazy Template:**
  Descriptions shortened or removed, wording generalized, less attention to detail, close to what someone hastily trying the app would try.
- **Abysmal Template:**
  All descriptions removed, titles replaced with loosely related terms.
  I still kept a connection between the value and the field, to allow for some possibility of extraction, though it would require significant interpretation by the LLM.

Results (Template Clarity)

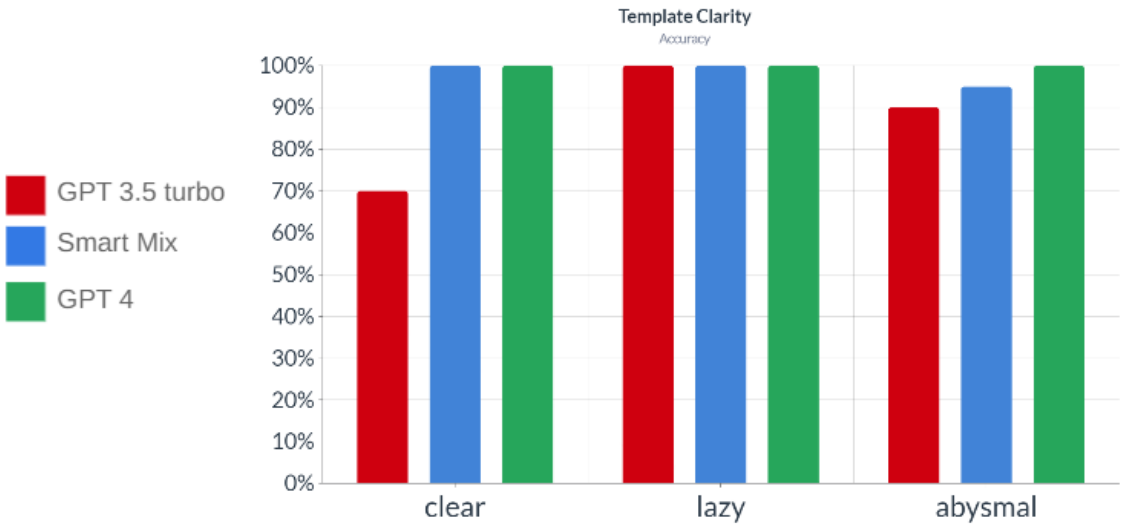


Figure 4.5: Average accuracy for different template clarity levels

- GPT-4 maintained perfect accuracy across all template clarity levels.
- The Smart Mix encountered minor issues with an interpreted field in the lowest quality template.
- Interestingly, GPT-3.5 demonstrated improved performance with the 'lazy' template compared to the high-quality template. This suggests that excessive precision in prompt engineering can sometimes be counterproductive, potentially leading to the phenomenon of 'LLM scare' discussed earlier in 3.2.7.
- Extraction time remained consistent regardless of template clarity, as discussed with document complexity, text redesign make minimal impact and is irrelevant in time calculations.

4.0.4 Table Testing

Table extraction was tested separately, focusing on the combination between table size, model used, and dynamism selected.

Given the multi-factorial nature of this evaluation, the complete test results are presented in tabular form in appendix C.1.

Here a deeper analysis will focus on the interaction of three factors at a time, averaging over the remaining factor, as it would need a very complex graph otherwise.

- **Table Sizes:**
  - **Normal**: **4** columns, **6** rows (**24** cells)
  - **Big**: **8** columns, **12** rows (**96** cells)
  - **Huge**: **15** columns, **15** rows (**225** cells)
    Representing the maximum size configurable within the template.
- **Dynamism Levels:**
  - **No dynamism**
    Fixed bi-dimensional structure
  - **Dynamic rows**
    This configuration, with fixed inputted columns and dynamic rows, represents the most common use case.
    You would see this in the app as a N x 0 table.
  - **Dynamic columns**
    This less frequently used configuration involves fixed rows and dynamic columns.
    You would see this in the app as a 0 x N table.
  - **Fully dynamic**
    This mode allows for the extraction of the entire table without specifying rows or columns, either by leaving the table structure empty or by activating the 'Find everything' button.
- **Models:**
  - **GPT-3.5-Turbo**
  - **GPT-4** (used in the Smart Mix for table extraction, so they coincide)

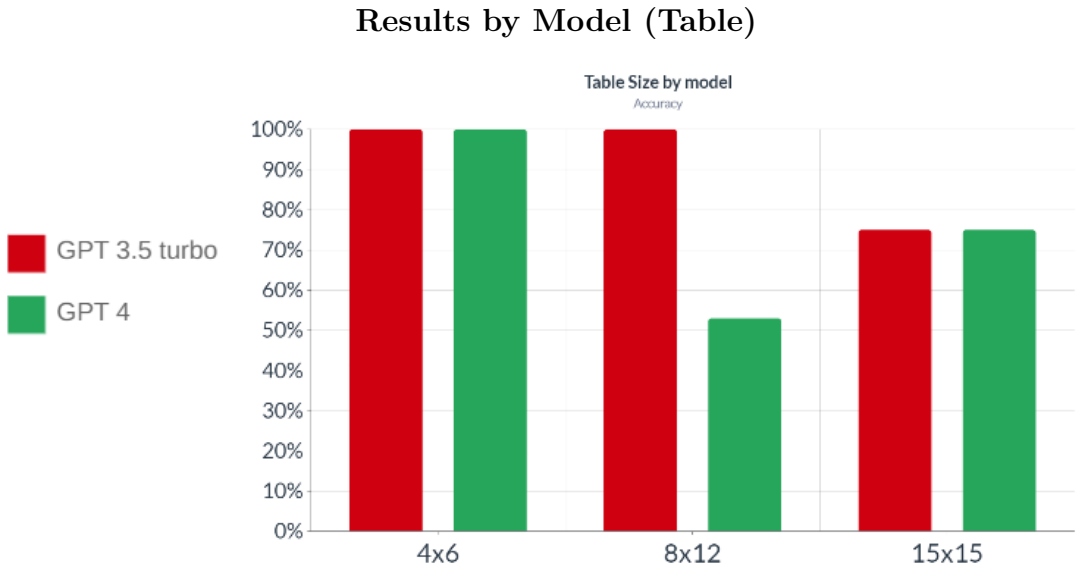


Figure 4.6: Average accuracy for different table size by Model

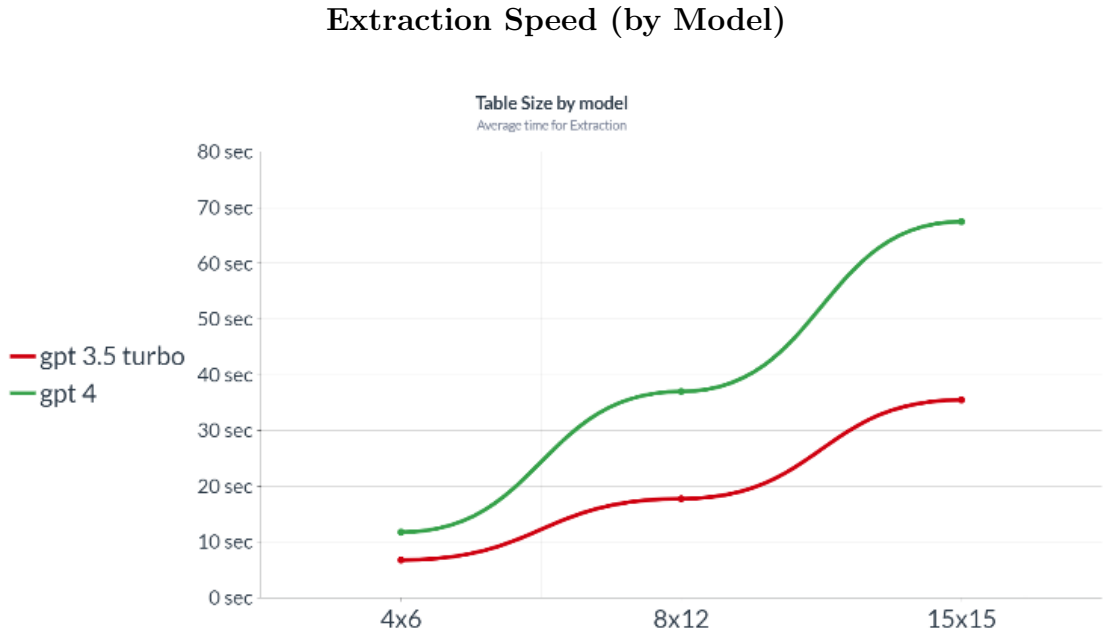


Figure 4.7: Average speed for different table size by Model

- Interestingly, GPT-3.5 Turbo exhibited superior performance in this scenario, as it did not encounter the difficulties observed with GPT-4 in handling dynamic columns.
- The drop in performance of the GPT-3.5 turbo can be attributed to the maximum input size limitation encountered when extracting large non-dynamic tables.
- Extraction time remained consistent with the models as expected

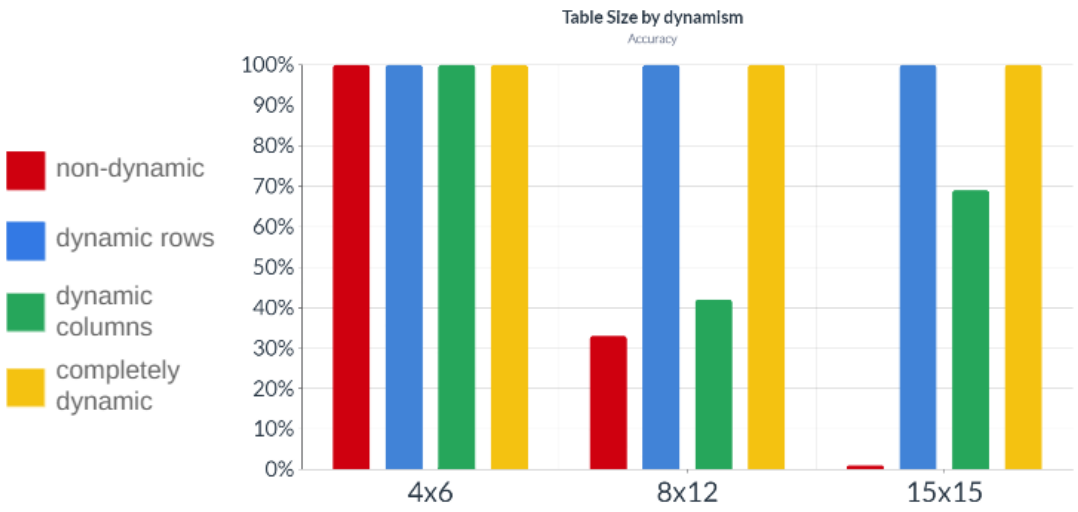


Figure 4.8: Average accuracy for different table size by Dynamism

- The extraction of large non-dynamic tables consistently failed due to parsing errors, it seems the temporary parsing replacement fails on big outputs.
  As this is solution is temporary I fully believe the real pydantic tagging will solve this no problem.
- Dynamic columns also get a drop on accuracy, returning only partial input, this is either cause of confusion or sub-optimal prompt engineering, and warrants further investigation.
- All other extraction came out perfectly even with big tables, which was very promising.

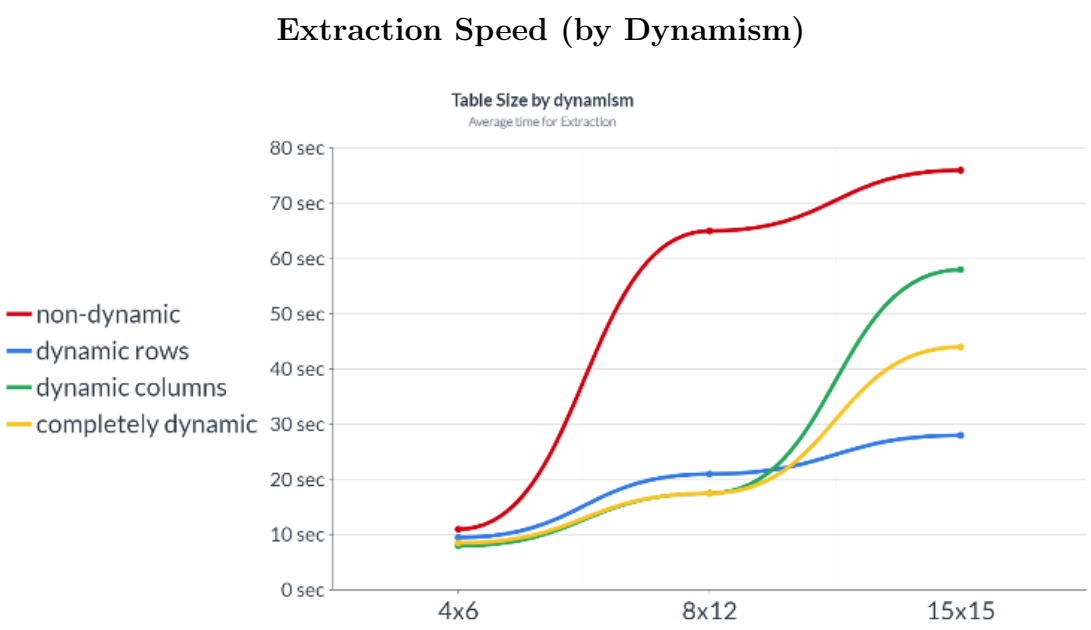


Figure 4.9: Average speed for different table size by Dynamism

- As expected, defining more and more constraints by defining both rows and column added to the time it took to extract the table.
- The rest of them kept pretty close together, with dynamic columns taking a bit longer, as most of the tables had more rows than columns, keeping a bigger input.

Discussed Results (Table Extraction)

- **Small Table:** Perfect extraction across all models and dynamism levels.
- **Medium Table:** Perfect extraction except for dynamic columns, which only extracted the first column, and some non dynamic extractions.
- **Large Table:** Failed extraction for the non-dynamic case due to message length limits.
  Perfect extraction for almost all other cases.

Observations:

- **Size:** Table size generally did not affect extraction accuracy, except for the length limitation in the non-dynamic case with the largest table, and the dynamic columns response.
- **Model:** Minimal difference between models, with GPT-3.5 turbo slightly outperforming GPT-4 in dynamic columns, god knows why.
- **Dynamism:**
  - Fully dynamic tables had some inconsistencies in including headers, but gave perfect performance on the values.
  - Dynamic columns posed challenges, particularly for GPT-4.
  - Dynamic rows and non-dynamic tables (within size limits) were extracted accurately.
- **Time:** Extraction time doubled when switching from GPT-3.5 to GPT-4. For non-dynamic tables, time generally increased linearly with the number of cells.
  For dynamic tables, time more or less increased linearly with the number of fixed rows or columns.

4.0.5 Time considerations

The extraction time is influenced by a multitude of factors more than the accuracy considerations discussed previously.
These factors include things like the response times and potential congestion of the OpenAI and Azure servers, the hardware capabilities of the user's device, the Android version, the presence of cached data, and other variables.

Measuring accurately the extraction time proved difficult due to the fluctuations in server response times and other external factors, and keeping the same parameters was especially hard considering the reliance on outside services.

To account for variability and ensure reliability, each extraction was performed multiple times and the data was averaged to generate the graphs presented earlier, but it is important to consider how it can fluctuate and generally respond.

I also found a "cold" overhead time, this is a fixed amount of time applied to the first extraction performed after launching the app.
I estimate it to be around 9 ± 1 sec generally, and can be observed by measuring the difference of time before the service notification get sent, as this is applied before Python gets called.

To assess the variability and consistency of the extraction times, variance and coefficient of variation (CV) were calculated.
The CV, which is the standard deviation as a percentage of the mean, provides a normalized measure of variability, and that means it gives us an actual comparisons across the different extraction scenarios, regardless of size differences.

As a baseline, the variance observed for the base document used in the previous tests was $0.37\,\mathrm{s}^2$ variance.

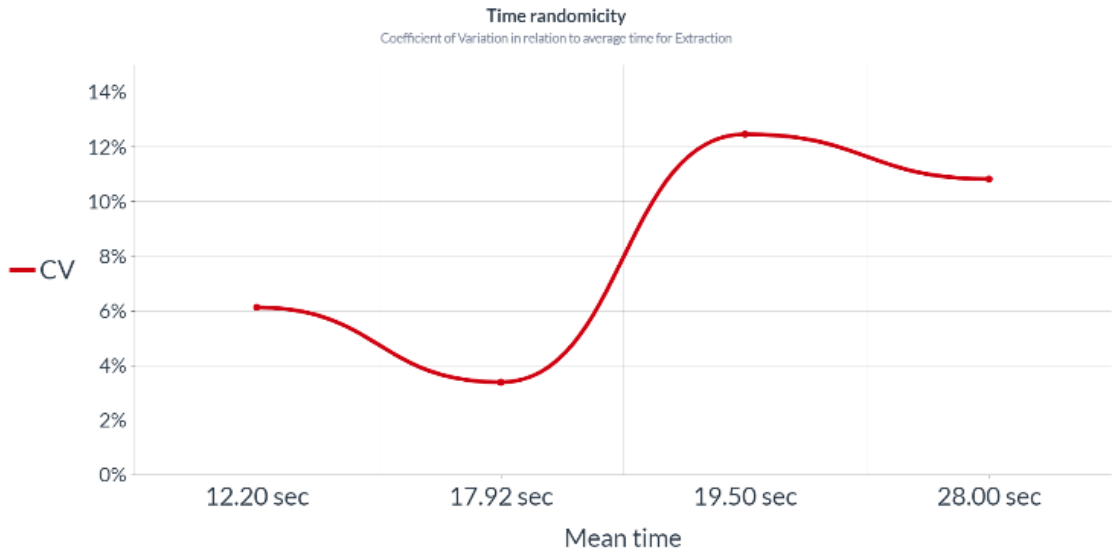


Figure 4.10: Coefficient of variations in different tests

- We can observe that bigger extraction offer a bigger variance in time, probably due to the more factors in play.

From a developer perspective, opportunities for further optimizing the extraction speed are very limited, as the execution time of the application code constitutes a negligible portion of the overall processing time.

The primary determinant of extraction time is in fact waiting on the external APIs. This latency is largely unavoidable. However, by ensuring that API calls are executed concurrently whenever possible, the process can be optimized.

DataDig already optimizes extraction time by extensively using asynchronous operations, this minimizes idle time and maximizes efficiency, leaving limited room for further speed improvements.

The only potential way to reach better speeds would be to perform the table search and text extraction asynchronously.
However, as the OCR employed by the Azure table extraction service yields superior results compared to my built-in OCR engine, I decided to prioritize accuracy over speed.

For larger and especially mixed extractions DataDig also features a real-time progress bar that provides feedback on the completion of each processing step.

4.0.6 Conclusions on Manual Testing

By addressing the input size limitations through switching to the real pydantic tagging, and doing further investigation on the dynamic column extraction behavior, I believe the accuracy of table extraction can be significantly enhanced, potentially approaching 100%.
While this may appear optimistic, the binary nature of the results (either complete extraction or complete failure) suggests that the targeted solutions can effectively address the identified issues.

With the LLM space becoming better and better each year it is not absurd to anticipate that future models will offer significant improvements in both speed and accuracy. Furthermore, the introduction of models with larger context windows, such as GPT-4-32k or a resigned architecture could allow longer and multiple extractions to be parallelized.

It could also be interesting to see real side by side testing of the app against manual extraction like we see in [3].
I have considered calculating Krippendorff's alphas for the values, but I believe it would not mean much if a human counterpart is not considered.

4.1 User Testing and Feedback

In addition to internal testing, the app underwent user testing with friends and family, who were provided instructions on various tasks to explore its full capabilities.
A questionnaire was also administered to gather feedback on both task completion and the overall app quality and intuitiveness.

It is essential to recognize that, while DataDig is designed for individual users, some level of technological knowledge is required.
This aspect became evident during user testing, highlighting the importance of considering users of different backgrounds and technological proficiency when testing.

Closed Testing

To publish the app on Google Play and get feedback and bug testing done a closed testing was started.

I set up 4 different extraction tasks that I felt would represent everything you could do with the app, and asked the 25 participants to download the app and perform these 4 types of extraction:

- **Basic extraction:**
  Using a preexisting template, take a photo and scan any ID or business card.
- **Interpreted extraction:**
  Using the preexisting "News Article" template, take a photo and scan any article or news story.
- **Table extraction:**
  The testers were tasked to find any document that contained a table on it, an utility bill was suggested for this.
  The testers were also tasked to go into the corresponding template and check that the table matched the one in said template before extracting.

- **Custom extraction:**
  The testers were asked to find any document or text around and create a new custom template to fit to it for the last extraction.

Testers were given several ways to review and rate their experience, they could leave a review on Google Play, send anonymous ones through the app, or as suggested fill in a Google Form questionnaire.
Said questionnaire asked not only general questions about the app, suggestions and bug reports, but also specific feedback on each task performed, with a corresponding rating.

Feedback received

Some interesting notes on the questionnaire feedback:

- **Extraction quality** was an unexpected success with normal fields, dropping slightly with tables in a couple cases.
- **Graphics and design** was rated decently with all testers rating it 4 or above out of 5.
- The **general app** was given an average of 9/10.
- Interestingly, some testers provided lower ratings for the **template creation and editing** process.
  This may be attributed to difficulties in locating the "new template" button, an issue that was also observed during in-person testing, or it could be the overwhelming options present for customization.
- Users primarily suggested **improvements** to the app's visual design, but their feedback often lacked specificity and stayed very vague.
  Some of these suggestions have been incorporated into the app's design, others I do not think would benefit the app, like adding a home screen.

Other feedback and Improvements:

Anonymous in-app feedback was useful to understand overall extraction happiness, and although I lost some data due to Supabase suspending the table, most of the feedback was still collected and showed great satisfaction by the users, 93% of the feedback awarded a full score to the extraction, while the remaining feedback primarily reported minor issues such as missing values, incorrect column extraction, or minor typographical errors, likely attributable to OCR limitations.

Only two instances of complete extraction failure were reported, one due to a parsing error that has since been addressed, and one did not specify the reason.

In addition to addressing specific bug reports, primarily in tables, and improving the prompts for interpretable fields to mitigate "LLM scare" (see 3.2.7), several UI/UX enhancements were implemented based on user feedback:

API Key Input:
Initially, the API key input field included a separate "save" button to prevent accidental overwrites.
However, user feedback indicated that this design often led to users closing the input field without saving their keys.

To address this, the input field was modified to automatically save the new key upon closing if the field is not left blank.

Extraction Result Access:
The initial design presented two buttons on the extraction results screen: "Go to Extraction" and a direct file access button.
This was intended to provide quick access to the extracted file for users primarily interested in integrating DataDig into their workflows in a corporate settings.

However, user testing revealed that many users overlooked the "Go to Extraction" button and attempted to access the file directly, leading to confusion and error messages if they lacked a compatible app.

To improve the user experience, the direct file access button was removed from the initial results screen, requiring users to scroll through the extraction results before accessing the file.
This also encourages users to review the extracted data before sharing it.

Key Takeaways from User Testing:

- **Assumptions vs. Reality:**
  The testing highlighted the discrepancy between developers assumptions and actual user behavior.
  Even straightforward features that I took for granted can be misinterpreted or overlooked by users.

- **Importance of Clear Instructions:**
  Clear and concise instructions are crucial, especially for features that require some technical understanding, such as API key input.

- **Old generations vs New generation:**
  I had 2 types of similar and opposite feedback arise between older and younger testers.
  Older users expressed a preference for a dedicated 'Save' button for templates, while younger users often neglected to use a dedicated Save button to save their API keys.
  This suggests the need for a more intuitive save mechanism, potentially incorporating visual feedback or modal reminders to ensure data is properly saved."

Chapter 5

Afterthoughts and Possible Improvements

While I am more than happy with the project's outcome, I recognize several areas where the app could be enhanced or where limitations were encountered due to financial, resource, or time constraints.

5.1 API Keys and Monetization

Requiring users to obtain and manage their own API keys can present a barrier to entry, especially for individuals who lack the technical expertise or familiarity with external platforms.
This approach was chosen due to the lack of support and resources for implementing in-app payments and managing the financial aspects of the app.

If this project were to be developed further with the backing of a company, several ideas for improvement could be explored:

- **In-app Payments:**
  Implementing a subscription or consumption-based payment system to provide access to the app's own API keys.
  This would greatly simplify the user first experience and could generate revenue.

- **Fine tuning:**
  Not relying on user keys would allow DataDig to use GPT Fine Tuning features, creating another layer of security and accuracy to the extraction.
  Fine tuning has been proved to be vital to good data extraction in studies like [4], suggesting that DataDig could probably also benefit from it.

- **Ads:**
  You could show ads during the extraction, to fill in the dead time and to monetize the app even with its current monetization model.

- **In-house AI Solution:**
  Developing a specialized model trained specifically for information retrieval.
  Deep learning-based Image Text Extraction (ITE) solutions have already had mass success in accessing data from text like seen in [15].

  However, developing a high-performing in-house AI solution would necessitate substantial investment in resources and specialized expertise to achieve even slight performance gains, but would surely enlarge the profit window by lowering costs.

  Hybrid parsers approaches could be also looked at, like seen in [6], as they do involve an increase in accuracy, but this would prove to be just as hard.

- **Self-hosted LLM:**
  Deploying an open-source LLM solution like a Llama distribution on a server, creating a custom API and backend.
  This would require careful management of server-side calls to ensure scalability and prevent abuse, but would be significantly easier and could provide comparable performances while cutting costs.

These options, while offering potential benefits, are beyond the scope of this student project due to the financial and logistical constraints associated with monetization and infrastructure management, but could be interesting if the situation changes.

5.2 Multiple Extractions and Rate Limits

While the app's infrastructure could technically support multiple extractions from multiple documents simultaneously, the UI would become way more complex, and extensive modifications would be required throughout the app to handle this feature.
Moreover, multiple extractions could trigger API rate limits, which varies depending on the user's API keys, needing complex workarounds and leading to long waiting times.

Therefore, the focus was shifted to ensuring accurate single extractions, prioritizing quality over quantity, and aligning with the needs of individual users and small businesses rather than large enterprises.

5.3 Prompts and Templates

Prompt engineering plays a crucial role in achieving optimal extraction results from LLMs.
While considerable effort was invested in refining prompts, there is always room for improvement, and further experimentation and fine-tuning could potentially lead to an increase in extraction accuracy and speed.
Similarly, the pre-built templates provided by the app could benefit from more in-depth testing and refinement.
Currently, some templates, like those for bills, are simple translations rather than being tailored to specific document structures of the country that speak that language.

5.4 Language Support and UI Testing

Expanding language support and conducting more comprehensive UI testing for different languages would help with the app's usability and accessibility to a wider audience.
A more expert hand in UI design would also improve the project by making it less bland.

5.5 Alternative implementations considered

Some alternatives were tried or investigated upon are are worthy to mention.

- GPT-4 Vision was tried and tested, but its poor performance in both accuracy and token economy were enough to exclude it.
- Tesseract, TesseractAPI and others OCR engines were considered, but by implicitly using Azure OCR, which has been proved to have an edge against the other options by [1], there was no reason to have heavy engines weight on the app in terms of space and speed.
- You could also move away from Azure Form Recognition SaaS and shift towards a semi-Custom solution like seen in [2].
  This would not allow for the integrated OCR to be used but would greatly decrease costs.
- Different models could be added or substitute the current ones.
  Claude 2 has demonstrated great accuracy in this field as seen in [8], but as I felt GPT was the leading API-accessible LLM in the market, and scared of Claude inability to infer data not explicitly reported in the articles ([16]), I ultimately followed was I was familiar with.

5.6 Tutorials and Instructions

An interactive tutorial could be added to walk the user through the application, by explaining buttons and workflows.
This would help a lot in explaining what the app does and how to use it, contributing in the intuitiveness factor of the application, which constantly feels lacking.

Conclusions

This thesis has presented DataDig, a mobile application designed to address the growing need for efficient and accurate data extraction from documents and images.

By combining OCR technology with the power of LLMs, we saw how DataDig enables users to fully customize the extraction process, from defining templates to choosing models, output formats, and OCR options, all without requiring model training or example documents.

Through testing and user feedback, I hope I gave evidence to DataDig effectiveness in accurately extracting information in text, tables and especially through context.

We looked at the app key strengths:
Flexibility, customizability, and focus on data privacy, and its limitations, long or large-scale extraction and the reliance on user-provided API keys.

I think I accurately demonstrated the app potential to significantly benefit who falls into the market target and niche DataDig can find as a free app.

While I hope DataDig presents a valuable contribution to the field of data extraction, I recognize there are opportunities for further improvement.
Future work could be focused on expanding language support, enhancing the user interface, or with greater resources even change the LLM target to a different solution, to eliminate or lessen the found weaknesses.

Despite the limitations of the app and of my resources making the app, I am proud of the results achieved and confident in its potential, as I feel it demonstrates the potential of AI-powered solutions to transform the way we interact with and process information from documents.

As the volume of digital data continues to grow, tools like DataDig will become increasingly essential and common for individuals and organizations looking to manage and leverage their data.

Appendix A

DataDig App Resources

The DataDig app can be downloaded from the Google Play Store at the following link:

`https://play.google.com/store/apps/details?id=com.friberg.dataDig`

If you want to try the app and give feedback on it, you can leave a review on Google Play or access the DataDig questionnaire in english:

`https://forms.gle/4K9PyRRAfwVbGnTc9`

or you can access the full italian testing questionnaire linked in the next appendix.

For the full flowchart of DataDig, check out:

`https://lucid.app/lucidchart/f9086c20-9cf5-4cb5-b671-9ea3765e6590/edit?viewport_loc=-271%2C205%2C3411%2C1722%2C0_0&invitationId=inv_f72e53fa-ef1f-4688-ba5f-37f03d03d`

If you are a developer trying to work with DataDig, read the README file explaining how to link your workflow to DataDig's, or ask read access to the Repository, all at:

`https://github.com/fri3erg/DataDig`

for any other information, you can contact me directly at:

elia.fri3erg@gmail.com

Appendix B

User Testing Materials

As all of my users were italian, testing material is provided in that language.
The full text of the questionnaire used in the user testing phase can be accessed at the following link:

`https://forms.gle/QWXRdr7n16uAvkJ99`

While these were the task asked to complete:

Task da fare

- **1: Inserire le Chiavi**

  Andate nelle impostazioni e inserite le chiavi date

- **2: Provare la vostra prima estrazione**

  Per la prima estrazione vi consiglio qualcosa di semplice, ad esempio un documento di identità (carta dìidentità, patente, abbonamento dell' autobus ecc) o un biglietto da visita
  Andate nella schermata della fotocamera, fate una foto a quello che volete estrarre, selezionate il modello appropriato e premete Estrai.
  Potete anche uscire dall'app mentre estrae, vi notificherà quando ha finito (generalmente 20-30 sec).
  Vai a vedere se i dati che erano presenti nella foto sono stati estratti.

- **3: Articolo di giornale**

  L'estrazione basandosi su Intelligenza artificiale riesce a capire il contesto di ciò che mandi.
  Fai una o più foto(o screenshots) ad un articolo o ad una notizia.
  Usa il modello: Articolo di Giornale
  Osserva il risultato, è stato estratto bene?

- **4: Tabella**

  Per questa sezione è consigliato usare una bolletta o qualsiasi documento che con una tabella ben visibile.
  Occorre ad esempio fotografare la parte della bolletta che ha tariffe e costi in formato tabellare.
  Vai nei Modelli e clicca sul modello scelto (Bolletta se hai usato una Bolletta)
  Controlla che le colonne e/o righe siano simili a quelle nel tuo documento. A pag 6 troverai in dettaglio come funzionano le tabelle .
  Prova l'estrazione e guarda se è stata avvenuta con successo.

- **5: Ultimo test, modello personalizzato**

  Pensa ad un altro qualsiasi pezzo di carta, pdf, immagine che hai sul dispositivo che contiene testo
  Vai alla sezione Modelli e clicca il + per creare un nuovo modello.
  Inserisci tutti i campi che vuoi estrarre e le eventuali tabelle (pag 6 per come funzionano i template)
  Prova il tuo nuovo modello, l'estrazione è corretta?

- **Evviva, valutiamo il tutto**

  Usa il questionario fornito per valutare queste estrazioni, e prendimi in giro se c'è qualche problema.
  Puoi segnalare qualsiasi problema, bug o errore grammaticale nel questionario (l'app è stata creata in inglese e tradotta in italiano perchè sono un giovane globalizzato, quindi spero le traduzioni siano corrette) Valuta l'app attraverso google play così che la possa inserire sulla piattaforma

Appendix C

Detailed Test Results

This appendix provides a detailed overview of the table extraction testing results. The following table presents the accuracy and extraction time for different table sizes, dynamism levels, and language models.

Results as table (Table Extraction)

Table C.1: Table showing test results for Table testing

Size	Model	Dynamism	Result	Avg time taken	Notes
4 x 6	GPT-3.5 turbo	completely dynamic	24/24	6 seconds	excludes column header
4 x 6	GPT-3.5 turbo	dynamic rows	24/24	7 seconds	no remarks
4 x 6	GPT-3.5 turbo	dynamic columns	24/24	6 seconds	no remarks
4 x 6	GPT-3.5 turbo	non dynamic	24/24	8 seconds	no remarks
4 x 6	GPT-4	completely dynamic	24/24	11 seconds	no remarks
4 x 6	GPT-4	dynamic rows	24/24	12 seconds	no remarks
4 x 6	GPT-4	dynamic columns	24/24	10 seconds	no remarks
4 x 6	GPT-4	non dynamic	24/24	14 seconds	no remarks
8 x 12	GPT-3.5 turbo	completely dynamic	96/96	13 seconds	excludes column header
8 x 12	GPT-3.5 turbo	dynamic rows	96/96	15 seconds	no remarks
8 x 12	GPT-3.5 turbo	dynamic columns	96/96	16 seconds	no remarks
8 x 12	GPT-3.5 turbo	non dynamic	96/96	27 seconds	no remarks
8 x 12	GPT-4	completely dynamic	96/96	22 seconds	no remarks
8 x 12	GPT-4	dynamic rows	96/96	24 seconds	no remarks
8 x 12	GPT-4	dynamic columns	12/96	19 seconds	gave only first column
8 x 12	GPT-4	non dynamic	0/96	83 seconds	parsing error
15 x 15	GPT-3.5 turbo	completely dynamic	225/225	26 seconds	puts column header at the bottom
15 x 15	GPT-3.5 turbo	dynamic rows	225/225	22 seconds	no remarks
15 x 15	GPT-3.5 turbo	dynamic columns	225/225	35 seconds	no remarks
15 x 15	GPT-3.5 turbo	non dynamic	0/225	59 seconds	parsing error
15 x 15	GPT-4	completely dynamic	225/225	62 seconds	excludes headers
15 x 15	GPT-4	dynamic rows	225/225	34 seconds	no remarks
15 x 15	GPT-4	dynamic columns	225/225	81 seconds	no remarks
15 x 15	GPT-4	non dynamic	0/225	93 seconds	parsing error

Bibliography

[1] LI, Yuchen. Synergizing Optical Character Recognition: A Comparative Analysis and Integration of Tesseract, Keras, Paddle, and Azure OCR. 2024.

[2] HIRNER, Bc Erik. Form recognition system architectures.

[3] TÖRNBERG, Petter. How to use llms for text analysis. arXiv preprint arXiv:2307.13106, 2023.

[4] PATINY, Luc; GODIN, Guillaume. Automatic extraction of FAIR data from publications using LLM. 2023.

[5] BISWAS, Anjanava; TALUKDAR, Wrick. Robustness of Structured Data Extraction from In-Plane Rotated Documents Using Multi-Modal Large Language Models (LLM). Journal of Artificial Intelligence Research, 2024.

[6] SEWUNETIE, Walelign Tewabe; KOVÁCS, László. Exploring Sentence Parsing: OpenAI API-Based and Hybrid Parser-Based Approaches. IEEE Access, 2024, 12: 38801-38815.

[7] PENG, Ruoling, et al. Embedding-based retrieval with llm for effective agriculture information extracting from unstructured data. arXiv preprint arXiv:2308.03107, 2023.

[8] GARTLEHNER, Gerald, et al. Data extraction for evidence synthesis using a large language model: A proof-of-concept study. Research Synthesis Methods, 2024.

[9] PETIT, Nicolas. Antitrust and artificial intelligence: a research agenda. Journal of European Competition Law & Practice, 2017, 8.6: 361-362.

[10] YANAMALA, Anil Kumar Yadav; SURYADEVARA, Srikanth; KALLI, Venkata Dinesh Reddy. Balancing Innovation and Privacy: The Intersection of Data Protection and Artificial Intelligence. International Journal of Machine Learning Research in Cybersecurity and Artificial Intelligence, 2024, 15.1: 1-43.

[11] IRIMIA, Cosmin, et al. Official Document Identification and Data Extraction using Templates and OCR. Procedia Computer Science, 2022, 207: 1571-1580.

[12] KAMBLE, Bhavna; NEMADE, Milind; WADHE, Vaishali. Data Extraction Techniques for Data on different Environment: A Review. image, 2019, 8.6: 51-66.

[13] PLOUG, Thomas. The right not to be subjected to AI profiling based on publicly available data—privacy and the exceptionalism of ai profiling. Philosophy & Technology, 2023, 36.1: 14.

[14] CHAWLA, Akhil; GUPTA, Aarushi; SHUSHRUTHA, K. S. Intelligent Information Retrieval: Techniques for Character Recognition and Structured Data Extraction. 2022.

[15] YE, Yibin, et al. A unified scheme of text localization and structured data extraction for joint OCR and data mining. In: 2018 IEEE International Conference on Big Data (Big Data). IEEE, 2018. p. 2373-2382.

[16] LIU, Jiayi, et al. AI-Driven Evidence Synthesis: Data Extraction of Randomized Controlled Trials with Large Language Models.

C.1 Further Readings

[17] NGUYEN, Kha Cong; NGUYEN, Cuong Tuan; NAKAGAWA, Masaki. Nom document digitalization by deep convolution neural networks. Pattern Recognition Letters, 2020, 133: 8-16.

[18] RIZVI, Mehdi, et al. Optical character recognition based intelligent database management system for examination process control. In: 2019 16th International Bhurban Conference on Applied Sciences and Technology (IBCAST). IEEE, 2019. p. 500-507.

[19] ALTO, Valentina. Modern Generative AI with ChatGPT and OpenAI Models: Leverage the capabilities of OpenAI's LLM for productivity and innovation with GPT3 and GPT4. 2023.

[20] GHANAWI, Ibrahim, et al. Integrating AI with MBSE for Data Extraction from Medical Standards. In: INCOSE International Symposium. 2024. p. 1354-1366.

[21] OSENI, Ayodeji, et al. Security and privacy for artificial intelligence: Opportunities and challenges. arXiv preprint arXiv:2102.04661, 2021.

[22] PACHPUTE, Bhushan B.; TANDALE, Suraj C. TIME MANAGEMENT BY DIGITALIZATION OF DOCUMENTS. 2022.

Acknowledgements

I want most and foremost thank my tutor, professor Montori, for following me closely for this thesis and being an inspiration.
For supporting me along my studies I also want to thank my family, my girlfriend, my friends and of course my dog.

And I want to thank all my testers, who found time out of their busy day to lose on thorough testing and answering questionnaires, from which personally I want to thank:

Valentina
Matteo
Maurizio
Michela
Alessandro
Anna
Margherita
Petter
Daniele
Jacopo M.
Giovanni
Mirco
Francesca
Nicola
Marco (MP)
Andrea (AC)
Sabkat
Piero
Niccolò
Mehdi
Simone
Jacopo G.
Carmen
Valerio
Marta
Elena