LLM PatentPete Documentation Group 2

Timucin Cicek, Matthias Meinke, Tim Messerschmidt, Tristan Ruhm

Documentation

Disclaimer

> ❗ Important
>
> To be able to start the application without errors, you need to update your .env please, with the following attributes:
>
> ```
> OPENAI_API_KEY = 'OWN SECRET KEY'
>
> GOOGLE_PATENT_API_KEY =
> '7661f86900cb42b2782c2af5c95142574b959dd5546c98491652475ea91e278e'
>
> DATABASE_URI =
> 'mongodb+srv://timmey:faB8MFdyyb7zWvVr@llm-ttt.8kqrnka.mongodb.net/?retryWrites=true&w=majority'
> ```

1. Concept

1.1 Introduction to Patent Pete

Patent research can be a critical and time-consuming process for inventors, researchers, and patent professionals. It involves not only identifying relevant patents, but also understanding the intricate connections and similarities between them. Patent Pete is designed to address these challenges by leveraging advanced technology to make patent research more accessible, efficient, and insightful.

Patent Pete automates the search and analysis process for patent discovery, making it ideal for inventors, research and development teams, patent professionals, and academics. It helps identify relevant patents, avoid infringement, guide innovation, conduct prior art searches, and

explore technological trends. Patent Pete is an essential tool for anyone involved in patent research or development.

1.2 Concept Overview

Patent Pete is based on a Retrievable Augmented Generation (RAG) model. It integrates OpenAI's large language model capabilities with advanced search and similarity comparison techniques. This application simplifies patent research by allowing users to enter a patent document. It then conducts a thorough search for related patents and provides the user the links to them for further research.

1.3 Concept of Patent Classifications

Patent classifications are used to organize patents into categories to make them easier to search, compare and understand. They enable patent offices, inventors, companies and researchers to find relevant patents and obtain information about the state of the art in specific fields. Here are some reasons why patent classifications are important:

Organization and structuring: With thousands of patents filed every year, patent classifications help to create order and divide patents into logical categories.

Searching and monitoring: Classification allows inventors and companies to monitor patents in areas relevant to their work and identify new developments.

Avoid duplicates: Classifications help patent examiners to identify similar inventions and avoid duplication of work.

International harmonization: Patent classification systems allow different countries to harmonize their patent data internationally, facilitating collaboration and information exchange.

There are various patent classification systems that have been developed by different countries and organizations. The best known are:

- The International Patent Classification (IPC): This was developed by the World Intellectual Property Organization (WIPO) and is used by most countries in the world. The IPC categorizes inventions into thematic groups and subclasses according to technical characteristics. It is very simple to use and easy to see how it works, as shown in Figure 1.

- Common Patent Classification (CPC): A common classification system developed by the patent offices of the United States (USPTO) and Europe (EPO). The CPC is based on the IPC, but has been adapted to the needs of the USA and Europe.

- US Patent Classification (USPC): The classification system used in the United States that categorizes patents into different classes.
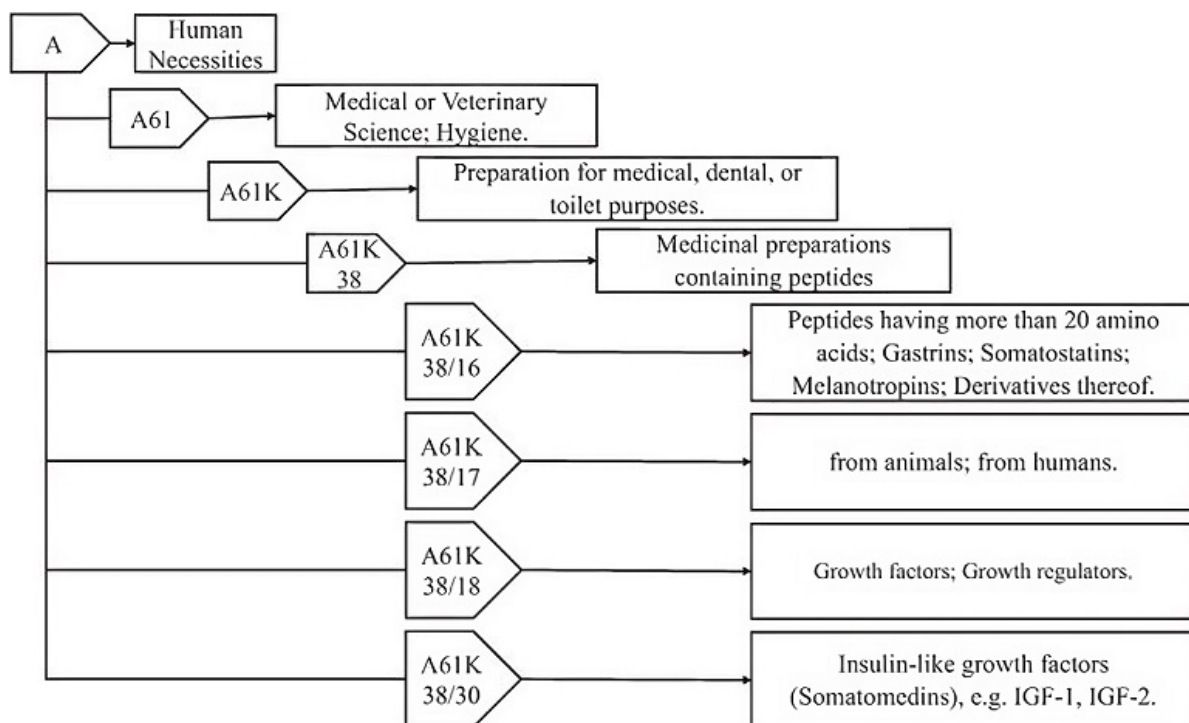
Figure 1: IPC Classification Hierarchy

1.4 How It Works

The user submits a patent document to Patent Pete to begin the process. Patent Pete employs a language model to analyze the abstract and generate relevant keywords and patent classifications. These elements form the basis for a targeted search in the Google Patents database, ensuring the relevance of the results for the user's initial query.

After retrieving potential matches, Patent Pete continues the search. It enhances search results by embedding them into a vector database. This enables a similarity search that pinpoints the top 10 results based on their relevance and likeness to the submitted abstract. The top contenders undergo a second similarity comparison by the LLM, ensuring thorough evaluation of each patent's nuances and specificities.

Patent Pete presents the user with the five best matching patents. These patents are the most relevant and similar to the user's initial submission, making the research process efficient and effective.

1.5 Goals

The goal of Patent Pete is not to replace traditional patent research methods completely but to provide users with a more accessible and efficient starting point for their patent exploration journey and saving them a lot of time. By automating the initial stages of research and analysis, Patent Pete aims to streamline the process, allowing users to quickly identify relevant patents and focus their efforts on deeper investigation where human expertise and judgment are irreplaceable.

2. User Interface

At the beginning, we decided to create our user interface with the streamlit framework. After some research and a recommendation from our lecturer, we switched to Gradio, as this framework is designed more for AI-based apps.

The user interface was originally only supposed to consist of an input and output field. The input field was a text field in which the patent idea could be entered or the abstract copied in. After receiving feedback from our project partner, we decided to implement an upload field instead to make the input more user-friendly and enable greater accessibility during use.

```
# Loading PDF file from user input and extracting first page containing the abstract
def input_analysis(file, progress=gr.Progress()):
    if file is not None:
        #Reading PDF file
        progress(0.2, desc="Processing file")
        pdf_reader = PyPDF2.PdfReader(file)
```

```
        # Extracting content
        content = pdf_reader.pages[0].extract_text()
        if content == "":
            raise gr.Error("The file seems to be invalid. Please check your file or \
                try another Pdf!")
        progress(0.3, desc="Analyzing file")
    return content
```

```
files= gr.File(file_types=['.pdf'], label="Upload your pdf here.")
```

The output field should contain the result of the research, including the necessary information about the patents, the user needs. First, we decided to give out the matching score of each patent with the input and the link to related PDF. Realizing, that the score would be irrelevant for the user or even confuse them, we changed the results. Now the result contains the title, the PDF link and number of the position of the ranking.

```
# Formatting final scoring results for user output
final_scoring_formatted = "Ergebnis:\n\n"
counter=1
for patent_id in final_scoring_patent_ids:
    final_scoring_formatted += "#" + str(counter) + ": "
    + patent_data[patent_id]["title"]
    + "\n" + "https://patentimages.storage.googleapis.com/"
    + patent_data[patent_id]["pdf"] + "\n\n"
    counter+=1

return final_scoring_formatted
```

```
#New Value "Top 5 PDFs ...."
endresult = gr.Textbox(label="End Result", value="None", show_copy_button=True)
```

In addition, we added a field for possible configurations for the patent search below the upload field. These should make it possible to select search elements such as classification type, number of keywords/classifications or the database to be used as desired. However, the options to choose another patent database or another type of classification only serves as a mockup in our final application. We just wanted to show which more possibilities the application could have.

```
# gradio interface for our configurations
with gr.Column() as main:
    gr.Markdown("<p><h2 style='color:#2563EB'>Input</h2></p>")

    files= gr.File(file_types=['.pdf'], label="Upload your pdf here.")
```

```
    gr.Markdown("<u>Configuration Options</u>")

    gr.Radio(["International Patent Classification (IPC)",
    "United States Patent and Trademark Office (USPTO)",
    "Cooperative Patent Classification (CPC)",
    "Deutsche Klassifizierung (DEKLA)"], label="Type of Classification",
    value="Cooperative Patent Classification (CPC)"),

    slide_keywords = gr.Slider(1, 5, step=1, value=2, label="Number of Key Words")
    slide_classes = gr.Slider(1, 5, step=1, value=2, label="Number of Classifications")

    gr.CheckboxGroup(["Google Patents", "Espacenet", "European Patent Office (EPO)",
    "DEPATISnet"],
    label="Databases",
    info="Which databases should be searched?",
    value="Google Patents"),

    button = gr.Button("Submit")
```

During development, we have built in a detailed output to be able to call up the respective steps of the program. This was to indicate, for example, whether a retrieval with Google Patents or a connection to the vector database had taken place. However, this greatly disfigured the UI, and we decided to put the details in a pop-up window. Ultimately, we came to the realization that this information was trivial for the user and removed it completely.

As a bonus, we wanted to integrate a button with which the results can be formatted and downloaded in a PDF. However, we were confronted with some technical problems, particularly the transfer of the results to the PDF, so that it did not make it into the final version due to time constraints. As an alternative, we have implemented a copy button in the output window so that the results can at least be copied.

3. Development

3.1 Basics

To realize our previous plans, we took various steps during our project. Firstly, we devised a series of sample patents and generated corresponding abstracts. The aim was to obtain a wide variety of abstracts to enable a comprehensive analysis. These abstracts were carefully formulated and saved as PDFs.

```
# Loading PDF file from user input and extracting first page containing the abstract
def input_analysis(file, progress=gr.Progress()):
    if file is not None:
        #Reading PDF file
```

```
        progress(0.2, desc="Processing file")
        pdf_reader = PyPDF2.PdfReader(file)
        # Extracting content
        content = pdf_reader.pages[0].extract_text()
        if content == "":
            raise gr.Error("The file seems to be invalid. Please check your \
                file or try another Pdf!")
        progress(0.3, desc="Analyzing file")
    return content
```

3.2 Generating Key Words and Classifications

In the planning phase, we focussed on developing efficient LLM prompts to generate keywords and classifications based on the sample abstracts. For this purpose, we created various sample abstracts and recorded them in an Excel spreadsheet. By defining and evaluating different prompts, we were able to select the one that provided the most search results to enable a comprehensive search.

```
# LLM Prompt generating Key Words on base of the PDF Input from User
def output_keywords(content, n, progress=gr.Progress()):
    progress(0, desc="Generating Key Words...")
    keyword_prompt = f"""
    The following abstract descripes a concept for a novel invention:\
    ```{content}```\
    Name {n} key words based on this abstract, that I can use for the search in a \
    patent database. Optimize the key words to get back more results. Result \
    as python string.
    """

    response_keywords = get_completion(keyword_prompt)

    return response_keywords
```

During the development of the project, concerns were raised by ZOI about the ability of LLM to correctly generate classifications based on the abstracts. For this reason, we decided to generate five classifications, both as International Patent Classification (IPC) and Cooperative Patent Classification (CPC) as well as in US Patent and Trademark Office (USPTO) classification.

```
# LLM Prompt generating Classifications on base of the PDF Input from User
def output_classes(content, n, progress=gr.Progress()):
    #Progress Bar from Gradio to visualize the current progress
  progress(0, desc="Generating Classifications...")
  classes_prompt = f"""
      The following abstract descripes a concept for a novel invention:\
```

```
    ```{content}```\
    Name {n} CPC classifications based on this abstract, that I can use for
    the search in a patent database. \
    Please give me a python string for the codes of the {n} most relevant \
    CPC classifications to a possible patent.
    """

response_classes = get_completion(classes_prompt)

return response_classes
```

After manually comparing the classifications with the official registers, we found that the LLM coped well with the IPC/CPC classifications in particular. However, we encountered difficulties when it came to the USPTO subclasses. The LLM did not seem to be able to correctly recognize the subtleties of the subclasses and assign them accordingly.

3.3 Google Patents API

Choosing the right patent databases proved to be more difficult than originally assumed. There are a variety of patent databases, each serving a specific purpose in the areas of research, innovation and intellectual property protection. These databases play a crucial role in identifying relevant patents, researching prior art and monitoring potential patent infringements.

Access to a comprehensive and up-to-date database is essential for an effective patent search. However, the patent database landscape is fragmented as many databases are operated by official authorities such as the German Patent and Trade Mark Office (DPMA) or the European Patent Organisation (EPO), each with their own API. This fragmentation leads to different logic and a limited amount of data, which makes detailed searches difficult.

When selecting a suitable patent database, we looked at various databases in advance and identified their strengths:

Espacenet: This database is particularly suitable for "state of the art" searches. It provides access to an extensive collection of patent documents from all over the world and enables users to research the current state of development in various fields of technology.

DPMA Register: With its register, the German Patent and Trade Mark Office offers a comprehensive database for IP searches and patent monitoring. Here, users can access information on German patents, trade marks and designs and track relevant legal events.

European Patent Register (EPO): The European Patent Register is a valuable resource for searching for European patents. It provides access to patent documents from various European countries and facilitates the search for European intellectual property rights.

World Intellectual Property Organisation (WIPO): The WIPO database provides a global overview of international patent information. Users can search for patents from various countries and retrieve information on international property rights.

Octimine: Octimine provides a complete overview of patent data from the EU, the USA and WIPO. The database enables users to search patents and find relevant information for their research and development projects.

DEPATISnet: With over 60 million patent documents, DEPATISnet is one of the most comprehensive patent databases in the world. It provides access to a large number of patents from different countries, however some documents may not be translated, which can limit their use.

Google Advanced Patent Search: Google Advanced Patent Search database provides a good overview of the international patent market. The metadata of the patent information is organised in a structured manner. It is particularly noteworthy that Google Patents utilises current CPC classifications and can therefore provide ideal opportunities for a more targeted overall overview.

Google Patents seemed to us to be the ideal place to start. On the one hand, this database allowed us to partially circumvent the above-mentioned problems of the various authorities and summarize them under a uniform logic. Although Google Patents does not have an official API, we used a third-party provider, SerpAPI, to perform searches. SerpAPI does provide useful data in JSON format, but its free plan limits queries to 100 per month and accommodates up to 1,000 patents with 10 results per query.

```python
# initializing base vars for the following loop
progress(0.5, desc="Researching patents")
#load secret key for the google patent api
patent_api_key = os.environ['GOOGLE_PATENT_API_KEY']
count = 0
#just to complete the url
patent_base_url = "https://patentimages.storage.googleapis.com/"
patent_data = {}

# Loop for multiple Google Patents API calls with Key Words
for i in keywords_list:
    openai_response = i #Search String for Google Patents
    url_base = "https://serpapi.com/search.html?engine=google_patents"
    query = openai_response.replace(" ", "+")
    url = url_base + "&q=" + query + "&language=ENGLISH&api_key=" + patent_api_key

    # API call Google Patents
    response = requests.get(url)

    # Check if API call was successful
    if response.status_code == 200:
```

```
        data = response.json() #write json-answer in var
        for cluster in data["results"]["cluster"]:
            for result in cluster["result"]:
                id = result["patent"]["publication_number"]
                if id not in patent_data.keys():
                    patent_data[id] = {
                        "pdf": result["patent"]["pdf"],
                    }
    else:
        print(f"Error with API request: Status code {response.status_code}")

# Parsing API answer .json and adding the patent ID to the
# patent_data dict, avoiding duplicates
for cluster in data["results"]["cluster"]:
    for result in cluster["result"]:
        patent_id = result["patent"]["publication_number"]
        if patent_id not in patent_data.keys():
            patent_data[patent_id] = {
                "pdf": result["patent"]["pdf"],
            }
```

However, our reliance on the JSON file from Google Patents API via SerpAPI revealed limitations during LLM-based similarity checks. This JSON file has a character limit for abstracts, omitting claims and descriptions entirely. This constraint stems from SerpAPI's output mirroring the structure of Google Patents search results, storing only partial abstracts displayed in previews.

To overcome this obstacle, we implemented a web scraping function. This function utilizes patent IDs from SerpAPI's JSON to sequentially retrieve individual patent pages, enabling us to scrape complete abstracts, descriptions, and claims for thorough analysis.

3.4 Abstract Extraction

To optimize the extraction of the abstracts from the patents, we decided to save the results in a JSON file. This enabled efficient utilisation of the information and also served as a mini-database replacement. We then developed an LLM prompt to compare the saved abstracts with a sample abstract and output the match percentage. Although this approach was promising, we realised that it was slow due to the large amount of data and exceeded OpenAI's token limit.

3.5 Vector Database

After realizing that the previous method of comparing the abstracts to the LLM led to performance issues, we looked closely at alternative solutions. One promising option was the use of

vector databases. We carried out extensive research to find the most suitable solution for our project.

Our first approach was to access the DocumentDB via AWS. Unfortunately, this attempt proved unsuccessful, as we encountered unexplained access restrictions that also affected the DocumentDB. This meant that we had to look for another solution to fulfil our requirements.

In the end, we decided on MongoDB ATLAS as an alternative. This service also utilizes the AWS infrastructure, which allowed for seamless integration into our existing setup. We set up a vector database in MongoDB ATLAS and developed an API to access this database.

The use of MongoDB ATLAS proved to be extremely successful. The API enabled us to insert, retrieve and delete new content from the vector database and, most importantly, to efficiently compare the stored abstracts to obtain the desired similarity results. In addition, we were able to successfully solve performance issues that we previously had with the LLM. The results from the vector database were more accurate and reliable than the previously used LLM method.

```
database = get_database_connection()
# Clearing db before adding new data, to avoid any distortion of results
clear_db(database)
ATLAS_VECTOR_SEARCH_INDEX_NAME = "vector_index" #intitalize vector similarity search

# insert the documents in MongoDB Atlas with their embedding
vector_search = MongoDBAtlasVectorSearch.from_documents(
    documents=patent_list,
    embedding=OpenAIEmbeddings(disallowed_special=()), #use openAI embeddings method
    collection=database,
    index_name=ATLAS_VECTOR_SEARCH_INDEX_NAME,
)

# waiting for the db to save
time.sleep(5)

# Performing a similarity search with Score between the embedding of the query
# and the embeddings of the documents
progress(0.9, desc="Compare the patents")
query = str(content)

results = vector_search.similarity_search_with_score(
    query=query,
    k=10, #Output for the top 10 results
)
```

By implementing the vector database, further performance optimizations were achieved, as the processing of the extracted abstracts was also improved. The key benefit was that we no longer had to rely on creating and reading the JSON file. Instead, we were able to write the

results directly to the vector database. This led to a significant increase in performance, as the data could be accessed faster and more efficiently.

Although we decided to use the vector database as the main method, we still retained the evaluation via the LLM as an additional quality feature. We re-evaluated the best 10 results from the vector database using the LLM method to ensure that the quality of the results continues to meet our standards.

```python
# Formatting vector search result for further usage
vector_scoring = {}
for result in results:
    vector_scoring[result[0].metadata['patent_id']] = result[1]

# Building LLM similarity scoring prompt
comparison_prompt = f"""The following texts are abstracts from patent specifications.
Your task is to compare the "Testing Abstract" to all the others. It is important
that you focus on comparing the concepts that the abstracts describe, not the way
they are written. Rank the remaining abstracts on how well they match with the
Testing Abstract by giving them a rating from 0 to 10 points. 0 meaning they have absolutely nothing
your output string to an actual dictionary, so make sure the formatting is right.

Testing Abstract: "{content}"
"""

# Adding patent abstracts to the prompt
for patent_id in vector_scoring.keys():
    # Check if there is an abstract for the patent
    if patent_id in patent_data and patent_data[patent_id]["abstract"]:
        comparison_prompt += f'{patent_id}: "{patent_data[patent_id]["abstract"]}"\n'

response = get_completion(comparison_prompt)

# Formatting LLM output
llm_scoring_raw = eval(response.replace("comparison = ",""))

# Calculating final scoring results (combining vector scoring with llm scoring)
def transform_ratings(ratings, new_min=0, new_max=10):
    # Determine the smallest and largest value in the original dictionary
    old_min, old_max = min(ratings.values()), max(ratings.values())
    transformed_ratings = {}
    for key, value in ratings.items():
        # Apply the transformation with dynamic old and new ranges
        transformed_value = ((value - old_min) / (old_max - old_min))
        * (new_max - new_min) + new_min
        transformed_ratings[key] = transformed_value
    return transformed_ratings
```

3.6 webscraping

Although the contents of the API provide initial information on relevant patents, they are not yet detailed enough to make a well-founded statement as to whether a patent is considered to be relevant or not. Important information such as the claims of the patents or the patent text were not included in the search results, and the abstract was often incomplete as it was cut off after a certain point.

Initially, we tried to extract this information from the PDFs of the individual patents. However, we found that this solution not only took an incredibly long time, as a large amount of data had to be loaded, but also delivered rather inaccurate results. The contents of the patent specifications could often not be accurately extracted and were also not always written in English, but rather in Chinese or Korean, which presented a major challenge.

We therefore decided to retrieve the detailed patent data directly from Google Patents via web scraping using BeautifulSoup. This gave us access to the complete title, abstract, the entire patent specification and the claims. This had several advantages: Firstly, we were able to significantly shorten the runtime in that we no longer had to download large amounts of data from PDFs and extract them in a time-consuming step, but were able to process plain text instead. However, what was even more important was that we now had access to patent texts that were translated directly into English by Google, which not only made the following content analysis much easier, but actually made it possible in the first place.

While the scraping of the search results page was blocked by Google after a few requests, we had no problems accessing the individual patent pages directly. This allowed us to integrate this solution into our final application.

```python
# Scraping complete patent data (title, abstract, description, claims)
progress(0.6, desc="Collecting patent data")
for patent_id in patent_data.keys():

    # generating Google Patent links for each ID
    url = "https://patents.google.com/patent/" + patent_id + "/en"

    response = requests.get(url)
    html_content = response.content
    soup = bs(html_content, 'html.parser')

    # Scraping Title
    title_span = soup.find('span', itemprop='title')
    if title_span is not None:
        title = title_span.get_text()

        # Removing weird ending of title
        to_remove = "\n"
        title = title.replace(to_remove, "").strip()
```

```python
else:
    title = False

# Scraping Abstract
abstract_div = soup.find('div', class_='abstract')
if abstract_div is not None:
    abstract = abstract_div.get_text()
else:
    abstract = False

# Scraping Description
description_section = soup.find('section', itemprop='description')
if description_section:

    # Removing H2 from section
    h2_tag = description_section.find('h2')
    if h2_tag:
        h2_tag.decompose()

    # Removing all 'notranslate' class items
    for notranslate_tag in description_section.find_all(class_='notranslate'):
        notranslate_tag.decompose()

    # Removing all <aside> elements
    for aside_tag in description_section.find_all('aside'):
        aside_tag.decompose()

    # Extracting and joining the text
    description = "".join(description_section.stripped_strings)
    if description == "":
        description = False
else:
    description = False

# Scraping Claims
description_section = soup.find('section', itemprop='claims')
if description_section:
    # Removing H2 from section
    h2_tag = description_section.find('h2')
    if h2_tag:
        h2_tag.decompose()

    # Removing all 'notranslate' class items
    for notranslate_tag in description_section.find_all(class_='notranslate'):
        notranslate_tag.decompose()

    # Removing all <aside> elements
    for aside_tag in description_section.find_all('aside'):
```

```
        aside_tag.decompose()

    # Extracting and joining the text
    claims = "".join(description_section.stripped_strings)
    if claims == "":
        claims = False
else:
    claims = False

# Adding scraped content do patent_data dict
patent_data[patent_id].update({
    "title": title,
    "abstract": abstract,
    "description": description,
    "claims": claims
})
```

Another problem that could be solved with this is the long runtime in the step before we load the patents into the vector database. The original plan was to directly open and read in the patent PDFs so that the similarity search in the patent database would be based on as much information as possible. Thanks to the extensive web scraping of the patents, we were able to simply use this data for embedding in the vector database and thus avoid the time-consuming reading of the PDFs.

3.7 Data Structure

With the implementation of our web scraping feature, we've also established a more efficient and organized framework for handling patent information, significantly enhancing our internal data management system. Previously, we had relied on the incomplete JSON of the API, while the PDF content was processed separately and not linked to the original data. This meant that if we needed to reuse an abstract, for example, we would have needed to redo the time-consuming process of reloading and re-scanning the corresponding PDF document. This procedure was far from ideal, especially from a performance point of view, as it would have led to unnecessary delays.

Therefore, we've transitioned to a new, more practical data structure. It uses a dictionary in which a new entry is created for each patent ID. Within each of these entries, we store key elements of the patent information, including the title, abstract, full text of the patent, claims, and a direct link to the PDF document.

This enables us to retrieve required patent information quickly and easily at various points in the code as often as needed. In addition, by implementing a system where new patents are only scraped and added if their patent ID does not already exist in the dictionary, we eliminate

the risk of duplicating entries. This not only ensures the quality and reliability of our data but also optimizes our storage and processing resources.

4. Evaluation

4.1 Evaluation

When choosing between LlamaIndex and TruLens, we decided in favour of TruLens. This framework offers a wide range of methods that seemed optimal for our use case in order to evaluate and improve the performance of our application. The following methods were considered:

Model Selection: With this method, TruLens selects the most powerful and efficient model for our application. By comparing different models, we can identify those that deliver the best results and best fulfil our requirements.

Detect and Mitigate Hallucination: The RAG Triad is used to ensure that our LLM only reacts to the information that comes from the uploaded abstract. This is to ensure that the model does not tend to generate false or misleading information.

Improve Retrieval Quality: The aim of this method is to improve the quality of information retrieval for our RAG. Through measurement and analysis, we identify opportunities to improve information retrieval and ensure that our results are accurate and relevant. The aim of this method is to see any hallucinations.

Verify the Summarisation Quality: This checks that the abstracts generated by the LLM contain the key points of the uploaded abstract. The idea behind this was to ensure the quality of the key words and, if applicable, the classifications.

Embeddings Distance: This method measures the similarity or dissimilarity of texts by representing them as points in a mathematical space. The most common measurement methods for this are: Cosine distance, Manhattan distance and Euclidean distance. The aim was to evaluate and assess the quality of the results of the similarity search that were output by the vector database at the end.

Due to time constraints, only the retrieval quality method was used as seen in Figure 2. Five sample patents were generated and run through the system with their respective abstracts in order to obtain meaningful results at the end.

Although only one evaluation method was implemented, it still provides valuable insights into the performance of the system and its results can already be used as a starting point for future developments and improvements.

Using the retrieval quality method, we evaluated three points in our app:

1. Generation of classifications based on the abstract uploaded by the user

RAG Classifications ⊙

Records	Average Latency (Seconds)	Total Cost (USD)	Total Tokens	Context Relevance	Groundedness	Answer Relevance
5	3.6	$0	0	0.58 🔴 low	0.2 🔴 low	1.0 ✅ high

RAG Key Words ⊙

Records	Average Latency (Seconds)	Total Cost (USD)	Total Tokens	Context Relevance	Groundedness	Answer Relevance
5	3.6	$0	0	0.86 ✅ high	1.0 ✅ high	1.0 ✅ high

RAG Patent Comparison ⊙

Records	Average Latency (Seconds)	Total Cost (USD)	Total Tokens	Context Relevance	Groundedness	Answer Relevance
5	3.6	$0	0	0.66 ⚠ medium	0.93 ✅ high	0.72 ⚠ medium

Figure 2: RAG Results

2. Generation of key words based on the abstract uploaded by the user

3. The LLM evaluation of five randomly generated abstracts based on the abstract uploaded by the user

The evluation showed different results. Regarding the generation of classifications, as can be seen in Figure 3, there was a low score in terms of groundness, suggesting that the evaluation method assumes that the LLM had difficulties in providing sound classifications and may be hallucinating. However, as mentioned at the beginning of this documentation (see chapter Basics), the group had already checked the classifications manually in advance against the official register and found them to be correct. This indicates that the quality of the classifications is indeed acceptable despite the low rating.

App ID	Time Stamp ↓	Answer Relevance	Context Relevance	Groundedness
RAG Classifications	2024-02-04T22:09:30.778443	1	0.8	0.6
RAG Classifications	2024-02-04T22:08:48.101700	1	0.7	0
RAG Classifications	2024-02-04T22:08:18.969327	1	0.5	0
RAG Classifications	2024-02-04T22:07:48.448497	1	0.4	0
RAG Classifications	2024-02-04T22:07:17.309890	1	0.5	0.42

Figure 3: RAG Details Classifications

The results of the key word generation were satisfactory. In the detailed view in Figure 4, it can be seen that the RAG does not consistently score 100% for Context Relevance. This could be due to the fact that the LLM has problems prioritising the most important key words in view of the abundance of abstracts. Overall, however, the result is satisfactory.

In contrast, the patent comparison method produced unfavourable results. The low scores suggest that the LLM may not have adequately captured the context of the original abstract

App ID	Time Stamp ↓	Answer Relevance	Context Relevance	Groundedness
RAG Key Words	2024-02-04T22:09:25.720998	1	0.8	1
RAG Key Words	2024-02-04T22:08:43.403758	1	0.8	1
RAG Key Words	2024-02-04T22:08:14.877875	1	0.8	1
RAG Key Words	2024-02-04T22:07:43.396142	1	0.9	1
RAG Key Words	2024-02-04T22:07:12.788335	1	1	1

Figure 4: RAG Details Key Words

and therefore does not provide accurate answers. This suggests that although the LLM evaluation can be considered helpful, it should be used with caution as it may not have the desired reliability. However, it should also be noted, as can be seen in Figure 5, that there was only one outlier that negatively affected the values.

App ID	Time Stamp ↓	Answer Relevance	Context Relevance	Groundedness
RAG Patent Comparison	2024-02-04T22:09:37.583806	1	1	1
RAG Patent Comparison	2024-02-04T22:08:58.197932	1	0.7	1
RAG Patent Comparison	2024-02-04T22:08:32.950330	0	0.2	0.6666666667
RAG Patent Comparison	2024-02-04T22:08:03.420618	0.7	0.6	1
RAG Patent Comparison	2024-02-04T22:07:27.289077	0.9	0.8	1

Figure 5: RAG Details Comparison

These results highlight the importance of carefully analysing and interpreting the evaluation results. Although the automated evaluation methods can provide insights, it is important to critically scrutinise them and perform manual checks where necessary to validate the quality of the results and ensure that the system meets the requirements and expectations.

5. Issues

5.1 Abstracts in other Languages or Fonts

When developing the application, we focused on analyzing English-language abstracts and patents. In principle, OpenAi has no problems automatically translating and analyzing texts with Latin fonts in our application. German and French patents could therefore be translated relatively easily. Nevertheless, it can be assumed that there may be translation problems with some patents, especially those in Slavic or other.

In addition, after several test runs with different patents, we found that texts with other fonts, including Chinese and Cyrillic, could not be extracted correctly, and therefore no abstract was available, which led to collapses during the process run. Especially the Arabic font can confuse the LLM, since it also contains a few English words in Latin font. In this case it only took the

English words to generate keywords and classifications, so that in the end the quality of these has been too bad, to get a proper output of similar patents.

We tried to minimize these issues by implementing an examination, which checks, if there has been extracted, an abstract or text. On this way, it was possible to pretend to get no abstract. Nevertheless, the problem of the translation or text structure could still come up, when the abstract is extracted and analyzed.

5.2 Extract Abstracts from PDFs

When extracting the abstracts from the results of the Google Patent API, we encountered various problems that we had to solve during the project. Normally PDFs are formatted the way you can see in Figure 6.

Figure 6: Correct formatted PDF

But a particularly common problem was that about one in ten PDFs was not formatted correctly in a machine-readable way: In some cases, it may be difficult for text extraction tools to recognize and select the text properly because of the structure of the PDF itself. This can happen if the PDF is poorly formatted or if the text is embedded in a way that makes it difficult to separate from other elements on the page.

This can be seen in Figure 7 if you simply try to select the corresponding text block of the abstract with the mouse courser in order to copy it, for example. The cursor jumpes to the left column of the page after the first line of the abstract, which led to a jumble of the read string. Here is an example of an incorrect formatted patent abstract. To solve this problem, the LLM was used again to reconstruct the text and remove interfering or incorrect words. The text was then copied manually from the PDF and compared with the reconstructed text

using the Levenshtein method. Agreements of regularly over 95% confirmed the accuracy of the LLM reconstruction.

Figure 7: Incorrect formatted PDF

5.3 Abstracts Not Always Found on the First Page of Patents

In order to reduce processing time and optimise the performance of our system, we decided to only scan the first page of each PDF. This decision was based on the fact that the abstract is typically found on the first page. By only analysing this first page, we were able to reduce the workload and increase efficiency.

However, we found that in about one in fifty PDFs, the abstract was not positioned on the first page as expected. Instead, it was randomly located on later pages, such as the fifth or even thirteenth page. This posed a challenge, as our assumption that the abstract should always be on the first page proved to be insufficient. However, since this problem was rare and our time was limited, we decided to put it on the back burner for the time being and focus on more pressing issues. Our priority was to ensure the main functionality of our system and to make sure that it worked flawlessly and efficiently. Unfortunately, in the end there was not enough time to solve this specific problem.

5.4 Not Maschine-Readable

Furthermore, about every fiftieth PDF was also not machine-readable. There are several reasons why some PDFs may not be selectable, meaning that the text cannot be easily copied or extracted from the document:

Scanned documents: When a PDF is created from scanned images rather than digital text, the text in the document is essentially an image and is not selectable. This is because the text is not recognized by the computer as text, but as pixels forming an image.

Document security: Some PDF files are created with security settings that prevent text from being extracted or copied. This is often done to protect sensitive information in the document. Text encoding: If the text in the PDF is encoded in a way that is not compatible with text extraction tools, it may not be selectable. This can happen with certain fonts or character encodings that are not recognized by the software trying to extract the text.

We were unable to solve this problem and find a solution due to lack of time constraints.

6. Further Possible Improvements

We concluded that the application still needs a few improvements. First and foremost is the performance. Basically, if you look at the output of the results via the application in relation to the general patent search, you can see a drastic improvement in the duration of the search. While a classic patent search can take several hours or even days, the application outputs the results within 2 minutes. Nevertheless, some performance improvements could be made to provide a better user experience.

For example, the prompts could be improved, especially the prompt for the classifications, which, as explained above, did not perform well in the evaluation. Giving some examples or a more detailed explanation could result in better performance. It would also be possible to combine the prompts for the keywords and the classifications into one prompt. Another improvement for the performance would be to collect the patents via other API providers or databases. Improvements can also be made in the content analysis of patents. An extended analysis of the patents based on the metadata (year of publication, period of validity, place of publication, etc.) or more configurations for the patent research could make the analysis more detailed and thus make it easier to check whether there is a match.

We also would intend to use another framework for our UI, since a good tool to encode a basic Interface for applications with AI, but it's not good for customize the website and creating a user-friendly interface.