NLP PROJECT REPORT 2024

Presented by
Kobe Schoeters
Thorsten Ooms

1. Project Information

We created a chatbot using the Groq API to leverage a large language model (LLM) and integrated it with CrewAI to manage various tasks through specialized agents. The chatbot has several key features designed to make it versatile and user-friendly.

One of the main functionalities is a question-answering agent. This allows users to ask any question, and the chatbot will search the internet in real-time to retrieve accurate and up-to-date answers. Another feature is the cheat sheet agent, which is especially useful for working with documents. Users can upload a PDF, and the chatbot generates a concise, one-page summary of the content, providing quick insights without needing to read the entire document.

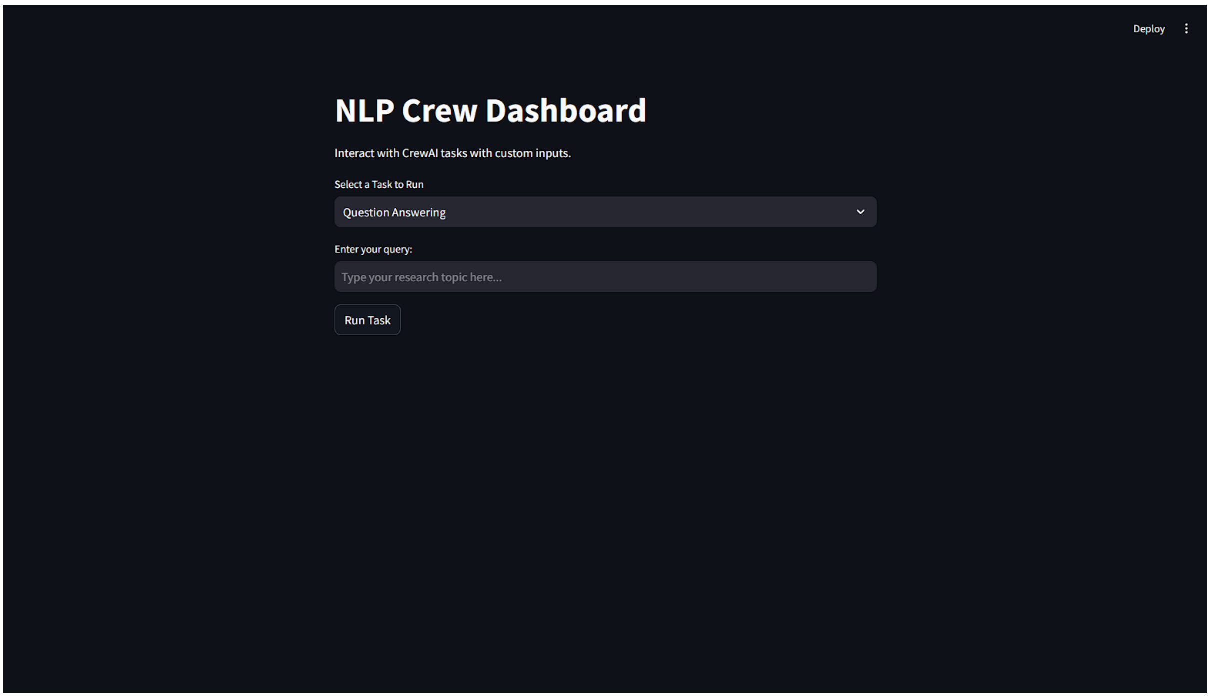
Additionally, we built a system for managing and storing PDFs. Uploaded documents are processed and stored in ChromaDB, a vector database that allows for efficient searching and retrieval. This makes it easy for users to organize their files and access specific information later.

For the frontend, we used Streamlit to create a simple and interactive interface. Users can ask questions, upload files, and interact with the chatbot easily, even if they have no technical background. The combination of Groq API for LLM capabilities, CrewAI for task management, and ChromaDB for storage ensures that the system is both powerful and flexible.

Contents

2. Approach

2.1 Basic streamlit application



Interact with CrewAI tasks with custom inputs.

Select a Task to Run: Question Answering

Enter your query: Type your research topic here...

We started with a basic application where we have our user interface for question answering. We can enter a query that the LLM uses to create an output.

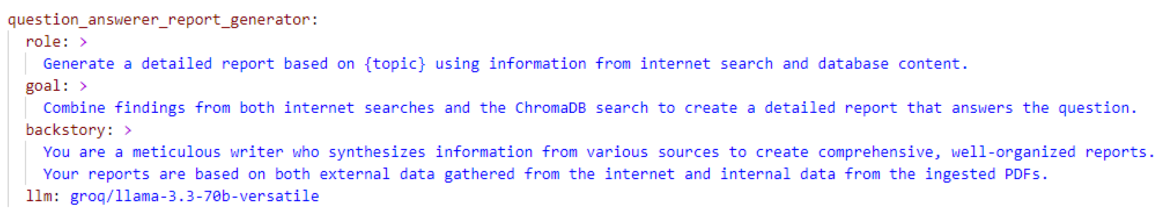
2.2 Crewai agents

```yaml
question_answerer_internet_searcher:
  role: >
    Search the web for information on {topic}.
  goal: >
    Perform an internet search and gather relevant articles, research papers, or other online resources about {topic}.
  backstory: >
    You're a diligent researcher who specializes in scouring the internet for accurate and up-to-date information.
    You use your expertise to gather data from various credible online sources to help answer questions.
  llm: groq/llama-3.3-70b-versatile
```

We create our agent for the question answering. This agent does a search on the internet about the provided topic

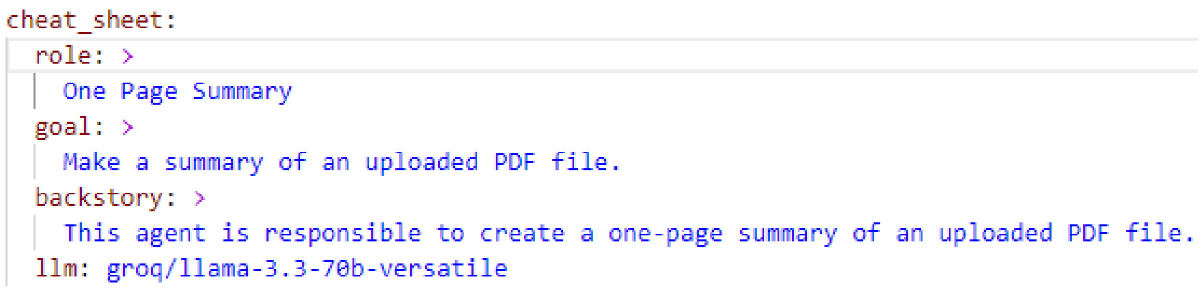
```yaml
question_answerer_database_searcher:
  role: >
    Search the ChromaDB for content related to {topic}.
  goal: >
    Search the database containing the ingested PDF content to find relevant sections or information about {topic}.
  backstory: >
    You're a database expert who specializes in extracting meaningful content from a collection of documents.
    You use advanced search techniques to find the most relevant parts of stored files based on a given topic.
  llm: groq/llama-3.3-70b-versatile
```

The second agent for question answering is an agent that searches the vector database for content related to the provided topic.



```yaml
question_answerer_report_generator:
  role: >
    Generate a detailed report based on {topic} using information from internet search and database content.
  goal: >
    Combine findings from both internet searches and the ChromaDB search to create a detailed report that answers the question.
  backstory: >
    You are a meticulous writer who synthesizes information from various sources to create comprehensive, well-organized reports.
    Your reports are based on both external data gathered from the internet and internal data from the ingested PDFs.
  llm: groq/llama-3.3-70b-versatile
```

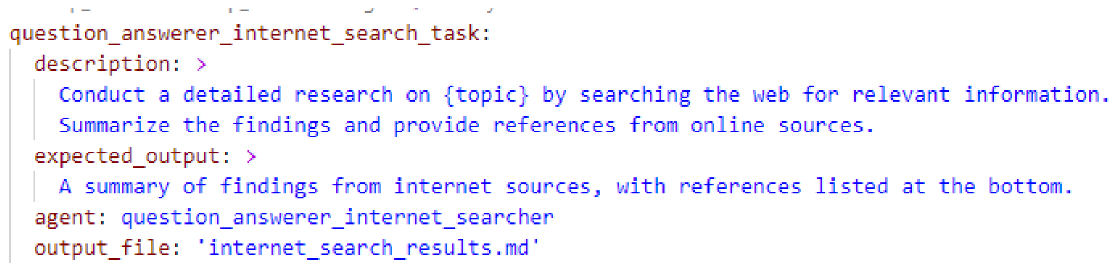
The third agent for question answering is an agent that makes a detailed report of all the information the other two agents have gathered.



```yaml
cheat_sheet:
  role: >
    One Page Summary
  goal: >
    Make a summary of an uploaded PDF file.
  backstory: >
    This agent is responsible to create a one-page summary of an uploaded PDF file.
  llm: groq/llama-3.3-70b-versatile
```

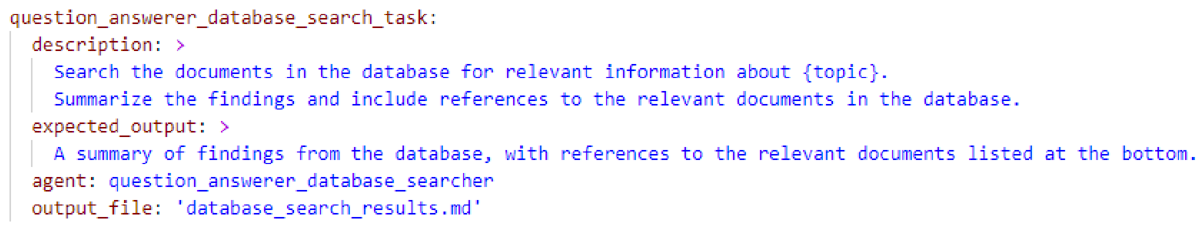
The final agent is to process an uploaded PDF file. Then he creates a one page summary with the content of the PDF file.

2.3 Crewai tasks



```yaml
question_answerer_internet_search_task:
  description: >
    Conduct a detailed research on {topic} by searching the web for relevant information.
    Summarize the findings and provide references from online sources.
  expected_output: >
    A summary of findings from internet sources, with references listed at the bottom.
  agent: question_answerer_internet_searcher
  output_file: 'internet_search_results.md'
```

Then we make our task for the internet searcher agent in CrewAI. The task is to do a detailed research on the internet on the provided topic. And the output is a md file with a small summary of what it found and the references.



```yaml
question_answerer_database_search_task:
  description: >
    Search the documents in the database for relevant information about {topic}.
    Summarize the findings and include references to the relevant documents in the database.
  expected_output: >
    A summary of findings from the database, with references to the relevant documents listed at the bottom.
  agent: question_answerer_database_searcher
  output_file: 'database_search_results.md'
```

Then we make our task for the database searcher agent in CrewAI. The task is to research the data in the database on the provided topic. And the output is a md file with a small summary of what it found and the references to the documents.

```yaml
question_answerer_report_task:
  description: >
    Generate a long detailed report based on the research findings from both internet and database searches.
    Add all the usefull data in this report. This report needs to have a good overview of everything
    The report should be divided into sections, and include two sections of references:
    - Online references, with links to the page.
    - Database references, with links to the relevant document.
  expected_output: >
    A detailed report divided into sections with headers. Each section should contain a summary of the findings.
    At the end of the report, include:
    - Online references, with links to the source.
    - Database references, with links to the relevant document.
  agent: question_answerer_report_generator
  output_file: 'question_answer_report.md'
```

This task is for the report agent. This task is to create a detailed report with all the information from the other two tasks. The report must be divided into different sections and at the bottom there must be an overview of the references, but the references from the internet and from the database must be separate.

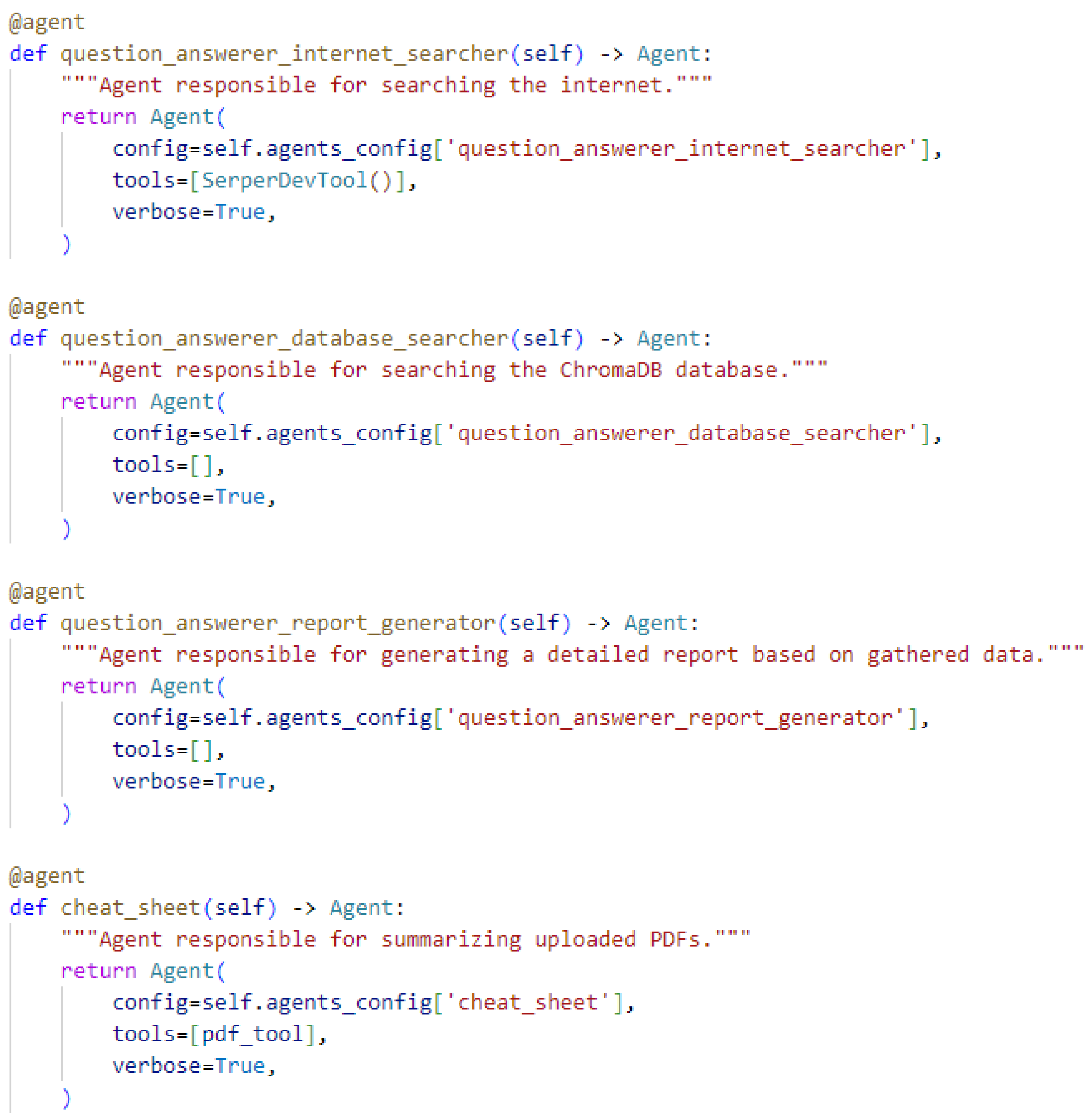
```yaml
cheat_sheet_task:
  description: >
    Review the context you got from the PDF file that was uploaded and make a one-page summary of the PDF file.
    Make sure you add everything in the report but don't make it to long
  inputs:
    - uploaded_file: A PDF file.
  expected_output: >
    A one-page summary of the provided file.
  agent: cheat_sheet
  output_file: 'report.md'
```

The final task is for the cheat sheet. This is for creating the one page summary of the input (PDF file). The output is a md file with the summary in it.

2.4 Crew.py configuration

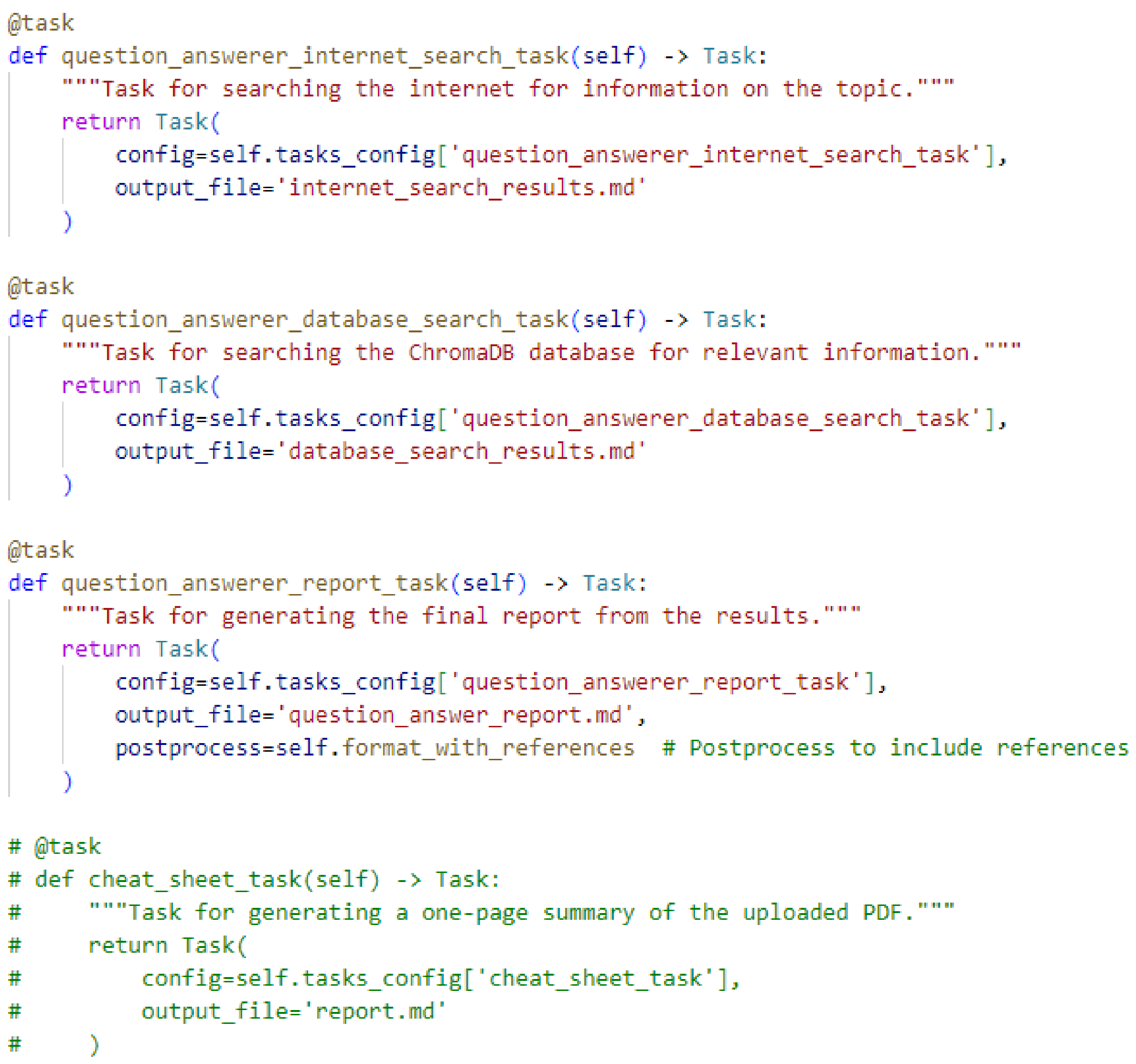
```python
# Specify the path to store ChromaDB data
db_path = "../../db_storage"

# Ensure the database directory exists
os.makedirs(db_path, exist_ok=True)

# Create the PersistentClient for ChromaDB
client = chromadb.PersistentClient(path=db_path)

# Initialize the SentenceTransformer model for text embeddings
model = SentenceTransformer("all-MiniLM-L6-v2")

# Initialize PDF tool for text extraction from PDFs
pdf_tool = PDFSearchTool()
```

First we setup our database (ChromaDB), our model for the embeddings and the PDFsearch tool.



```python
@agent
def question_answerer_internet_searcher(self) -> Agent:
    """Agent responsible for searching the internet."""
    return Agent(
        config=self.agents_config['question_answerer_internet_searcher'],
        tools=[SerperDevTool()],
        verbose=True,
    )

@agent
def question_answerer_database_searcher(self) -> Agent:
    """Agent responsible for searching the ChromaDB database."""
    return Agent(
        config=self.agents_config['question_answerer_database_searcher'],
        tools=[],
        verbose=True,
    )

@agent
def question_answerer_report_generator(self) -> Agent:
    """Agent responsible for generating a detailed report based on gathered data."""
    return Agent(
        config=self.agents_config['question_answerer_report_generator'],
        tools=[],
        verbose=True,
    )

@agent
def cheat_sheet(self) -> Agent:
    """Agent responsible for summarizing uploaded PDFs."""
    return Agent(
        config=self.agents_config['cheat_sheet'],
        tools=[pdf_tool],
        verbose=True,
    )
```

Then in our crew we define our agents with the configuration of the agents we made above. The internet_searcher_agent we give the SerperDevTool(), because he needs this tool to get information from the internet. We give the cheat sheet agent the pdf_tool so that he can store the embeddings from the pdf file in the database and can get the content to make his cheat sheet.



```python
@task
def question_answerer_internet_search_task(self) -> Task:
    """Task for searching the internet for information on the topic."""
    return Task(
        config=self.tasks_config['question_answerer_internet_search_task'],
        output_file='internet_search_results.md'
    )

@task
def question_answerer_database_search_task(self) -> Task:
    """Task for searching the ChromaDB database for relevant information."""
    return Task(
        config=self.tasks_config['question_answerer_database_search_task'],
        output_file='database_search_results.md'
    )

@task
def question_answerer_report_task(self) -> Task:
    """Task for generating the final report from the results."""
    return Task(
        config=self.tasks_config['question_answerer_report_task'],
        output_file='question_answer_report.md',
        postprocess=self.format_with_references  # Postprocess to include references
    )

# @task
# def cheat_sheet_task(self) -> Task:
#     """Task for generating a one-page summary of the uploaded PDF."""
#     return Task(
#         config=self.tasks_config['cheat_sheet_task'],
#         output_file='report.md'
#     )
```

Then in our crew we define our tasks with the configuration we made in in the Crewai tasks section. Each task creates a md file with the output of his task. The most important output is the question_answer_report.md file because this contains the output that will be send back to the front-end. The cheat sheet task is in comment, because it didn't work and if we didn't put it in comments it would also give problems when we tried to use the other tasks. The report task also uses the format_with_references function to add the references to the report this function will be explained below.

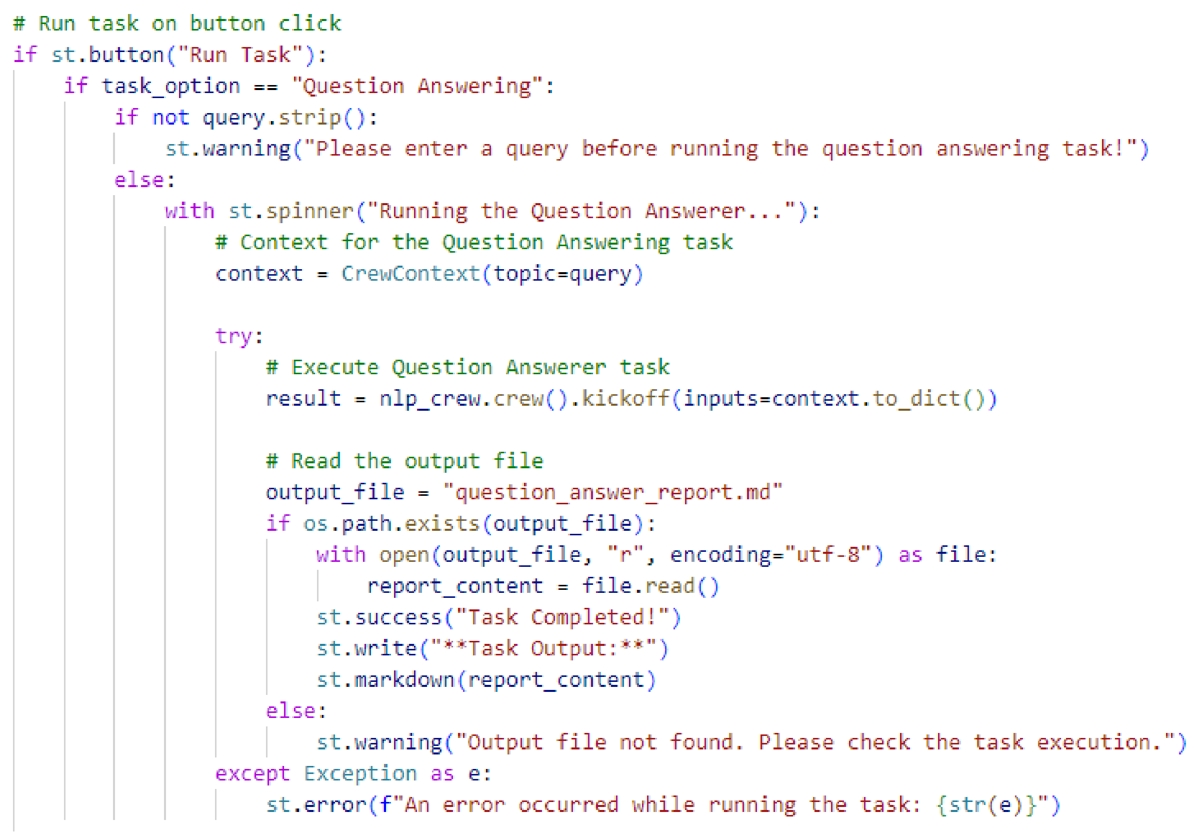
```python
def format_with_references(self, task_output):
    """
    Postprocess the task output to include references.
    """
    try:
        # Extract the main answer and references from the output
        answer = task_output.get("answer", "No answer provided.")
        references = task_output.get("references", [])

        # Format the references as a markdown-friendly list
        formatted_references = "\n".join(
            [f"{i+1}. {ref['title']} - {ref['url']}" for i, ref in enumerate(references)]
        )

        # Combine answer and references
        result = f"### Answer\n\n{answer}\n\n### References\n\n{formatted_references}"

        return result
    except Exception as e:
        return f"An error occurred while formatting the output: {e}"
```

In this function we first extract the answer and the references. Then we format the references so we can get a markdown-friendly list.

```python
@crew
def crew(self) -> Crew:
    """Creates the NlpCrew crew"""
    return Crew(
        agents=self.agents, # Automatically created by the @agent decorator
        tasks=self.tasks, # Automatically created by the @task decorator
        process=Process.sequential,
        verbose=True,
    )
```

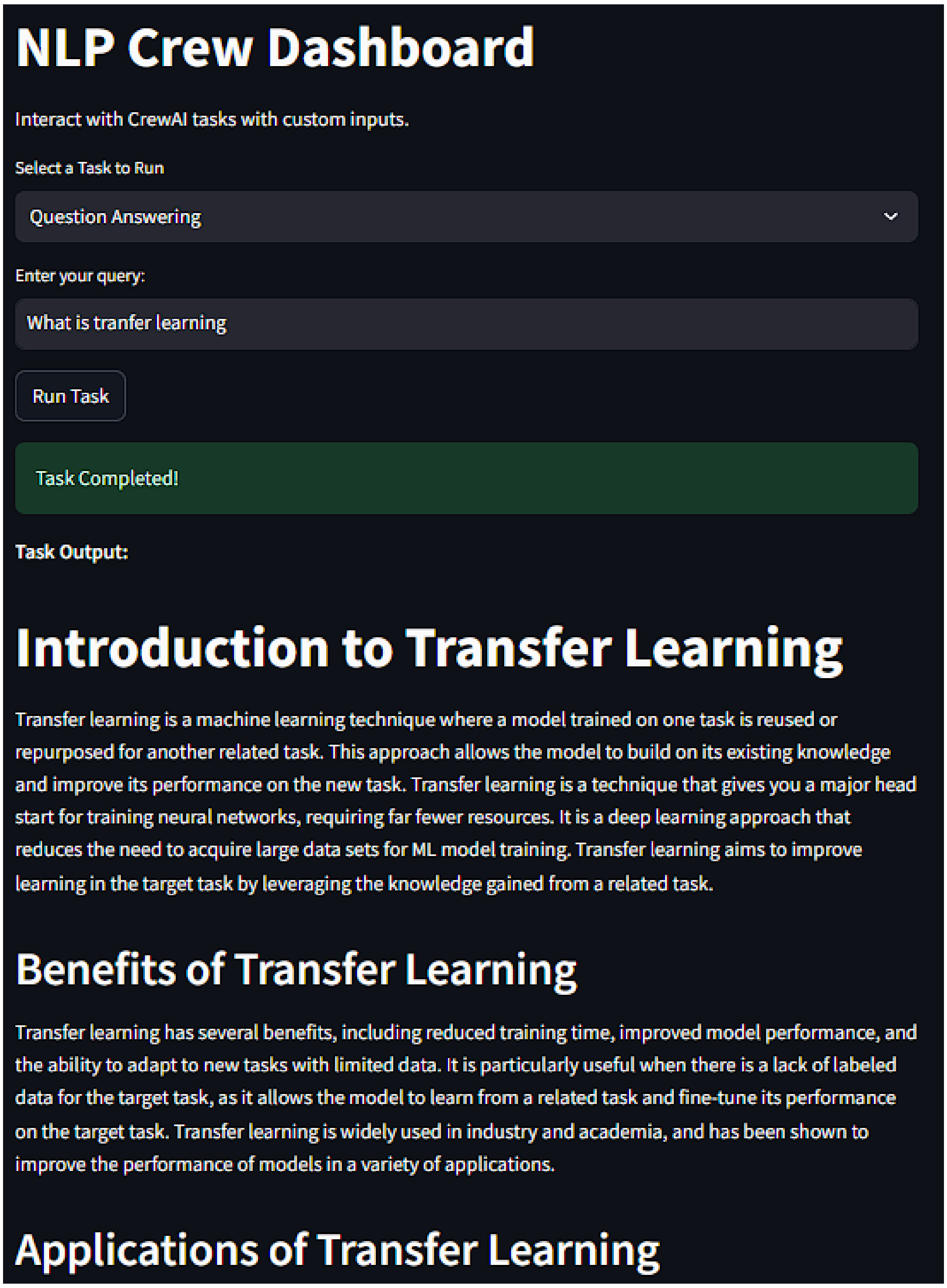
Then we made the crew where we define the agents and the tasks. The process is done sequentially.

2.5 Output



```python
# Run task on button click
if st.button("Run Task"):
    if task_option == "Question Answering":
        if not query.strip():
            st.warning("Please enter a query before running the question answering task!")
        else:
            with st.spinner("Running the Question Answerer..."):
                # Context for the Question Answering task
                context = CrewContext(topic=query)

                try:
                    # Execute Question Answerer task
                    result = nlp_crew.crew().kickoff(inputs=context.to_dict())

                    # Read the output file
                    output_file = "question_answer_report.md"
                    if os.path.exists(output_file):
                        with open(output_file, "r", encoding="utf-8") as file:
                            report_content = file.read()
                        st.success("Task Completed!")
                        st.write("**Task Output:**")
                        st.markdown(report_content)
                    else:
                        st.warning("Output file not found. Please check the task execution.")
                except Exception as e:
                    st.error(f"An error occurred while running the task: {str(e)}")
```

This is the streamlit code to get the output and display it in the front end. We get the md file that the report agent has created and display it as markdown in the view.



NLP Crew Dashboard

Interact with CrewAI tasks with custom inputs.

Select a Task to Run

Question Answering

Enter your query:

What is tranfer learning

Run Task

Task Completed!

Task Output:

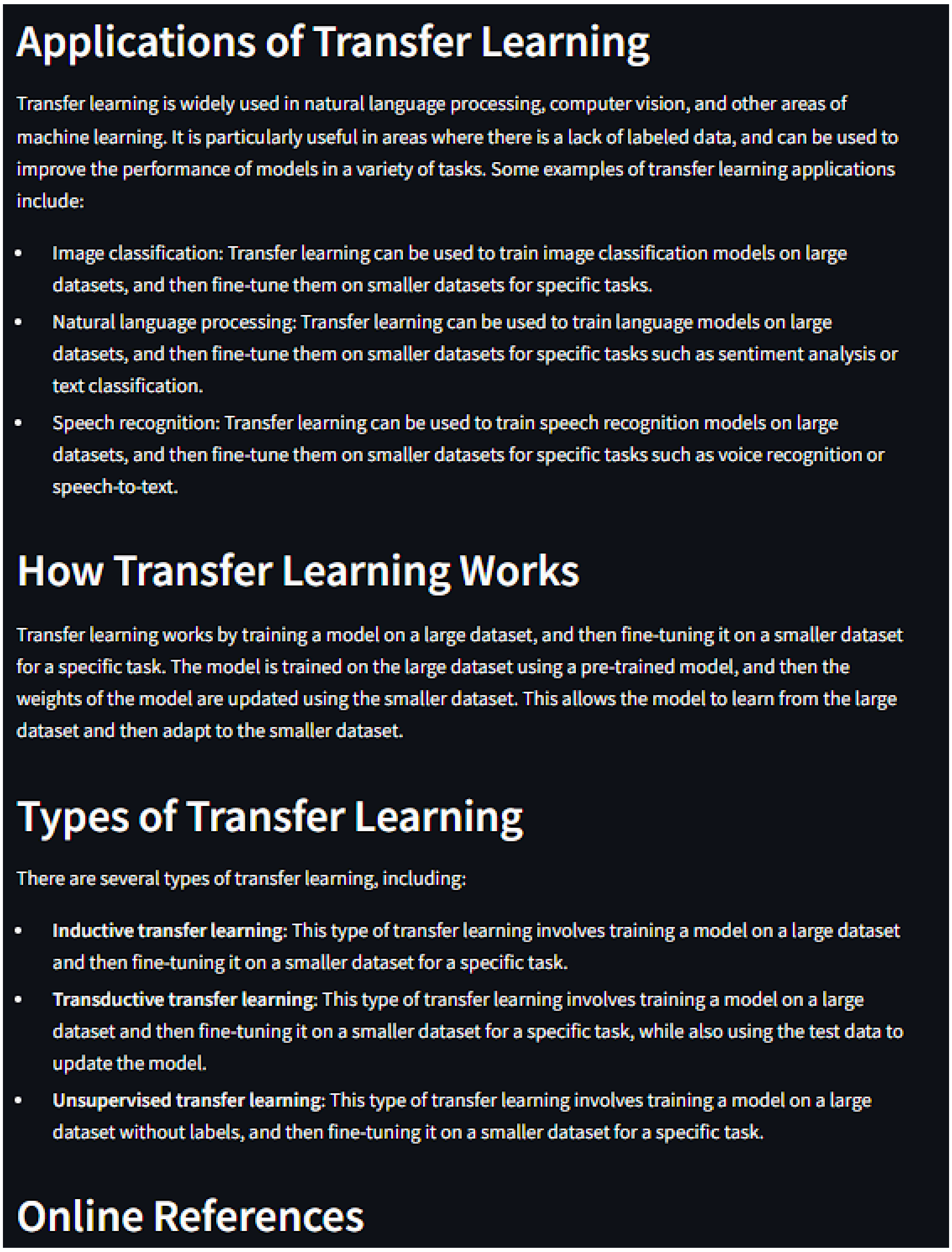
Introduction to Transfer Learning

Transfer learning is a machine learning technique where a model trained on one task is reused or repurposed for another related task. This approach allows the model to build on its existing knowledge and improve its performance on the new task. Transfer learning is a technique that gives you a major head start for training neural networks, requiring far fewer resources. It is a deep learning approach that reduces the need to acquire large data sets for ML model training. Transfer learning aims to improve learning in the target task by leveraging the knowledge gained from a related task.

Benefits of Transfer Learning

Transfer learning has several benefits, including reduced training time, improved model performance, and the ability to adapt to new tasks with limited data. It is particularly useful when there is a lack of labeled data for the target task, as it allows the model to learn from a related task and fine-tune its performance on the target task. Transfer learning is widely used in industry and academia, and has been shown to improve the performance of models in a variety of applications.

Applications of Transfer Learning



Applications of Transfer Learning

Transfer learning is widely used in natural language processing, computer vision, and other areas of machine learning. It is particularly useful in areas where there is a lack of labeled data, and can be used to improve the performance of models in a variety of tasks. Some examples of transfer learning applications include:

- Image classification: Transfer learning can be used to train image classification models on large datasets, and then fine-tune them on smaller datasets for specific tasks.
- Natural language processing: Transfer learning can be used to train language models on large datasets, and then fine-tune them on smaller datasets for specific tasks such as sentiment analysis or text classification.
- Speech recognition: Transfer learning can be used to train speech recognition models on large datasets, and then fine-tune them on smaller datasets for specific tasks such as voice recognition or speech-to-text.

How Transfer Learning Works

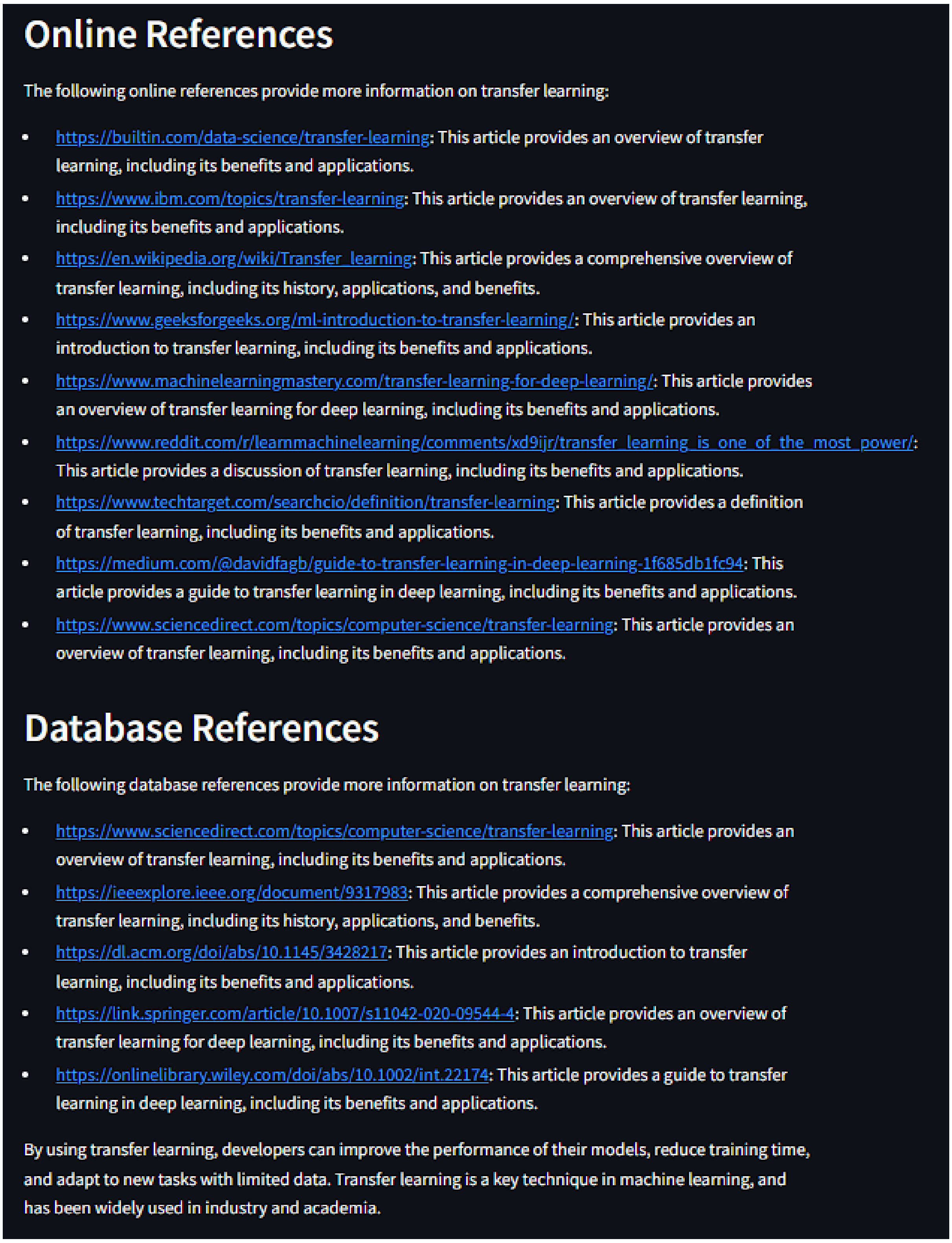
Transfer learning works by training a model on a large dataset, and then fine-tuning it on a smaller dataset for a specific task. The model is trained on the large dataset using a pre-trained model, and then the weights of the model are updated using the smaller dataset. This allows the model to learn from the large dataset and then adapt to the smaller dataset.

Types of Transfer Learning

There are several types of transfer learning, including:

- **Inductive transfer learning**: This type of transfer learning involves training a model on a large dataset and then fine-tuning it on a smaller dataset for a specific task.
- **Transductive transfer learning**: This type of transfer learning involves training a model on a large dataset and then fine-tuning it on a smaller dataset for a specific task, while also using the test data to update the model.
- **Unsupervised transfer learning**: This type of transfer learning involves training a model on a large dataset without labels, and then fine-tuning it on a smaller dataset for a specific task.

Online References



Online References

The following online references provide more information on transfer learning:

- https://builtin.com/data-science/transfer-learning: This article provides an overview of transfer learning, including its benefits and applications.
- https://www.ibm.com/topics/transfer-learning: This article provides an overview of transfer learning, including its benefits and applications.
- https://en.wikipedia.org/wiki/Transfer_learning: This article provides a comprehensive overview of transfer learning, including its history, applications, and benefits.
- https://www.geeksforgeeks.org/ml-introduction-to-transfer-learning/: This article provides an introduction to transfer learning, including its benefits and applications.
- https://www.machinelearningmastery.com/transfer-learning-for-deep-learning/: This article provides an overview of transfer learning for deep learning, including its benefits and applications.
- https://www.reddit.com/r/learnmachinelearning/comments/xd9ijr/transfer_learning_is_one_of_the_most_power/: This article provides a discussion of transfer learning, including its benefits and applications.
- https://www.techtarget.com/searchcio/definition/transfer-learning: This article provides a definition of transfer learning, including its benefits and applications.
- https://medium.com/@davidfagb/guide-to-transfer-learning-in-deep-learning-1f685db1fc94: This article provides a guide to transfer learning in deep learning, including its benefits and applications.
- https://www.sciencedirect.com/topics/computer-science/transfer-learning: This article provides an overview of transfer learning, including its benefits and applications.

Database References

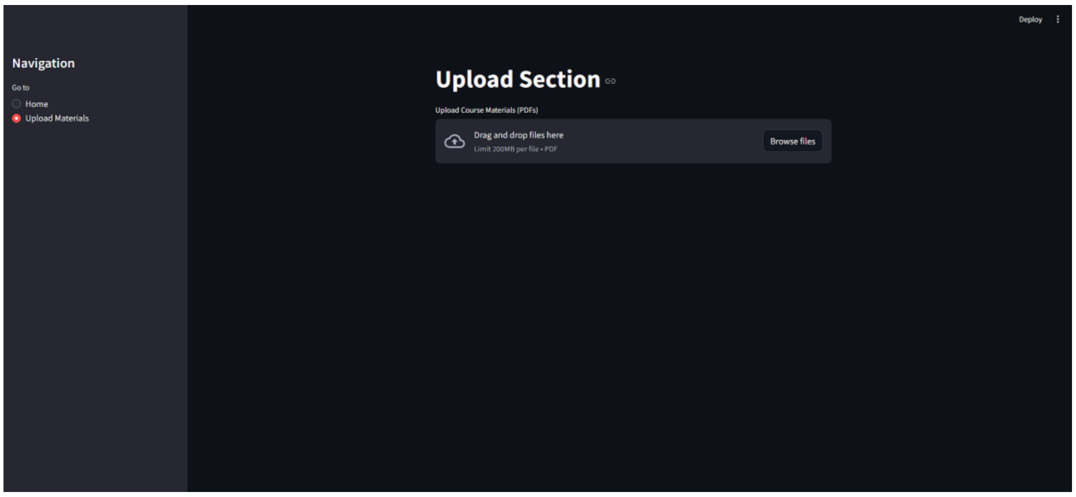
The following database references provide more information on transfer learning:

- https://www.sciencedirect.com/topics/computer-science/transfer-learning: This article provides an overview of transfer learning, including its benefits and applications.
- https://ieeexplore.ieee.org/document/9317983: This article provides a comprehensive overview of transfer learning, including its history, applications, and benefits.
- https://dl.acm.org/doi/abs/10.1145/3428217: This article provides an introduction to transfer learning, including its benefits and applications.
- https://link.springer.com/article/10.1007/s11042-020-09544-4: This article provides an overview of transfer learning for deep learning, including its benefits and applications.
- https://onlinelibrary.wiley.com/doi/abs/10.1002/int.22174: This article provides a guide to transfer learning in deep learning, including its benefits and applications.

By using transfer learning, developers can improve the performance of their models, reduce training time, and adapt to new tasks with limited data. Transfer learning is a key technique in machine learning, and has been widely used in industry and academia.

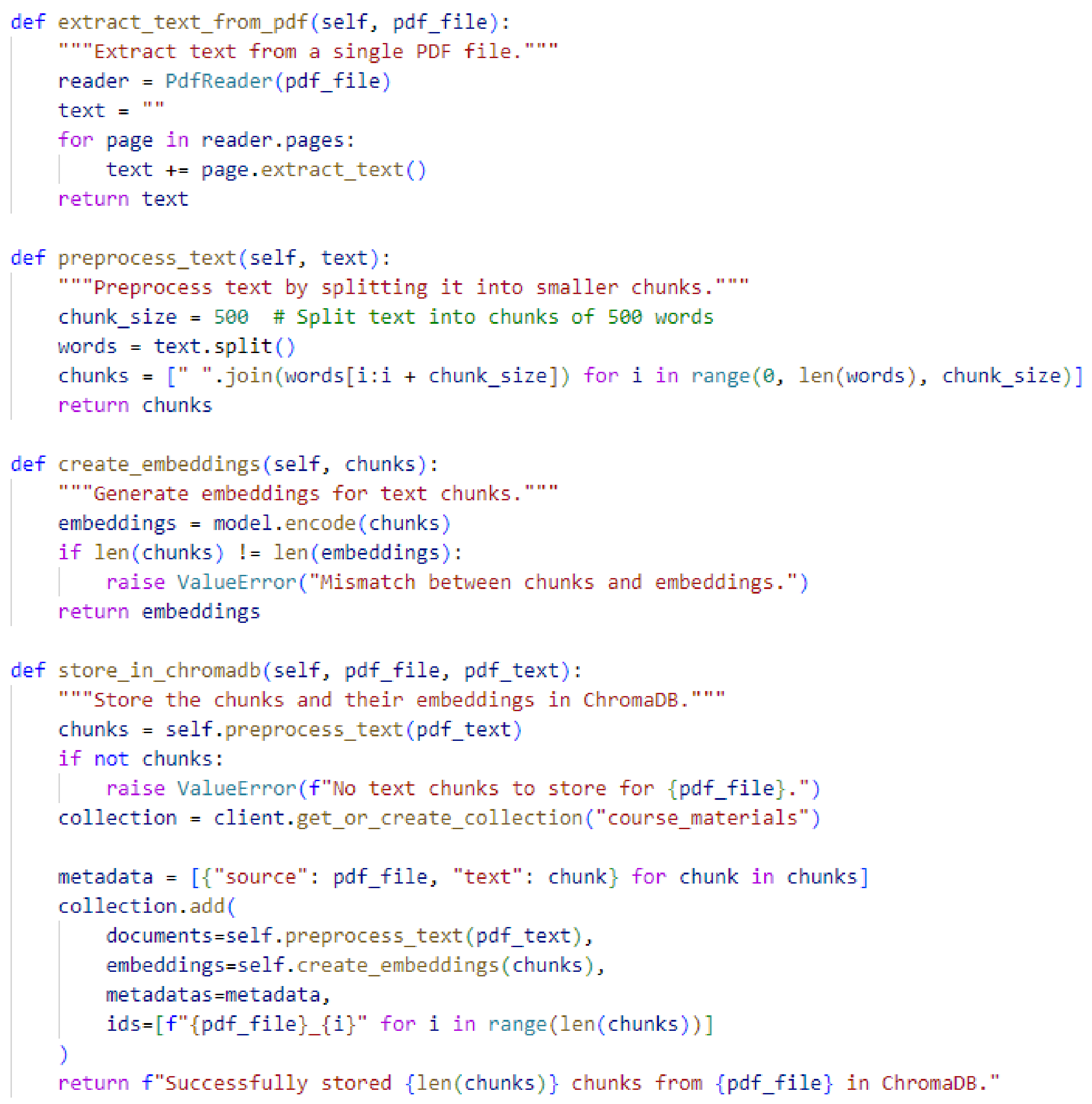
The images above is an example of the output that is created by the crew. As we can see It divides it in section and it splits the references in online and database references. The only problem is that the database searcher also has an output with references from the internet and not from the database.

2.6 PDF input



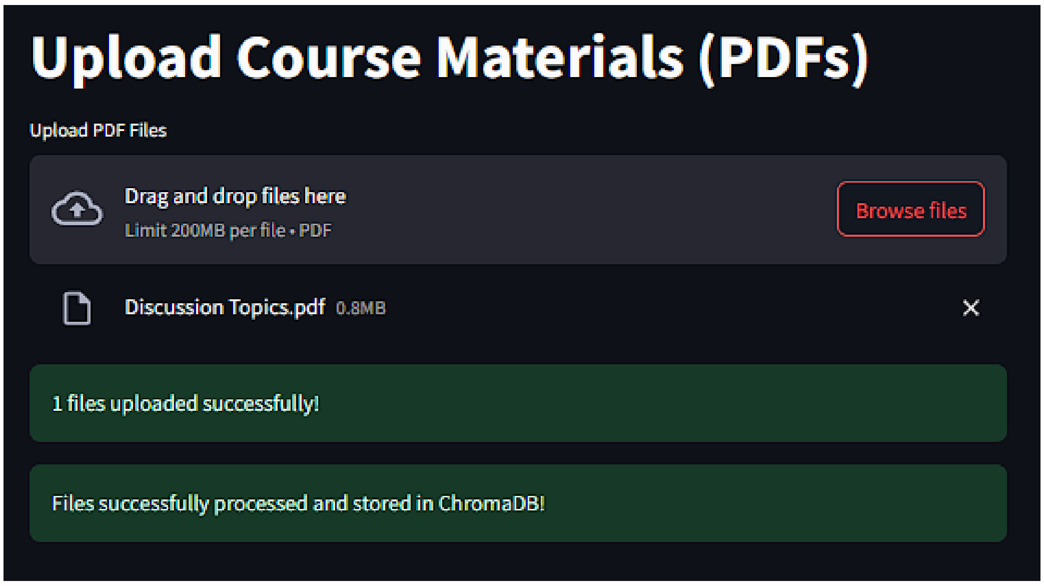
```python
elif selection == "Upload Materials":
    # Upload Section for PDFs
    st.title("Upload Course Materials (PDFs)")

    # Upload file button
    uploaded_files = st.file_uploader("Upload PDF Files", type="pdf", accept_multiple

    if uploaded_files:
        # Create a temporary directory to store uploaded files
        temp_dir = "temp_files"
        os.makedirs(temp_dir, exist_ok=True)

        # Save the uploaded files temporarily and store their paths
        temp_file_paths = []
        for uploaded_file in uploaded_files:
            temp_file_path = os.path.join(temp_dir, uploaded_file.name)
            with open(temp_file_path, "wb") as f:
                f.write(uploaded_file.getbuffer())  # Save the file content
            temp_file_paths.append(temp_file_path)

        # Inform the user about the successful upload
        st.success(f"{len(uploaded_files)} files uploaded successfully!")

        # Process the uploaded PDFs and store them in ChromaDB
        with st.spinner("Processing files and storing in ChromaDB..."):
            try:
                # Convert PDFs to text and store in ChromaDB
                for file_path in temp_file_paths:
                    # Extract text from the PDF
                    pdf_text = nlp_crew.extract_text_from_pdf(file_path)

                    # Store the extracted text into ChromaDB
                    nlp_crew.store_in_chromadb(pdf_text=pdf_text, pdf_file=file_path)

                st.success("Files successfully processed and stored in ChromaDB!")
            except Exception as e:
                st.error(f"Error processing files: {str(e)}")
```

Next we improved our frontend where we made a navigation to go to the upload section where we can upload PDF’s that we store in our vector database.



```python
def extract_text_from_pdf(self, pdf_file):
    """Extract text from a single PDF file."""
    reader = PdfReader(pdf_file)
    text = ""
    for page in reader.pages:
        text += page.extract_text()
    return text

def preprocess_text(self, text):
    """Preprocess text by splitting it into smaller chunks."""
    chunk_size = 500  # Split text into chunks of 500 words
    words = text.split()
    chunks = [" ".join(words[i:i + chunk_size]) for i in range(0, len(words), chunk_size)]
    return chunks

def create_embeddings(self, chunks):
    """Generate embeddings for text chunks."""
    embeddings = model.encode(chunks)
    if len(chunks) != len(embeddings):
        raise ValueError("Mismatch between chunks and embeddings.")
    return embeddings

def store_in_chromadb(self, pdf_file, pdf_text):
    """Store the chunks and their embeddings in ChromaDB."""
    chunks = self.preprocess_text(pdf_text)
    if not chunks:
        raise ValueError(f"No text chunks to store for {pdf_file}.")
    collection = client.get_or_create_collection("course_materials")

    metadata = [{"source": pdf_file, "text": chunk} for chunk in chunks]
    collection.add(
        documents=self.preprocess_text(pdf_text),
        embeddings=self.create_embeddings(chunks),
        metadatas=metadata,
        ids=[f"{pdf_file}_{i}" for i in range(len(chunks))]
    )
    return f"Successfully stored {len(chunks)} chunks from {pdf_file} in ChromaDB."
```

This are the function to store the content in the vector database. In the first function we read our pdf and get the text from the pdf file. Then we split our text in chunks of 500 words. With the next function we make embeddings from the chunks we created. We make those embeddings with the SenteceTransformer we defined at the top of out python file. In the last function we store our pages and embeddings in the vector database. In the collection “course_materials”.


Upload Course Materials (PDFs)

Upload PDF Files

Drag and drop files here
Limit 200MB per file • PDF

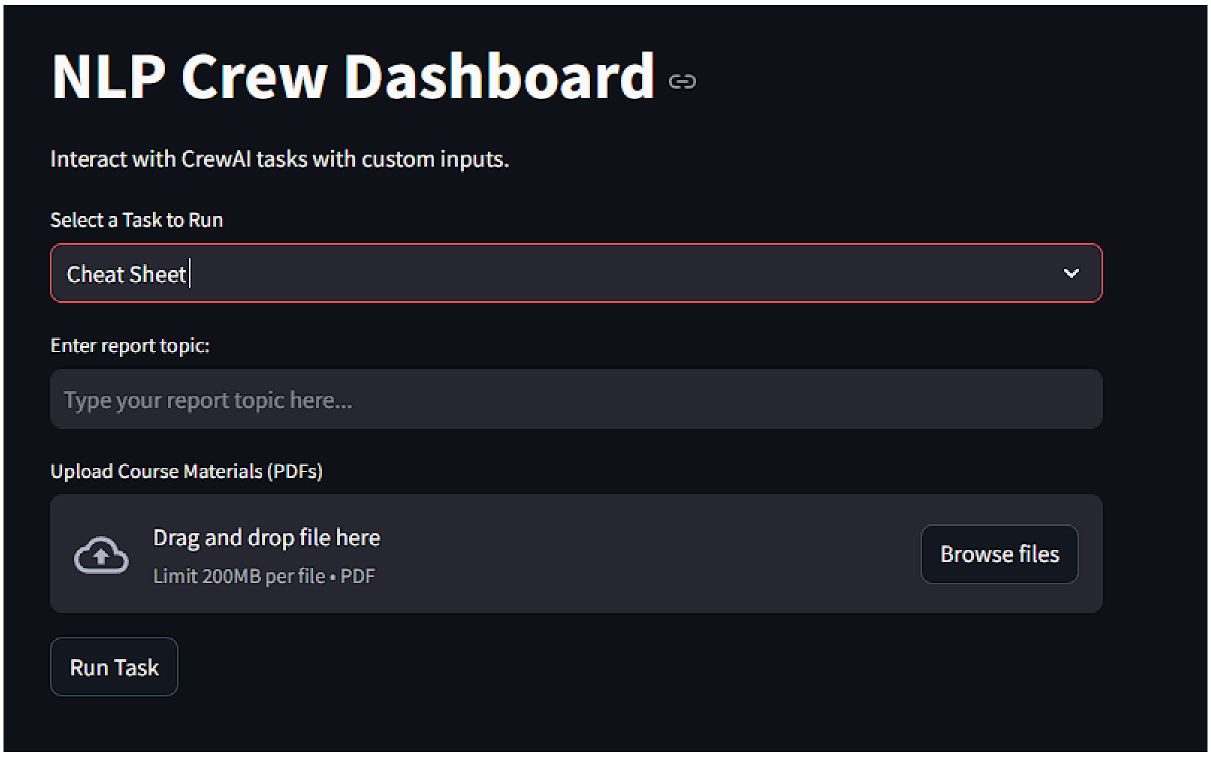
Browse files

Discussion Topics.pdf 0.8MB

1 files uploaded successfully!

Files successfully processed and stored in ChromaDB!

2.7 Cheat Sheet


NLP Crew Dashboard

Interact with CrewAI tasks with custom inputs.

Select a Task to Run: Cheat Sheet

Enter report topic: Type your report topic here...

Upload Course Materials (PDFs)

Drag and drop file here
Limit 200MB per file • PDF

Browse files

Run Task

We improved our frontend where we now can choose for cheat sheet and an extra field will show where we can upload our content that is used to create a one page summary of the content.

3. Obstacles

3.1 Content ingestion

First we made the content ingestion with a crewai agent. This agent had the PDFsearch tool, so that it could store the content in the database. For some reason this wouldn’t work.

We deleted this agent and tried to store the content in the database without crewai. In this way we created our own functions to store the content. In this way we could store our content in the database.

3.2 Cheat sheet

When using the cheat sheet in the agent to create our one-page document. The other agent were also contributing. This way the output we got was not from the uploaded document, but also with information from the internet. When we tried to fix this problem we only made it worse and it didn’t work anymore. That’s why we commented the code, so the rest of the application would still work.

4. Generative AI

Generative AI played a crucial role in shaping various aspects of our project, helping us streamline development and solve complex challenges. Below, we outline the specific ways in which AI assisted us:

1. **Identifying Problems and Providing Explanations**
   Generative AI was a valuable resource for identifying and debugging issues in our code. It helped us pinpoint problems efficiently and offered clear, detailed explanations of the underlying causes. This allowed us to resolve issues faster and deepen our understanding of the technologies we were working with.

2. **Content Ingestion Development**
   The most significant contribution of AI was in the **content ingestion module**. We leveraged generative AI to design and implement functions for parsing and processing data. AI assisted us in conceptualizing and creating the foundation of this module, ensuring it was both robust and efficient. The outputs from AI served as a starting point, which we adapted and customized to meet our specific requirements.

3. **Streamlit Code Generation**
   For the **Streamlit** user interface, generative AI provided templates and examples of basic code structures. These were not used verbatim but served as a foundation that we modified to align with the unique needs of our project. This saved us significant time during the initial stages of development.

4. **CrewAI Integration**
   While we explored using generative AI for **CrewAI** integration, its contributions in this area were limited. As CrewAI is relatively new, the AI-generated suggestions for this framework were not always accurate or usable. We relied more on manual exploration and documentation to implement CrewAI functionalities effectively.

5. Conclusion

This project was both fun and highly educational. Working with the latest technologies made it exciting and relevant, but also challenging due to the limited resources available online. Despite these difficulties, the experience pushed us to think critically, experiment, and deepen our understanding. Overall, it was a rewarding journey that enhanced both our skills and adaptability.