Case Analysis Assistant in Danswer Application

The goal of this project was to integrate a Case Analysis Assistant feature in the Danswer application. This assistant is designed to analyze legal case queries and provide a prediction (e.g., Accepted/Rejected) along with reasoning for the prediction.The implementation involved both backend and frontend work, as well as modifications to configuration files to integrate FastAPI, Hugging Face models, Docker, and the application's assistant-handling logic.

Server Implemented on (danswer):
(http://ec2-54-79-231-211.ap-southeast-2.compute.amazonaws.com)

Server Referenced (open-webui-main):
(http://ec2-13-237-148-39.ap-southeast-2.compute.amazonaws.com)

(Site: http://54.79.231.211)

1) open-webui-main server:

Path: cd open-webui-main/backend/apps/caseprediction/:

Files present in **caseprediction dir**:

➔ **abc.py, config.json, main.py, model.safetensors, special_tokens_map.json, spiece.model, tokenizer_config.json**

Summary: The **case prediction logic** from the **open-webui-main server served** as the **basis** for this **implementation**. The **caseprediction directory** on the **open-webui-main server** provided a **framework** for **handling case queries** using **AutoModelForSequenceClassification** and **AutoTokenizer**, as well as **generating reasoning** with **HuggingFaceHub**.

❖ **Key components:**

- The **abc.py file** in the **open-webui-main server** used **SentenceTransformer** for **encoding queries** and **HuggingFaceHub** for **reasoning generation (NousResearch/Nous-Hermes-2-Mixtral-8x7B-DPO)**.

- The **case prediction model** was **loaded** from the **local model directory (i.e model.safetensors** & **spiece.model)**, and the **reasoning** was **generated** based on the **query**.

- **Modifications**: **Instead** of **directly copying**, we **adapted these ideas** to **fit** into the **Danswer application** by **designing** a new **main.py** and **modifying Docker configurations**, **among** other **changes**.

2) (main.py) Backend:

Path: cd danswer/backend/danswer/caseprediction/main.py:

Summary: The **backend** for the **Case Prediction feature** was **implemented using FastAPI**, which provides a **POST endpoint** that **accepts legal case queries** and **returns** the **prediction** along with **reasoning**. The **backend** also **interacts** with **Hugging Face's models** for both **prediction** and **reasoning generation**.

❖ Key Modifications:

- **Imported Hugging Face** model **loading**, **tokenization**, and **configuration** for **prediction tasks**.

  Code:

  from transformers import AutoModelForSequenceClassification, AutoTokenizer, AutoConfig

➔ **AutoModelForSequenceClassification**: **Loads** a pre-**trained** model for **sequence classification** (**binary classification** in this **case**).

➔ **AutoTokenizer**: **Loads** the **tokenizer corresponding** to the **model** for **input tokenization**.

➔ **AutoConfig**: **Loads** the **configuration** of the **model**.

  Code:

- **load_dotenv()**
  HUGGINGFACE_API_TOKEN = os.getenv("HUGGINGFACEHUB_API_TOKEN")

  Purpose: **Loads** the **API token** for **Hugging Face** from the .**env file**, allowing the **backend** to **authenticate** and **access Hugging Face models** for **reasoning generation**.

  .env: HUGGINGFACEHUB_API_TOKEN=(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)

Code:

- **MODEL_PATH = "/app"**
  config = AutoConfig.from_pretrained(MODEL_PATH)
  tokenizer = AutoTokenizer.from_pretrained(MODEL_PATH, config=config, local_files_only=True, use_fast=False)
  model = AutoModelForSequenceClassification.from_pretrained(MODEL_PATH, config=config, local_files_only=True)

➔ **Purpose:** **Loads** the **model**, **tokenizer**, and **configuration** from a **pre-defined directory inside** the **Docker container**. **These are required** to process **incoming queries** and make **predictions** using the **sequence classification model**.

Code:

- **llm_client = HuggingFaceHub(repo_id="mistralai/Mixtral-8x7B-Instruct-v0.1", model_kwargs={"temperature": 0.2, "max_length": 10000})**

➔ **Purpose:** **Initializes** the **Hugging Face LLM** (**Large Language Model**) **client** using the **langchain package**. This **model** is used to **generate reasoning based** on the **case prediction result**.

- **class CaseQuery(BaseModel):**
  query: str

  class PredictionResult(BaseModel):
  prediction: int
  reasoning: str

➔ **Purpose:** **Defines** the **input** and **output models** using **Pydantic's BaseModel**. These **models** structure the **API request body** and **response**, ensuring **data** is **validated** and returned in a **predictable format**:

- **CaseQuery: Contains** the **query** for **case prediction**.

- **PredictionResult**: **Contains** the **prediction** (**Accepted/Rejected**) and **reasoning text**.

Code:

- **@app.post("/caseprediction", response_model=PredictionResult)**
  async def case_prediction(query_data: CaseQuery):
  **   try:**
  **   # Tokenization**

```
        inputs = tokenizer(query_data.query, return_tensors="pt", padding=True, truncation=True, max_length=512)

        # Model inference
        with torch.no_grad():
            outputs = model(**inputs)
            predictions = torch.argmax(outputs.logits, dim=-1)
            prediction = predictions.numpy()[0]

        # Generate label
        label = "accepted" if prediction == 1 else "rejected"

        # Generate reasoning using LLM
        reasoning_prompt = f"The following case was {label} based on the analysis: {query_data.query}"
        reasoning = llm_client.generate(
            prompts=[reasoning_prompt],
            max_new_tokens=1000,
            temperature=0.2
        )
        reasoning_text = reasoning.generations[0][0].text.strip()

        # Prepare response
        response_content = {"prediction": int(prediction), "reasoning": reasoning_text}
        return jsonable_encoder(response_content)

    except Exception as e:
        logger.error(f"Error processing case prediction: {str(e)}")
        return JSONResponse(status_code=500, content={"detail": str(e)})
```

Purpose: The **/caseprediction endpoint processes** a **legal case query** and **performs** the **following** steps:

- **Tokenization**: The **input query** is **tokenized** using the **pre-loaded Hugging Face tokenizer**.
- **Prediction**: The **model predicts** whether the **case** will be **accepted** or **rejected**.
- **Reasoning Generation**: **Using** the **Hugging Face Hub client**, **reasoning** is **generated based** on the **prediction**.
- **Response**: The **prediction** and **reasoning** are **returned** as a **JSON response** using **jsonable_encoder**.

3) Dockerfile for caseprediction setup:

Summary: This **Dockerfile sets** up the **environment** and **dependencies** for the **Case Prediction service**.

❖ **Key Modifications:**

- **Model File Copying:**

  Code:

- **COPY model.safetensors config.json spiece.model special_tokens_map.json tokenizer_config.json /app/**

➔ **Copies model files** (**including model**.**safetensors**, **config**.**json**, **special_toknes_map.json, tokenizer_config.json** and **tokenizer files**) into the **container**.

- **Application Setup:** **Copies** the **application files** (**main**.**py**, .**env**, and **config**.**py**) and **sets environment variables**.

  Code:

  COPY main.py ./
  COPY config.py ./
  COPY .env ./

- **Why is MODEL_PATH set to /app?**

➔ **In Docker-based environments**, **every container runs** in its own **isolated filesystem**. **When** you **build** a **Docker image**, you **define** a **working directory inside** the **container where** all the **files** will be **stored** and **accessed during runtime**. In **our case**, the **working directory** for **our application** is /**app inside** the **Docker container**.

In the Dockerfile for the caseprediction service, we have this line:

```
WORKDIR /app
```

Now, coming to **MODEL_PATH**, it's **set** to /**app because** that's **where we're copying** all the **necessary files**, including **the model files** (**model**.**safetensors**, **config**.**json**, **special_tokens_map**.**json**, **tokenizer_config**.**json** and **spiece**.**model**) that are **required** for **loading** and **running** the **model**. This **ensures** that the **tokenizer** and

model can **find** the **correct** files in **the** /**app directory** when **they** are **loaded** in the **code**.

4) (ChatPage.tsx) Frontend:

Path: cd danswer/web/src/app/chat/ChatPage.tsx:

Summary: In the **ChatPage.tsx file**, **several** key **modifications** were **made** to **integrate** the **Case Analysis Assistant**.

❖ **Key Modifications:**

- **API Call Logic Based on Selected Assistant:** A **new function** was **added** to **check** if the **selected assistant** is the **Case Analysis Assistant** and **route** the **API request accordingly**:

  Code:

```
// Helper function to check if the selected assistant is Case Analysis
const isCaseAnalysisAssistant = (assistant: Persona | null): boolean => {
  return assistant?.name === "Case Analysis"; // Assuming we know the assistant's name
};
```

  This function determines if the current assistant selected by the user is the Case Analysis assistant.

- **Conditional API Call for Case Analysis Assistant:** The **onSubmit function** was **modified** to make an **API call** to the **Case Prediction backend** when the **Case Analysis Assistant** is **selected**

  Code:

```
if (isCaseAnalysisAssistant(liveAssistant)) {
  console.log("Case Analysis assistant selected, sending request...");

  try {
    const response = await fetch(
      "http://54.79.231.211:8006/caseprediction", // URL of the case prediction server
      {
        method: "POST",
        body: JSON.stringify({
          query: currMessage,
        }),
```

```
        headers: {
          "Content-Type": "application/json",
        },
      }
    );

    if (!response.ok) {
      console.error(
        "Error with Case Analysis Assistant:",
        await response.text()
      );
      setPopup({
        message: `Error with Case Analysis Assistant: ${await response.text()}`,
        type: "error",
      });
      return;
    }

    const result = await response.json();
    console.log("Received case prediction result:", result);

    // Handle the response from the case prediction API
    setCompleteMessageDetail((prev) => ({
      ...prev,
      messageMap: new Map(prev.messageMap).set(currChatSessionId, {
        messageId: currChatSessionId,
        message: `Prediction: ${result.prediction ? "Accepted" : "Rejected"
          }\n\nReasoning:\n${result.reasoning}`,
        type: "assistant",
        retrievalType: RetrievalType.None,
        files: [], // Empty array or you can add relevant files if needed
        toolCalls: [], // Empty array or any tool calls if needed
        parentMessageId: null, // If there's a parent message, set the appropriate ID,
or use null
      }),
    }));
  } catch (error) {
    setPopup({
      message: "Error with Case Analysis Assistant",
      type: "error",
    });
  }

  setChatState("input"); // Stop further processing once case prediction is handled
```

return;
}

- If the **Case Analysis Assistant** is **selected**, a **POST request** is **made** to the **/caseprediction API (http://54.79.231.211:8006/caseprediction).**

- The **user query** is **sent** in the **request body**.

- The **response from** the **API includes** a **prediction** (**accepted/rejected**) and **reasoning text**, **which** is **added** to the **chat**.

- The **chat UI** is **updated** with the **response** from the **Case Analysis API**, and the **request processing** is **stopped** after the **case prediction** is **handled**.

- **Message Reset After Submission:** **To ensure** the **input field** is **cleared right after** the **message** is **submitted**, the **following** line **was** added in the **onSubmit function**

  resetInputBar(); // Call this function right here to clear the input

- This **clears** the **input bar** as **soon** as a **message** is **submitted**, **improving** the **user experience** by **ensuring that** the **input area** is **empty** when a **new message** is **being processed**.

1. PREVIOUS CHAT MESSAGE RECOVER - YET TO BE IMPLEMENTED - COMPLETED

2. STREAMING RESPONSE - YET TO BE IMPLEMENTTED

5) docker-compose.dev.yml

Path: cd danswer/deployment/docker-compose/docker-compose.dev.yml:

Summary: The **docker-compose.dev.yml file defines** the **configuration** for **running** the **caseprediction service** within the **Danswer application**. Here's a **breakdown** of the **configuration** for the **caseprediction service**

❖ **Key Modifications:**

Code:

```
services:
  caseprediction:
    build:
      context: ../../backend/danswer/caseprediction
```

```
    dockerfile: Dockerfile
  env_file:
    - ../../backend/danswer/caseprediction/.env
  ports:
    - "8006:8000"
  restart: unless-stopped
  volumes:
    - ../../backend/danswer/caseprediction:/app
```

- The **caseprediction service** is **responsible** for **running** the **Case Analysis Assistant backend.**

- It **builds** the **service** from the **caseprediction directory** using the **specified Dockerfile**.

- The **service exposes port 8006** on the **host**, allowing **access** to the **case prediction API**.

- It **uses environment variables** from the .**env file** for **configurations** like **API tokens**.

- The **volumes section** ensures that **code** and **model files** from the **host machine** are **available** inside the **Docker container**.

6) personas.yaml:

Summary: The **Case Analysis Assistant** was **introduced** by **defining** a **new persona** in the **personas**.**yaml file**. **Each persona** in the **Danswer application** is **defined** with **various parameters** that **determine** how the **assistant functions**, the **prompt** it **uses**, and its **appearance** in the **UI**. Here's how the **Case Analysis assistant** was **created** in the **personas**.**yaml file**

Code:

```
- id: 4
  name: "Case Analysis"
  description: >
    Assistant for predicting case outcomes and providing legal reasoning.
  prompts:
    - "CaseAnalysis"
  num_chunks: 0
  llm_relevance_filter: false
  llm_filter_extraction: false
  recency_bias: "no_decay"
  document_sets: []
```

icon_shape: 123456
icon_color: "#FFA500"
display_priority: 4
is_visible: true

- The **Case Analysis Assistant** was **added** to the **personas.yaml file** with a **unique ID** (**4**), **its own name** ("**Case Analysis**"), and a **corresponding "CaseAnalysis" prompt** defined in **prompts.yaml (Currently Commented Out)**.

- The **assistant** is **designed** to **predict case outcomes** and **generate legal reasoning**, **without interacting** with **external document sources (Based on prompts.yaml)**.

- The **assistant** was **made visible** in the **UI** with a **specific icon** and **color scheme**, **providing users** with a **tailored experience** for **case analysis tasks**.

7) Deprecation Warnings to be Noted:

➢ ARC4 Deprecation in cryptography library:

Code:

/usr/local/lib/python3.11/site-packages/pypdf/_crypt_providers/_cryptography.py:32: CryptographyDeprecationWarning: ARC4 has been moved to cryptography.hazmat.decrepit.ciphers.algorithms.ARC4 and will be removed from this module in 48.0.0.
from cryptography.hazmat.primitives.ciphers.algorithms import AES, ARC4

- **Description: The ARC4 cipher is deprecated in the cryptography library and will be removed in version 48.0.0. This warning is triggered by the pypdf package, which is using the ARC4 cipher.**

- **Action for the Future: Ensure that the pypdf library and any other relevant dependencies are updated to versions that no longer rely on ARC4 before cryptography version 48.0.0 is released.**

➢ Coercing Subquery Object in SQLAlchemy:

Code:

/app/danswer/db/llm.py:119: SAWarning: Coercing Subquery object into a select() for use in IN(); please pass a select() construct explicitly
LLMProvider__UserGroup.user_group_id.in_(user_groups_subquery) # type: ignore

/app/danswer/db/persona.py:192: SAWarning: Coercing Subquery object into a select() for use in IN(); please pass a select() construct explicitly
_Persona__UserGroup.user_group_id.in_(user_groups_subquery) # type: ignore

- **Description: These warnings indicate that SQLAlchemy is automatically coercing subquery objects into select() constructs for use in an IN() clause. While this behavior is currently supported, SQLAlchemy advises explicitly passing a select() construct in future code to avoid reliance on this automatic coercion.**

- **Action for the Future: Refactor the relevant SQLAlchemy queries in llm.py and persona.py to explicitly use select() constructs in place of subquery objects.**

Note: Failed to get max tokens for LLM with name mistralai/Mixtral-8x7B-Instruct-v0.1. Defaulting to 4096. This indicates that the system failed to retrieve the maximum token limit for the specific LLM model mistralai/Mixtral-8x7B-Instruct-v0.1. As a result, it defaulted to a token limit of 4096.

To access the Danswer application as an admin, please log in as:
Email: noelshallum@gmail.com
Password: 12345
Case Analysis Assistant: (http://54.79.231.211/chat?assistantId=-4)

- **Ibrahim Sultan**