Rubrics Generator

Transforming Grading with Intelligent Automation.

By Ansh Aya

07/23/2024

Introduction

The AI-driven Automated Rubric Generation System signifies a significant advancement in educational technology, particularly in the realm of grading and assessment. By harnessing the capabilities of AI and LLMs, this system automates the creation of detailed and customized rubrics that align with specific grading needs. An intuitive user interface empowers educators to easily generate rubrics based on their preferences, facilitating a more efficient, reliable, and fair grading process. This innovation not only saves time but also enhances the consistency and accuracy of assessments across various educational settings.

Relevance

In today's rapidly evolving educational environment, the ability to create accurate and consistent grading rubrics efficiently is crucial. The AI-driven Automated Rubric Generation System addresses this need by offering an advanced solution that leverages cutting-edge technologies to streamline the grading process. This system has the potential to revolutionize the way educators approach assessment, making it highly relevant for various educational institutions, from K-12 schools to universities. By enhancing productivity and ensuring fair assessments, this project contributes significantly to improving educational outcomes.

Problem Formulation

The AI-driven Automated Rubric Generation System project tackles several key challenges inherent in traditional grading methodologies:

- **Time-consuming manual rubric creation**: Educators often spend considerable time developing rubrics, detracting from instructional time and student interaction.
- Inconsistent grading standards: Manually created rubrics can lead to variability in grading standards, impacting fairness and reliability.
- **Difficulty in adapting rubrics to diverse contexts:** Creating rubrics that cater to varied subjects and educational levels requires substantial effort and expertise.
- **Limited scalability:** Traditional methods of rubric creation do not easily scale to accommodate growing student populations and diverse educational needs.

Motivation

The AI-driven Automated Rubric Generation System project is motivated by several factors:

- **Increasing demand for efficient grading solutions:** With the expansion of educational institutions, there is a pressing need for advanced tools that streamline the grading process.
- **Advancements in AI and machine learning:** Recent breakthroughs in AI and LLMs enable the development of sophisticated systems capable of automating complex tasks like rubric generation.
- **Potential for transformative impact:** By enabling educators to create accurate and consistent rubrics rapidly, the system can significantly improve grading efficiency and educational outcomes.

Scope of the Project

The scope of the AI-driven Automated Rubric Generation System project encompasses the following areas:

- **Development of a prototype system:** Create a robust prototype capable of generating rubrics from diverse educational contexts, including different subjects and grade levels.
- **Implementation of a scalable architecture:** Design a scalable architecture to accommodate large numbers of users and varying grading needs, ensuring optimal performance and responsiveness.
- **Integration of advanced AI and LLM algorithms:** Leverage cutting-edge AI and LLM algorithms for criteria definition and rubric creation, ensuring precision and adaptability.
- **Evaluation of the system's performance:** Conduct rigorous testing and validation, including benchmarking against traditional grading methods and gathering feedback from educators.
- **Provision of comprehensive documentation and user guides:** Develop detailed documentation and user guides to facilitate effective utilization of the system by educators and developers.

Data Modeling

The project employs a sophisticated data modeling approach by integrating Chroma, a robust vector database engine, with LLMs renowned for their natural language understanding capabilities. This integration enhances the system's ability to understand and adapt to diverse grading criteria, enabling accurate and contextually relevant rubric generation. By leveraging these technologies, the platform becomes more versatile and intelligent, improving its capability to interpret and respond to user requirements effectively.

Implementation Overview

The implementation stage of the AI-driven Automated Rubric Generation System project involves the integration of various technologies and APIs to create a robust and efficient system for automated rubric generation. This section provides an overview of the key components and processes involved in the implementation of the system.

Technologies and APIs Used: The deployment of the AI-driven Automated Rubric Generation System is performed using the Streamlit Python framework, known for its capability to build attractive and interactive user interfaces. By leveraging Streamlit, the system ensures that users have a seamless and engaging experience while interacting with the platform. The framework's versatility allows for the creation of visually appealing dashboards and intuitive controls, enhancing the overall usability and accessibility of the system.

Implementation Process

The implementation process of the AI-driven Automated Rubric Generation System can be divided into the following stages:

Streamlit Page Features and User Interaction: The implementation begins with utilizing Streamlit's page feature to create a user-friendly interface. One page is dedicated to selecting options and interacting with the chat, while the second page allows users to upload additional documents that provide reference and context for rubric generation.

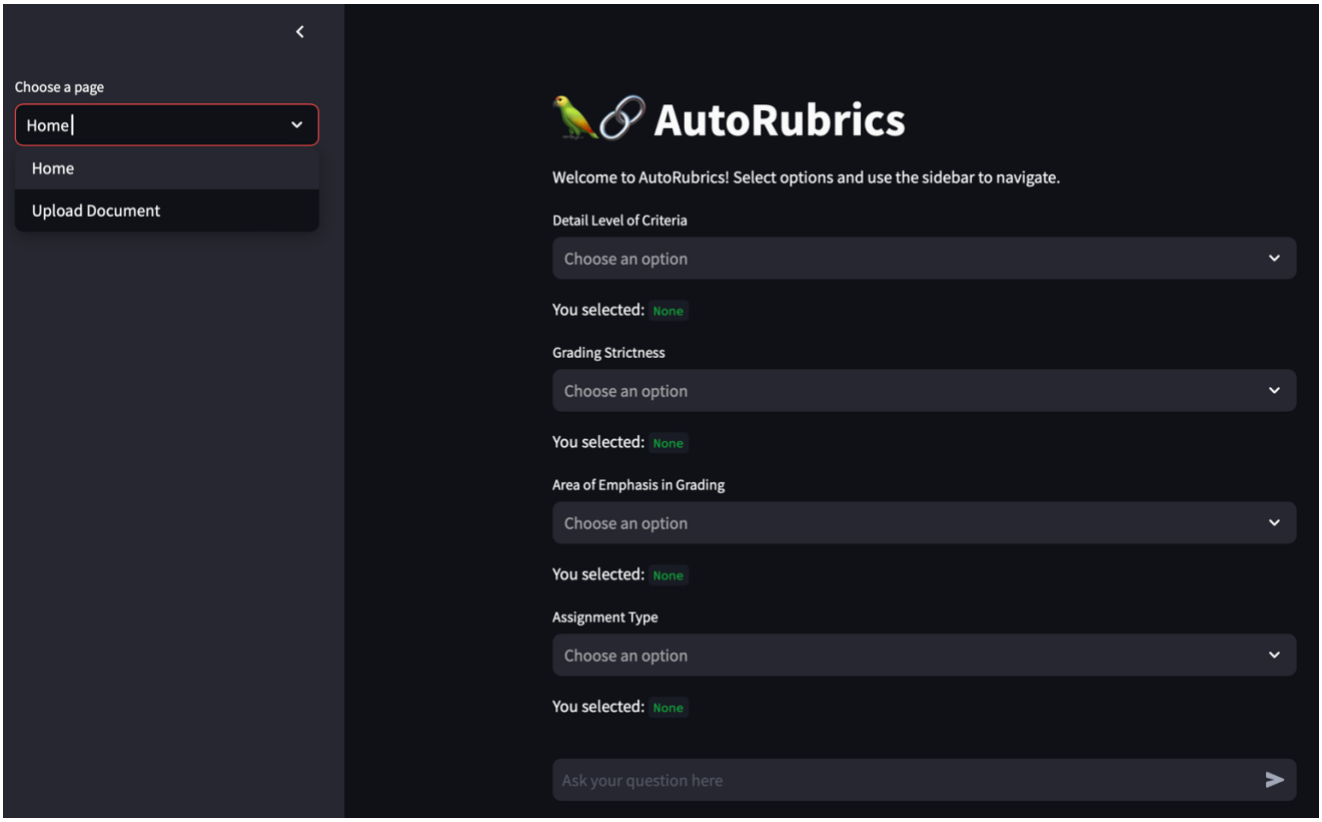

Image 1: Home page.

Collecting User Inputs: The selectbox feature of Streamlit is employed to gather user inputs necessary for generating rubrics. Users can select various options that define the criteria and preferences for their rubrics. This stage ensures that all required information is collected in an organized manner.

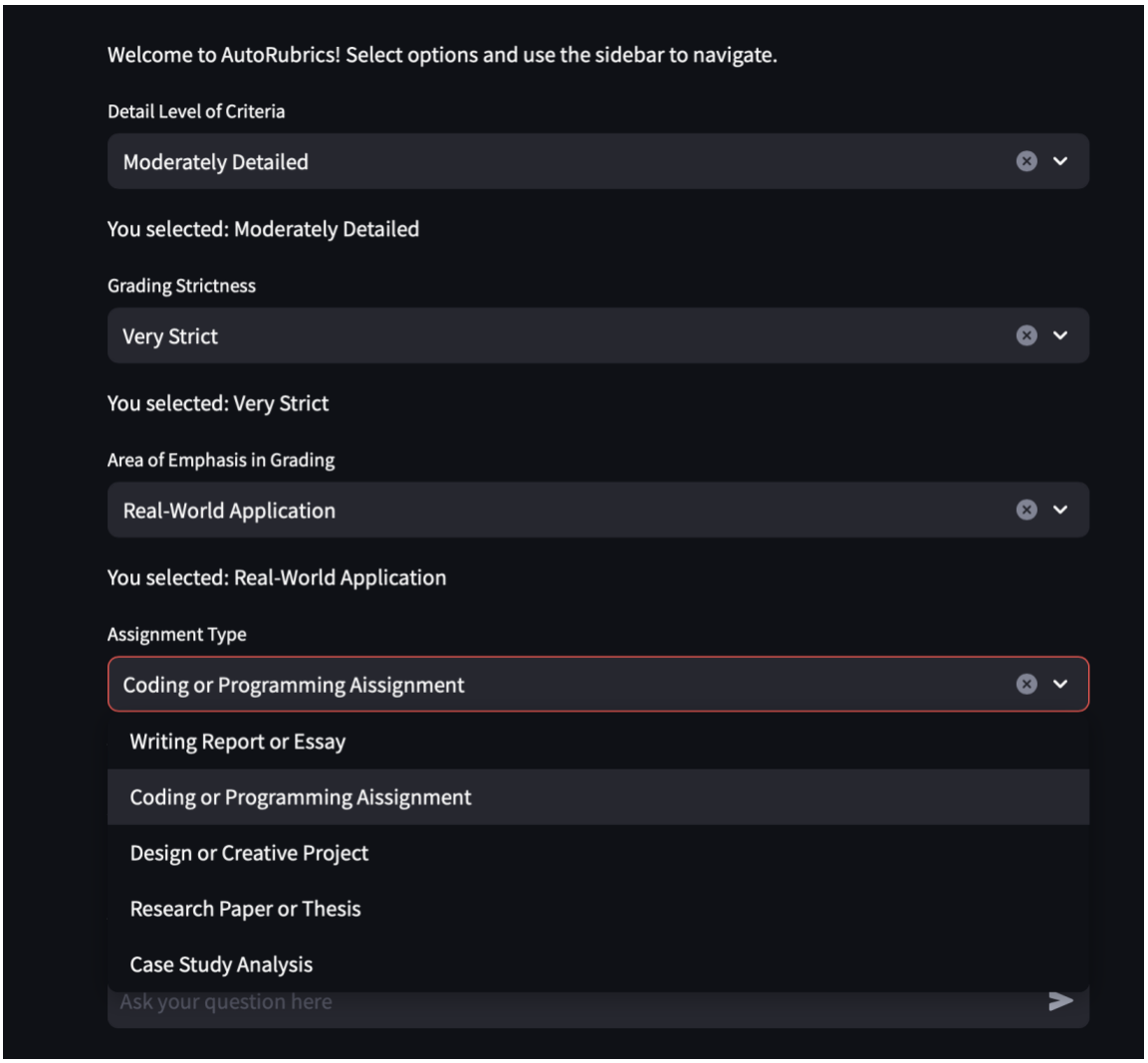

Image 2: Options Interface.

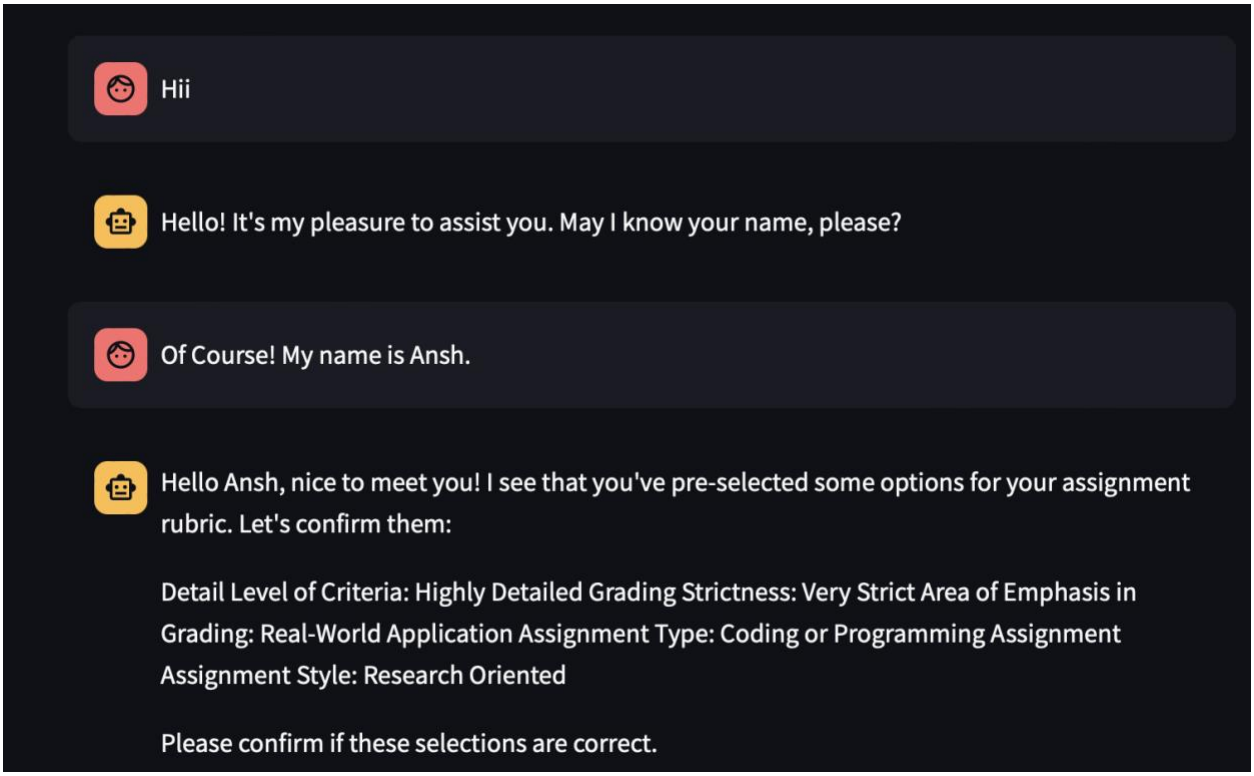

Image 3: Chat Interface – Part 1.

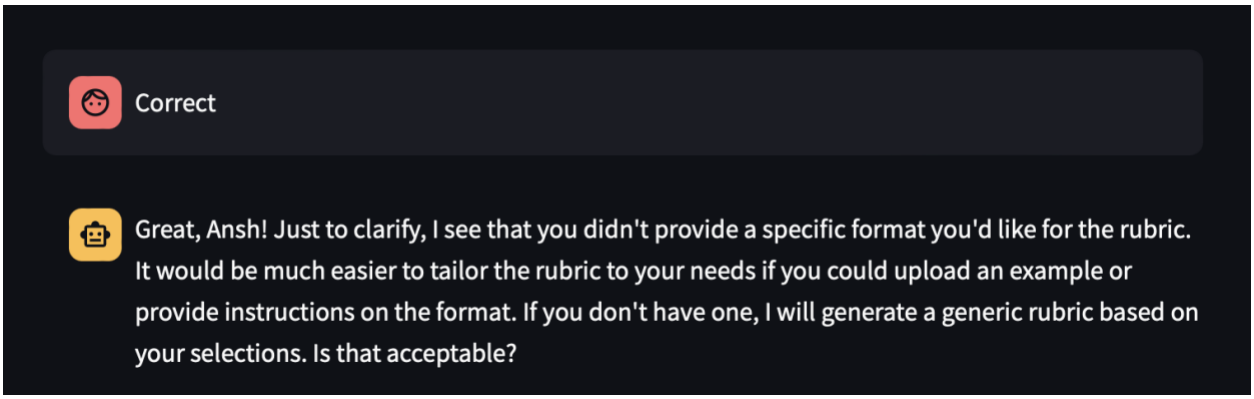

Image 4: Chat Interface – Part 2.

Image 5: Upload Page.

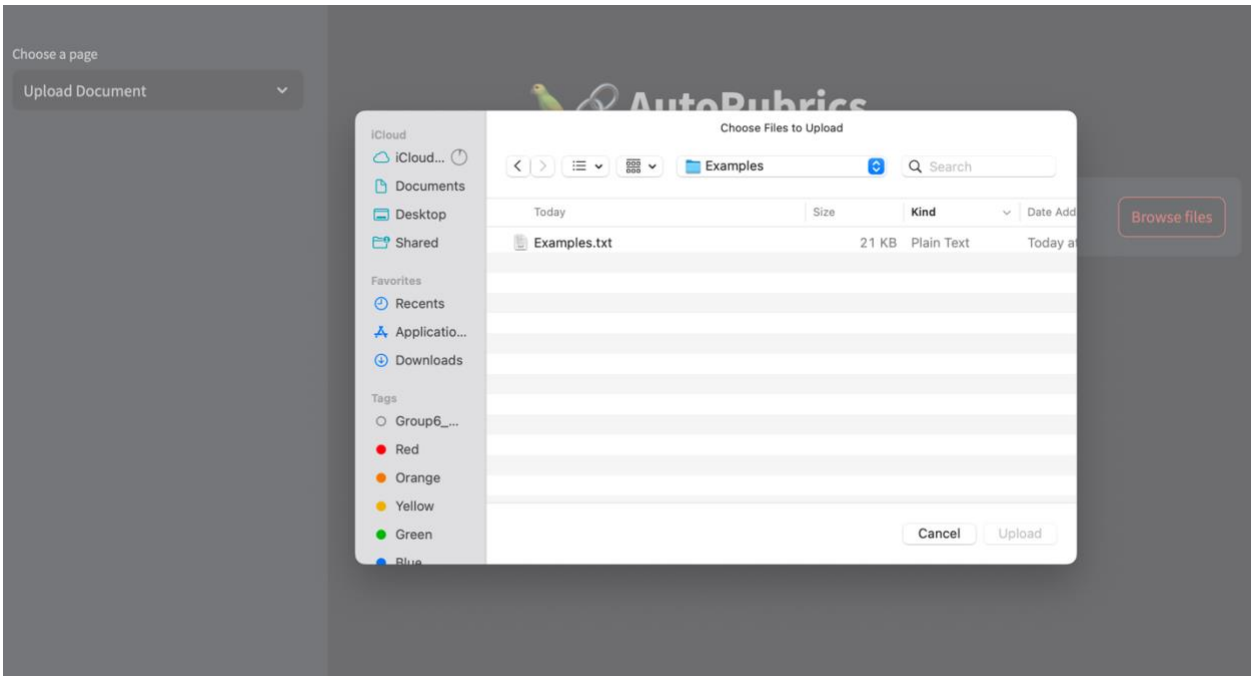

Image 6: File Uploader.

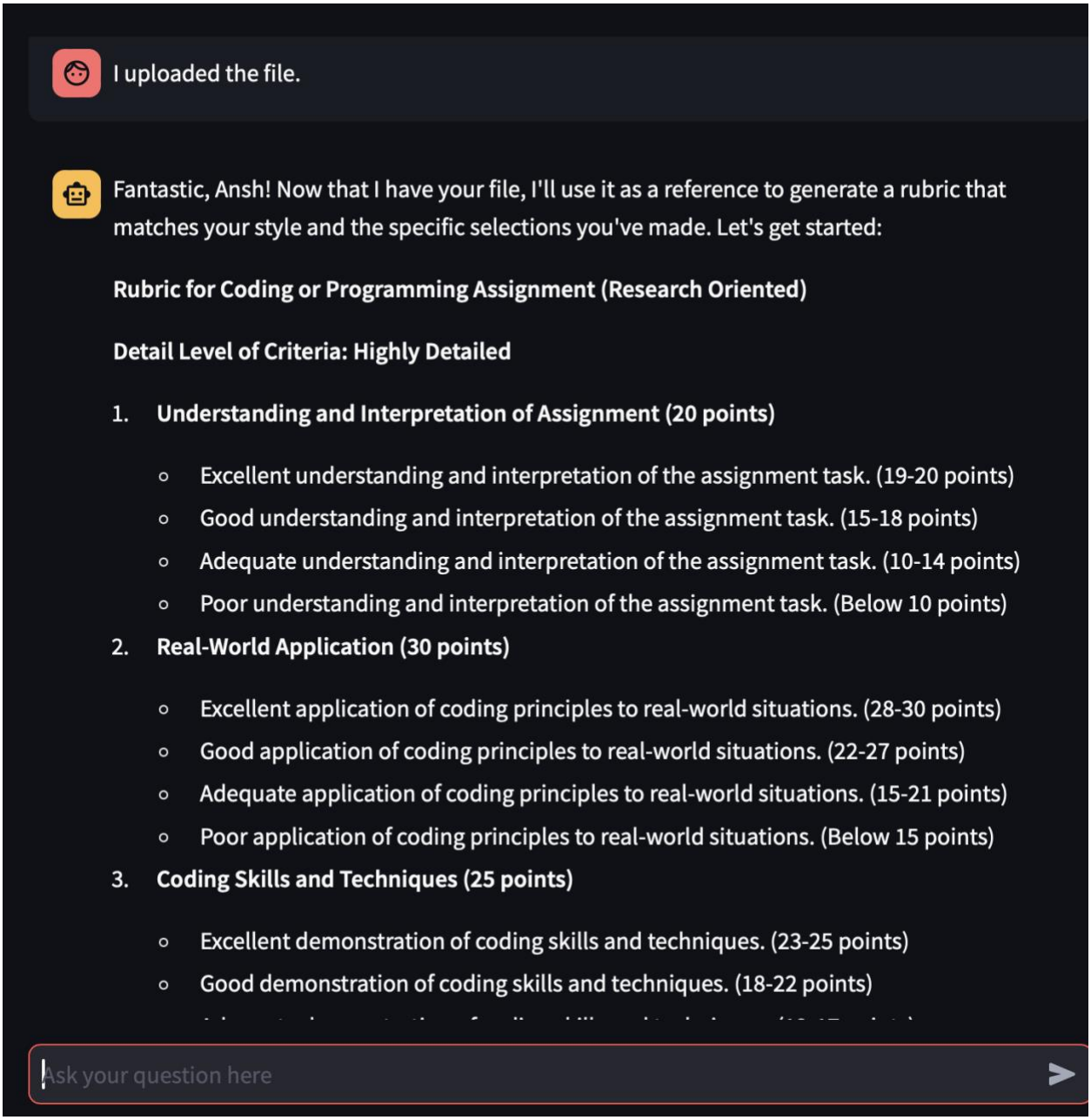

Image 7: Chat Interface – Part 3.

Processing Inputs and Generating Responses: The system processes inputs using two distinct chains. The first is a normal LLMChain that handles responses when no document is uploaded, providing immediate feedback based on user inputs. The second is a retrieval chain, which is activated when a document is uploaded, enabling the system to fetch relevant information from the document. Both chains are designed to generate accurate and contextually appropriate rubrics based on the processed inputs.

Prompt Crafting and Response Generation: Crafting effective prompts is a key aspect of the system's functionality. Prompts are structured to ensure clarity and relevance, guiding the model to generate concise and precise responses. The final answer, or rubric, is formulated based on the user inputs and any additional context provided by uploaded documents.

system_prompt = """

You are an expert in rubric generation for any given type of assignment.
Start by greeting the user respectfully, collect the name of the user.
The user has already selected {options} for the factors like Detail level of criteria, Grading strictness, Area of emphasis, Assignment type and Assignment style.
Verify these selections with user by displaying the options in the following format:

Detail Level of Criteria:
Grading Strictness:
Area of Emphasis in Grading:
Assignment Type:
Assignment Style:

After verifying all the options, generate a rubric referring to the format of examples and instructions provided in the {context}, make sure you use the same format.
If there is nothing available in {context}, suggest the user to upload one for better response.
Use the persona pattern to take the persona of the user and generate a rubric that matches their style.
Lastly, ask user if you want any modification or adjustments to the rubrics generated? If the user says no, then end the conversation.
Keep the chat history to have memory and not repeat questions.

chat history: {chat_history}

"""

Conclusion

The AI-driven Automated Rubric Generation System stands as a transformative solution in the realm of educational assessment. By leveraging advanced AI and LLM technologies, this project aims to streamline the grading process, enhance reliability, and ensure fairness in evaluations. With its user-friendly interface and scalable architecture, the system is poised to significantly improve educational productivity and outcomes. By addressing key challenges in traditional grading methods, this innovative approach promises to empower educators and enrich the educational experience for students.

Working of Algorithms (Appendix):

```
# Importing libraries and modules

import os
import re
import openai
import chardet
import os.path
import pathlib
import tempfile
import streamlit as st
from langchain.chat_models import ChatOpenAI
from langchain_community.document_loaders import TextLoader
from langchain_openai import OpenAIEmbeddings
from langchain_text_splitters import CharacterTextSplitter
from langchain.vectorstores import Chroma
from langchain.chains import LLMChain, create_retrieval_chain
from langchain.chains import LLMChain,RetrievalQA
from langchain.prompts.chat import ChatPromptTemplate
from langchain_core.runnables import RunnablePassthrough
__import__('pysqlite3')
import sys
sys.modules['sqlite3'] = sys.modules.pop('pysqlite3')
import sqlite3



## Set up the environment
# Load secret keys

secrets = st.secrets  # Accessing secrets (API keys) stored securely

openai_api_key = secrets["openai"]["api_key"]  # Accessing OpenAI API key from secrets
os.environ["OPENAI_API_KEY"] = openai_api_key  # Setting environment variable for
OpenAI API key
```

```
# Initialize session state variables

if "selected_option" not in st.session_state:
    st.session_state.selected_option = ["None"] * 5

if "messages" not in st.session_state:
    st.session_state.messages = []

if "uploaded_files" not in st.session_state:
    st.session_state.uploaded_files = None

if "vector_store" not in st.session_state:
    st.session_state.vector_store = None

if "chain" not in st.session_state:
    st.session_state.chain = None



# Load the document, split it into chunks, embed each chunk and load it into the vector
store.

def example_file(uploaded_files):
    detector = chardet.UniversalDetector()
    for uploaded_file in uploaded_files:
        # Display file details
        file_details = {"filename": uploaded_file.name, "filetype": uploaded_file.type}

        # Create temporary directory and save file there
        temp_dir = tempfile.mkdtemp()
        path = os.path.join(temp_dir, uploaded_file.name)
        with open(path, "wb") as f:
            f.write(uploaded_file.getvalue())

        with open(path, "rb") as f:
            for line in f:
                detector.feed(line)
                if detector.done:
                    break

        detector.close()
        encoding = detector.result['encoding']

        raw_documents = TextLoader(path,encoding = encoding).load()
```

```
    text_splitter = CharacterTextSplitter(chunk_size=1000, chunk_overlap=0)
    documents = text_splitter.split_documents(raw_documents)
    db = Chroma.from_documents(documents, OpenAIEmbeddings())

  return db



def  get_chain(options,context,chat_history):

  system_prompt = """

    You are an expert in rubric generation for any given type of assignment.

    Start by greeting the user respectfully, collect the name of the user.
    The user has already selected {options} for the factors like Detail level of
    criteria, Grading strictness, Area of emphasis, Assignment type and Assignment style.
    Verify these selections with user by displaying the options in the following format:

    Detail Level of Criteria:
    Grading Strictness:
    Area of Emphasis in Grading:
    Assignment Type:
    Assignment Style:

    After verifying all the options, generate a rubric referring to the format of examples and
instructions provided in the {context}, make sure you use the same format.
    If there is nothing available in {context}, suggest the user to upload one for better
response.
    Use the persona pattern to take the persona of the  user and generate a rubric that
matches their style.
    Lastly, ask user if you want any modification or adjustments to the rubrics generated? If
the user says no then end the conversation.
    Keep the chat history to have memory and not repeat questions.

    chat history: {chat_history}

    """

  prompt = ChatPromptTemplate.from_messages(
    [("system", system_prompt), ("human", "{input}")]
  )
```

```
    prompt.format_messages(input = "query", options = "st.session_state.selected_option",
    context = "st.session_state.vector_store", chat_history = "chat_history")

    model_name = "gpt-4"
    llm = ChatOpenAI(model_name=model_name)

    chain = LLMChain(llm=llm, prompt=prompt)

    if st.session_state.vector_store:
        retriever = context.as_retriever()
        chain = create_retrieval_chain(retriever, chain)

    st.session_state.chat_active = True

    st.session_state.chain = chain

    return st.session_state.chain


def get_answer(query):
            chains =
            get_chain(st.session_state.selected_option,st.session_state.vector_store,chat_hist
            ory)

            response = chains.invoke({"input": query, "options":
            st.session_state.selected_option, "context" : st.session_state.vector_store,
            "chat_history": chat_history})

             try:
                        answer = response['text']
            except:
                        ans = response['answer']
                        answer = ans['text']

            return answer


def format_chat_history(messages):
    formatted_history = ""
    for message in messages:
        role = "user" if message["role"] == "user" else "Assistant"
        content = message["content"]
        formatted_history += f"{role}: {content}\n"
    return formatted_history
```

```
def select_option():

    options_1 = ("None", "Broad Overview", "Moderately Detailed", "Highly Detailed")
    options_2 = ("None", "Linient","Somewhat Linient","Moderate","Very Strict")
    options_3 = ("None", "Technical Accuracy","Depth of Analysis","Clarity and
Creativity","Real-World Application","Problem Solving Skills")
    options_4 = ("None", "Writing Report or Essay","Coding or Programming
Aissignment","Design or Creative Project","Research Paper or Thesis","Case Study Analysis")
    options_5 = ("None", "Research Oriented","Problem Solving","Case
Studies","Presentations","Experiential Learning","Literature Reviews","Reflective Journals")

  # Maintain index based on previous selections if they exist
    option_1 = st.selectbox(
        "Detail Level of Criteria",
        options_1,
        index=options_1.index(st.session_state.selected_option[0]) if
len(st.session_state.selected_option) > 0 else 0
    )
    st.session_state.selected_option[0:1] = [option_1]

    option_2 = st.selectbox(
        "Grading Strictness",
        options_2,
        index=options_2.index(st.session_state.selected_option[1]) if
len(st.session_state.selected_option) > 1 else 0
    )
    st.session_state.selected_option[1:2] = [option_2]

    option_3 = st.selectbox(
        "Area of Emphasis in Grading",
        options_3,
        index=options_3.index(st.session_state.selected_option[2]) if
len(st.session_state.selected_option) > 2 else 0
    )
    st.session_state.selected_option[2:3] = [option_3]

    option_4 = st.selectbox(
        "Assignment Type",
        options_4,
        index=options_4.index(st.session_state.selected_option[3]) if
len(st.session_state.selected_option) > 3 else 0
    )
    st.session_state.selected_option[3:4] = [option_4]
```

```
    option_5 = st.selectbox(
        "Assignment Style",
        options_5,
        index=options_5.index(st.session_state.selected_option[4]) if
len(st.session_state.selected_option) > 4 else 0
    )
    st.session_state.selected_option[4:5] = [option_5]

    return st.session_state.selected_option



# Title for the web app
st.title("🚀📎 AutoRubrics")

# Multi-page navigation
page = st.sidebar.selectbox("Choose a page", ["Home", "Upload Document"])


if page == "Home":
    st.write("Welcome to AutoRubrics! Select options and use the sidebar to navigate.")


    st.session_state.selected_option = select_option()

    if st.session_state.selected_option:
        # Display chat messages from history on app rerun
        for message in st.session_state.messages:
            with st.chat_message(message["role"]):
                st.markdown(message["content"])

        if query := st.chat_input("Ask your question here"):
            # Display user message in chat message container
            with st.chat_message("user"):
                st.markdown(query)
            # Add user message to chat history
            st.session_state.messages.append({"role": "user", "content": query})

            chat_history = format_chat_history(st.session_state.messages)

            answer = get_answer(query)

            # Display assistant response in chat message container
            with st.chat_message("assistant"):
```

```
            st.markdown(answer)
        # Add assistant response to chat history
        st.session_state.messages.append({"role": "assistant", "content": answer})

        # Button to clear chat messages
        def clear_messages():
            st.session_state.messages = []
        st.button("Clear", help = "Click to clear the chat", on_click=clear_messages)


if page == "Upload Document":

    uploaded_files = st.file_uploader("Choose a file", accept_multiple_files=True)

    if uploaded_files:
        st.session_state.uploaded_files = uploaded_files
        st.session_state.vector_store = example_file(st.session_state.uploaded_files)
        st.write("Documents uploaded successfully.")

    if st.session_state.uploaded_files:
        for uploaded_file in st.session_state.uploaded_files:
            st.write("Uploaded files:", uploaded_file.name)
```