Proceedings

Introducing a chatbot on the Web Portal of a Higher Education Institution to enhance student interaction

Pedro Filipe Oliveira [1,*,‡] and Paulo Matos [2,‡]

[1] Research Centre in Digitalization and Intelligent Robotics (CeDRI), Laboratório Associado para a Sustentabilidade e Tecnologia em Regiões de Montanha (SusTEC), Instituto Politécnico de Bragança, Campus de Santa Apolónia, 5300-253 Bragança, Portugal; poliveira@ipb.pt
[2] Research Centre in Digitalization and Intelligent Robotics (CeDRI), Laboratório Associado para a Sustentabilidade e Tecnologia em Regiões de Montanha (SusTEC), Instituto Politécnico de Bragança, Campus de Santa Apolónia, 5300-253 Bragança, Portugal; pmatos@ipb.pt
* Correspondence: poliveira@ipb.pt;
‡ These authors contributed equally to this work.

Abstract: This paper introduces the implementation of a chatbot on the web portal of a higher education institution, aiming to enhance student interaction and provide seamless access to information and support services. With the increasing reliance on digital platforms for student engagement, a chatbot offers a user-friendly and efficient means of communication, catering to the diverse needs of students in a higher education setting. The chatbot developed utilizes natural language processing, machine learning, and artificial intelligence algorithms to engage in dynamic conversations with students. We use Large Language Models (LLMs), because this and vector databases are revolutionizing the way we handle and retrieve complex data structures. Its main objective is to provide instant responses, personalized guidance, and timely support for various aspects of student life within the institution namely: Information Retrieval, the chatbot acts as a virtual collaborator, offering quick and accurate responses to frequently asked questions regarding admissions, programs, course registration, financial aid, and campus facilities, reducing the need for manual information searches; Academic Support the chatbot assists students in academic matters, such as course selection, prerequisites, graduation requirements, and study resources. It can offer personalized recommendations based on the student's academic profile and preferences; Campus Services, provides information about campus services, extracurricular activities, events, and resources; Appointment Scheduling, facilitates appointment scheduling with academic advisors, and support staff, streamlining the administrative processes and ensuring timely access to guidance and assistance. This development follows a user-centric approach, incorporating feedback from students, faculty, and administrators to ensure that the chatbot meets their specific needs and preferences. Rigorous testing and quality assurance measures are implemented to guarantee the accuracy, reliability, and security of the chatbot. In conclusion, we achieve a functional chatbot with a medium computational heavy, in this way it can be practical to use it in real time by the students at the institution web portal. The introduction of a chatbot on the web portal of a higher education institution represents a significant advancement in facilitating student interaction and support services. By providing instant and personalized responses, the chatbot streamlines communication, reduces response times, and empowers students to find information and resources efficiently. As the chatbot technology continues to evolve, ongoing enhancements and refinements will ensure that it remains a valuable tool in enhancing student experiences, promoting engagement, and fostering a positive learning environment within the institution.

Keywords: chatbot; machine-learning; nlp; artificial-intelligence; students.

Citation: Oliveira, P.; Matos, P. Introducing a chatbot on the Web Portal of a Higher Education Institution to enhance student interaction. *Eng. Proc.* **2023**, *1*, 0. https://doi.org/

Published:

1. Introduction

In the digital age, higher education institutions face evolving challenges in engaging and supporting their student populations. With the increasing prevalence of online learning

and the diverse needs of modern students, the demand for efficient and responsive student services has never been greater. One innovative solution that has gained significant traction in recent years is the integration of chatbots into web portals [1].

This paper explores the implementation of a chatbot within the web portal of a higher education institution, with a primary focus on enhancing student interaction and improving the overall student experience. Chatbots, powered by natural language processing (NLP) and artificial intelligence (AI), have emerged as versatile tools capable of addressing a wide array of inquiries, from answering frequently asked questions to providing personalized academic and administrative support.

2. Methods

Creating a natural language chatbot (LLM - Natural Language) [2] can be a challenging task, but it is a powerful technology for improving interaction with users and automating customer service tasks, technical support, among others. For do that, we have followed the following steps to create a chatbot with LLM:

- Define the Purpose and Objectives: This will help guide development and ensure the chatbot is useful;
- Choose a Platform or Framework: There are several options available to develop chatbots with LLM, namely:
  - *Dialogflow* (*Google*): A platform from Google that uses Google Cloud NLP;
  - *Microsoft Bot Framework*: A Microsoft platform for creating chatbots;
  - *Rasa*: An open source framework for chatbot development with LLM;
  - *IBM Watson Assistant*: An IBM solution for creating chatbots.
- Training and Data: For the chatbot to properly understand and respond to user queries, it has to be trained with relevant data;
- Channel Integration: It can be on a website, mobile app, social media, etc. It is important to integrate the chatbot with relevant channels;
- Development and Configuration: Use the chosen platform or framework to develop the chatbot. Configure the conversation flow, responses and actions;
- Test and Tune: Test the chatbot exhaustively to identify issues and adjust its responses and behavior. Make sure it understands questions and responds appropriately;
- Deployment: After testing, the chatbot must be deployed;
- Continuous Improvements: An LLM chatbot can be improved over time as more data and feedback from users is collected;
- Support and Maintenance: The chatbot must be continuously updated and provide ongoing support to deal with user issues and queries.

Creating a chatbot with LLM requires programming and natural language skills, as well as natural language processing capabilities such as machine learning models and data training [3]. On this work we're focusing on building a chatbot using *Streamlit*, an open-source application framework that's a big hit among Machine Learning and Data Science enthusiasts. Not only that - we'll also explore how to integrate with *LangChain* and various language models, including LLM models.

2.1. Concepts

What is *Streamlit*?

Streamlit is a fast, easy, and fun open source tool for building web applications. It is designed to help machine learning engineers and data scientists build interactive web applications around their projects without the need for web development knowledge. Streamlit's simplicity and speed make it an excellent choice for building a chatbot interface [4].

What is a Chatbot?

A chatbot is a software application designed to conduct online chat conversations through text or text-to-speech, rather than providing direct contact with a human attendant

in real time. Designed to convincingly simulate human behavior as a conversational partner, chatbot systems often require continuous tuning and testing, and many in production are still unable to properly conduct conversations or pass the industry standard, the Turing test.

2.2. Environment configuration

Before we start the chatbot building, we must configure the development environment with the needed libraries and tools [5]. To ensure that the code runs successfully and that the chatbot works well. At the *streamlit_requir.txt* file we have a list of libraries/tools that this project needs. This list includes:

- *streamlit* -> Library to create interactive web applications for projects that use machine learning and data science;
- *streamlit_chat* -> This component is used to create user interfaces;
- *langchain* -> Framework used to develop applications that use language models. This provide a regular interface to use chains, integration's with different tools, and end-to-end chains for regular applications;
- *sentence_transformers* -> Library to allow the use of transformer models (*BERT*, *RoBERTa*, and others), and generate text semantic representations (embeddings), used for document indexation;
- *openai* -> *OpenAI's* library which allow to use its language models, (*GPT-3.5-turbo*), and generate human like text;
- *unstructured* and *unstructured[local-inference*] -> Libraries used for document processing and unstructured data management;
- *pinecone-client* -> Client for *Pinecone*, a service vector database which allow to do similarity searches at vector data.

For theses libraries installation process, we can run the following command at a terminal window: This command will instructs "*pip*" (package installer) to install all the libraries that are previously defined in the *streamlit_requir.txt* file.

Listing 1. Streamlit libraries installation

```
pip install -r streamlit_requir.txt
```

3. Results and Discussion

These results are related, to a specific use case implemented at the Web Portal of a Higher Education. Following, we detail all the necessary steps to develop the chatbot.

Document Indexing

Following to build the chatbot involves to prepare and index the documents that the chatbot will use to answer the queries. We going to use *the indexing_documents.py* script.

Loading documents from a directory with *LangChain*

First step at the script (*indexing_documents.py*) involves to load the documents from a directory. For that we use the (*DirectoryLoader*) class provided by *LangChain*. This class receives a directory as input and it will load all the documents the folder contains.

Listing 2. Loading documents

```
from langchain.document_loaders import DirectoryLoader
directory_to_load = '/content_info/data'
def load_documents(directory_to_load):
loader = DirectoryLoader(directory_to_load)
loaded_docs = loader.load()
return loaded_docs
loaded_docs = load_documents(directory_to_load)
len(loaded_docs)
```

Documents splitting

After the documents loading, the script continues to divide each document into smaller pieces. The chunks size and the overlap between them is defined by the user. We done this, to ensure that the documents size can be managed and that no important information is lost in the splitting process. For this, *LangChain's RecursiveCharacterTextSplitter* class is used.

Listing 3. Documents splitting

```
from langchain.text_splitter import RecursiveCharacterTextSplitter
def split_documents(docs, size_chunk=500, overlap_chunk=20):
splitter_text = RecursiveCharacterTextSplitter(size_chunk=size_chunk, overlap_chunk=overlap_chunk)
documents = splitter_text.split_documents(docs)
return documents
documents = split_documents(docs)
print(len(documents))
```

Embeddings creation

After the documents are broken down, we must convert these text chunks into a format that the AI model understand. We do this, creating text embeddings using the class (*SentenceTransformerEmbeddings*) foreseen by *LangChain*.

Listing 4. Embeddings creation

```
from langchain.embeddings import SentenceTransformerEmbeddings
embeddings_txt = SentenceTransformerEmbeddings(mdel_name="all-MiniLM-L6-v2")
```

Embeddings storing using *Pinecone*

After embeddings creation, we need store them at a location where they can be accessed and searched. *Pinecone*, a vector database service that can suit this task. An example code is as follows:

Listing 5. Embeddings storing using *Pinecone*

```
from langchain.pinecone import PineconeIndexer
def index_embeddings(embeddings, documents):
indexer = PineconeIndexer(api_key='your-api-key-from-pinecone', index_name='your-index-name')
indexer.index(embeddings, documents)
index_embeddings(embeddings, documents)
```

This script creates an index in *Pinecone* and stores the embeddings along with the corresponding text. Now, whenever a user asks a question, the chatbot can search this index for the most similar text and return the corresponding answer.

Creating the chatbot Interface with *Streamlit*

With our documents indexed and ready to be searched, we can now focus on creating the chatbot interface. *Streamlit* provides a simple and intuitive way to create interactive web applications, and is perfect for our chatbot interface.

Streamlit Chat Component

Streamlit's chat component is a new way to create chatbots. It provides a chat app-like interface, making a chatbot deployed on *Streamlit* with a cool interface. To use this component, you need to install it separately using *pip*:

Listing 6. Install *streamlit-chat* component

```
pip install streamlit-chat
```

After installation, you can import it into your *Streamlit* app:

Listing 7. Streamlit import

```
import streamlit as stlit
from streamlit_chat import chat
@st.cache(allow_output_mutation=True)
def get_chat():
return chat()
chat = get_chat()
```

This code creates a new chat interface in the *Streamlit* app. We can add messages to this chat using the *add_message* method:

Listing 8. Chat interface creation

```
chat.add_message("Hi, how can i help you today?", "bot")
```

Integrating *Chatbot* with *LangChain*

LangChain is a framework to develop applications that can use language models. It has a standard interface for chains, many other tools integration, and end-to-end chains for regular applications. To integrate our chatbot with *LangChain*, we need to modify the *load_chain* function in *main.py*.

Listing 9. Chatbot integration

```
from langchain import LangChain
def load_chain_openai():
chain_openai = LangChain(api_key='your-api-key-from-openai')
return chain_openai
chain_openai = load_chain_openai()
```

This code creates a new instance of *LangChain* with your *OpenAI* API key. You can use this instance to generate responses to user queries [6].

At figure 1, it can be seen the architecture that supports the developed chatbot.

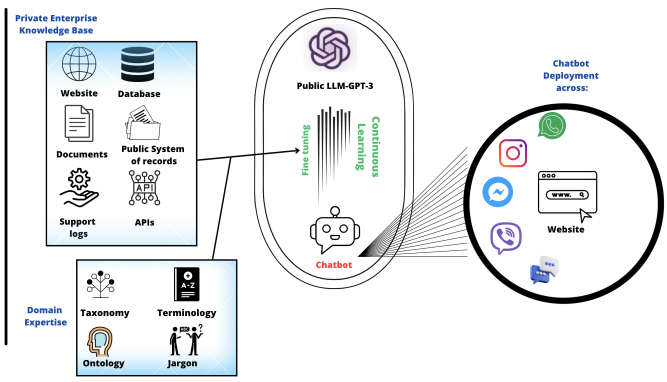

Figure 1. Chatbot - system architecture.

4. Conclusions

The introduction of a chatbot has the potential to revolutionize the student experience in higher education. Its ability to provide instant responses to common queries, offer personalized guidance, and facilitate seamless access to important resources can significantly alleviate the burden on administrative staff while empowering students to navigate the academic journey more efficiently.

One of the most notable advantages of a well-implemented chatbot is its availability round the clock, ensuring that students have access to information and support when they need it most, irrespective of time zones or office hours. This accessibility aligns with the expectations of the digital-native generation and demonstrates an institution's commitment to providing modern, responsive services.

However, the successful deployment of a chatbot does not come without challenges. Privacy and data security concerns must be addressed meticulously, and ongoing maintenance and refinement are essential to ensure that the chatbot remains a valuable asset. Moreover, while chatbots excel at handling routine queries, the human touch remains indispensable for more complex and nuanced issues.

Clearly, some limitations can be identified, namely the continuous needed of the input information (websites, documents).

As future work, we continue to test the chatbot, retrieving feedback from users, to a continuous tuning of the chatbot performance. And also the continuous upgrade of the input information.

Author Contributions: Conceptualization, P.O. and P.M.; methodology, P.O.; software, P.M.; validation, P.O. and P.M.; formal analysis, P.M.; investigation, P.O.; resources, P.M.; data curation, P.O.; writing—original draft preparation, P.O.; writing—review and editing, P.M.; visualization, P.O.; supervision, P.M.;. All authors have read and agreed to the published version of the manuscript.

Funding: The authors are grateful to the Foundation for Science and Technology (FCT, Portugal) for financial support through national funds FCT/MCTES (PIDDAC) to CeDRI (UIDB/05757/2020 and UIDP/05757/2020) and SusTEC (LA/P/0007/2021).

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results

References

1. Sjöström, J.; Dahlin, M. Tutorbot: A chatbot for higher education practice. In Proceedings of the Designing for Digital Transformation. Co-Creating Services with Citizens and Industry: 15th International Conference on Design Science Research in Information Systems and Technology, DESRIST 2020, Kristiansand, Norway, December 2–4, 2020, Proceedings 15. Springer, 2020, pp. 93–98.
2. Hien, H.T.; Cuong, P.N.; Nam, L.N.H.; Nhung, H.L.T.K.; Thang, L.D. Intelligent assistants in higher-education environments: the FIT-EBot, a chatbot for administrative and learning support. In Proceedings of the Proceedings of the 9th International Symposium on Information and Communication Technology, 2018, pp. 69–76.
3. Sandoval, Z.V. Design and implementation of a chatbot in online higher education settings. *Issues in Information Systems* **2018**, *19*.
4. Taecharungroj, V. "What Can ChatGPT Do?" Analyzing Early Reactions to the Innovative AI Chatbot on Twitter. *Big Data and Cognitive Computing* **2023**, *7*, 35.
5. Lappalainen, Y.; Narayanan, N. Aisha: A Custom AI Library Chatbot Using the ChatGPT API. *Journal of Web Librarianship* **2023**, pp. 1–22.
6. Fan, L.; Lafia, S.; Li, L.; Yang, F.; Hemphill, L. DataChat: Prototyping a Conversational Agent for Dataset Search and Visualization. *arXiv preprint arXiv:2305.18358* **2023**.