Monthly Progress Report - Infosys Springboard Virtual Internship 6.0

Prepared by : SHAMRUTHA S - BATCH 5

1.0 Introduction

- This report outlines the key concepts, technical skills, and practical applications covered during the first month of the virtual internship program. The curriculum provided a comprehensive overview of foundational Artificial Intelligence, moving rapidly into advanced Natural Language Processing (NLP) concepts and practical development.

2.1 Foundational Artificial Intelligence

- The initial sessions reviewed the fundamentals of AI. This included:
- **Core Definitions:** Differentiating between Artificial Intelligence, Machine Learning (ML), and Deep Learning (DL).
- **Key Concepts:** Understanding the building blocks of AI, such as models, training data, and algorithms.
- **AI Landscape:** Discussing the current state of AI and its various sub-fields (e.g., Computer Vision, NLP).

2.2 Deep Dive: LLM Architectures and Concepts

- A significant portion of the training was dedicated to Large Language Models (LLMs), the technology powering modern generative AI. Key topics included:
- **The Transformer Architecture:** A high-level overview of the 'Transformer' model, which is the basis for models like GPT and BERT.
- **Attention Mechanisms:** Learning the core concept of 'self-attention' and how it allows models to weigh the importance of different words in a sentence.
- **Tokenization:** Understanding the process of breaking text down into 'tokens' for the model to process.
- **Key Points:** Discussed concepts like prompt engineering, model parameters, and the emergent abilities of large-scale models.

2.3 NLP Word Vectorization Techniques

- To understand how models "read" text, we explored word vectorization—the process of turning words into numbers (vectors).
- **Basic Techniques:** Reviewed traditional methods like **TF-IDF** (Term Frequency-Inverse Document Frequency).
- **Advanced Embeddings:** Focused on predictive models like **Word2Vec** and **GloVe**, which capture semantic meaning and context (e.g., 'King' - 'Man' + 'Woman' = 'Queen').

3.1 Introduction to Streamlit

- We were introduced to Streamlit, a powerful Python library for rapidly building and deploying data applications.
- **Core Functionality:** Learned how to use Streamlit's simple, script-based approach to create interactive web UIs.
- **Key Components:** Gained experience with widgets like sliders, buttons, text inputs, and layout containers.

3.2 Project: Streamlit Chatbot UI

- The ClauseEase AI based Contract Language Simplifier project for this module was to synthesize our learnings by building a functional Chatbot UI.
- **Objective:** To create an interactive, front-end interface for a future chatbot.
- **Implementation:** The project involved:
- Setting up a Streamlit environment.
- Using st.text_input for user queries and st.button to submit.
- Managing conversation history (session state) to display both user and bot messages.
- Styling the chat elements to create a clear, message-based layout.
- **Outcome:** A functional, locally-hosted web application that successfully mimics a modern chatbot interface, ready for a back-end LLM to be integrated.

CODE:

```
import streamlit as st

import time

st.set_page_config(

page_title="Chatbot with Upload",

page_icon="   ",

layout="wide"

)

st.markdown("<h1 style='text-align: center;'>Chatbot</h1>", unsafe_allow_html=True)

if "chat_sessions" not in st.session_state:

st.session_state.chat_sessions = {}

if "current_chat_id" not in st.session_state:
```

```
st.session_state.current_chat_id = None
if "last_processed_file" not in st.session_state:
st.session_state.last_processed_file = {}
def create_new_chat():
chat_id = f"chat_{int(time.time())}"
st.session_state.chat_sessions[chat_id] = [
{"role": "assistant", "content": "Hello! How can I help you today?"}]
st.session_state.current_chat_id = chat_id
st.session_state.last_processed_file[chat_id] = None
st.rerun()
with st.sidebar:
st.header("Controls")
if st.button("➕ New Chat", use_container_width=True):
create_new_chat()
    st.header("Search")
search_query = st.text_input("Search messages (coming soon!)")
st.header("Upload File")
uploader_key = f"uploader_{st.session_state.current_chat_id}"
uploaded_file = st.file_uploader(
"Upload an image or document",
type=["png", "jpg", "jpeg", "pdf", "txt", "md", "docx"],
key=uploader_key,
disabled=(st.session_state.current_chat_id is None)
st.header("Chat History")
if not st.session_state.chat_sessions:
```

```
st.caption("No chat history yet.")
else:
sorted_chat_ids = sorted(st.session_state.chat_sessions.keys(), reverse=True)
for chat_id in sorted_chat_ids:
messages = st.session_state.chat_sessions[chat_id]
title = "New Chat"
for msg in messages:
if msg["role"] == "user":
title = msg["content"][:30] + "..."
break
if st.button(title, key=chat_id, use_container_width=True):
st.session_state.current_chat_id = chat_id
st.rerun()
if st.session_state.current_chat_id is None:
if not st.session_state.chat_sessions:
create_new_chat()
else:
st.write("Click 'New Chat' or select a chat from the history to begin.")
else:
current_chat_id = st.session_state.current_chat_id
current_messages = st.session_state.chat_sessions[current_chat_id]
if uploaded_file is not None:
file_identifier = f"{uploaded_file.name}_{uploaded_file.size}"
if st.session_state.last_processed_file.get(current_chat_id) != file_identifier:
file_name = uploaded_file.name
```

```
user_file_message = f"I've uploaded a file: `{file_name}`"

if uploaded_file.type.startswith("image/"):

img_data = uploaded_file.getvalue()

current_messages.append({

"role": "user",

"content": user_file_message,

"type": "image",

"data": img_data})

else:

current_messages.append({"role": "user", "content": user_file_message})

assistant_response = f"Thanks! I've received `{file_name}`. (This is a demo, I
can't process it yet)."

current_messages.append({"role": "assistant", "content": assistant_response})

st.session_state.last_processed_file[current_chat_id] = file_identifier

st.rerun()

for message in current_messages:

avatar = "       " if message["role"] == "user" else "    "

with st.chat_message(message["role"], avatar=avatar):

st.markdown(message["content"])

if message.get("type") == "image":

st.image(message["data"], width=200)

if prompt := st.chat_input("What is up?"):

current_messages.append({"role": "user", "content": prompt})

with st.chat_message("user", avatar="       "):

st.markdown(prompt)

with st.chat_message("assistant", avatar="    "):
```

```
message_placeholder = st.empty()
full_response = ""
assistant_response = f"Echo: {prompt}"
for chunk in assistant_response.split():
full_response += chunk + " "
time.sleep(0.05)
message_placeholder.markdown(full_response + "▌ ")
message_placeholder.markdown(full_response)
current_messages.append({"role": "assistant", "content": full_response})
st.rerun()
```

OUTPUT: