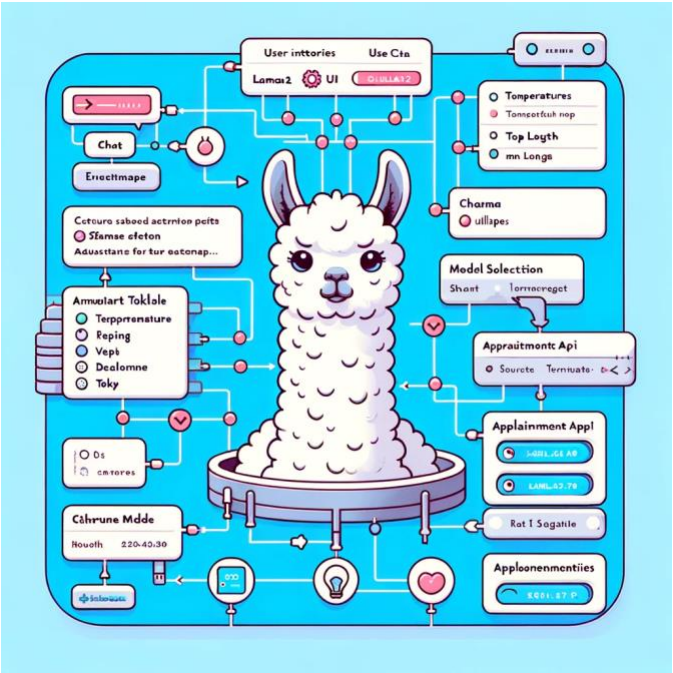
Comprehensive Project Report: Abhi's Llama Chatbot By Abhishek Nandakumar, LinkedIn

Anandak5@asu.edu



Executive Overview:

Abhi's Llama Chatbot is a cutting-edge conversational AI platform which was more of a pet project, engineered to deliver a sophisticated dialogue experience via a streamlined web interface. This platform harnesses the potent capabilities of the Llama2 language models through the Replicate API, encapsulated within a Streamlit-based frontend environment. Initially conceived for local deployment, the architecture of this project inherently supports seamless migration to cloud infrastructures, thereby amplifying its scalability and global accessibility.

Project Genesis:

The genesis of Abhi's Llama Chatbot was driven by an ambition to democratize access to advanced conversational AI technologies. The core ethos of this initiative is to empower users to seamlessly engage with AI entities, thereby fostering a dynamic interactive ecosystem. This is underpinned by a robust framework that allows for extensive customization of the AI's conversational parameters, ensuring a tailored user experience.

System Architecture and Design:

The architecture of Abhi's Llama Chatbot is bifurcated into a frontend user interface and a backend AI model processing unit:

1) Frontend User Interface:

Developed using Streamlit, the frontend serves as the conduit for user interactions, enabling real-time communication with the AI model. This component is meticulously designed to ensure intuitive navigation, allowing users to effortlessly configure the AI parameters and engage in conversations.

2) Backend AI Model Processing:

   The backend leverages the Replicate platform to access the pre-eminent Llama2 models, which are instrumental in processing user inputs and generating coherent, contextually relevant responses.

Deployment and Operational Workflow:

To operationalize Abhi's Llama Chatbot within a local environment, the following procedural steps are delineated:

1) Preparatory Configuration:
   - Ensuring the availability of Python 3.x and the establishment of an optional virtual environment for project isolation.
2) Dependency Acquisition:
   - Streamlit and Replicate libraries are integrated through pip to furnish the necessary operational capabilities.

3) Secure Credential Management:
   - Utilization of environment variables for the encapsulation of the `REPLICATE_API_TOKEN`, thereby fortifying the security posture.

4) Application Invocation:
   - The Streamlit framework is leveraged to launch the application, rendering the user interface for interaction.

Operational Script:

```
# Script for environmental setup and application execution

# Virtual environment establishment for isolation
python -m venv chatbot_env
source chatbot_env/bin/activate

# Integration of requisite libraries
pip install streamlit replicate

# Secure management of Replicate API token
export REPLICATE_API_TOKEN='<Your_Replicate_API_Token>'
```

```
# Invocation of the Streamlit-based application
streamlit run app.py
```

Feature Synopsis

- Model Stratification: Provision for selecting among Llama2-7B, Llama2-13B, and Llama2-70B models, enabling a balance between computational efficiency and response fidelity.

- Parametric Customization: Implementation of temperature, top_p, and max_length sliders within the Streamlit sidebar, facilitating granular control over the chatbot's response dynamics.

- Interactive Dialogue Interface:The main viewport is dedicated to the dialogue interface, presenting a chronological view of the conversation and enabling real-time interaction with the AI entity.

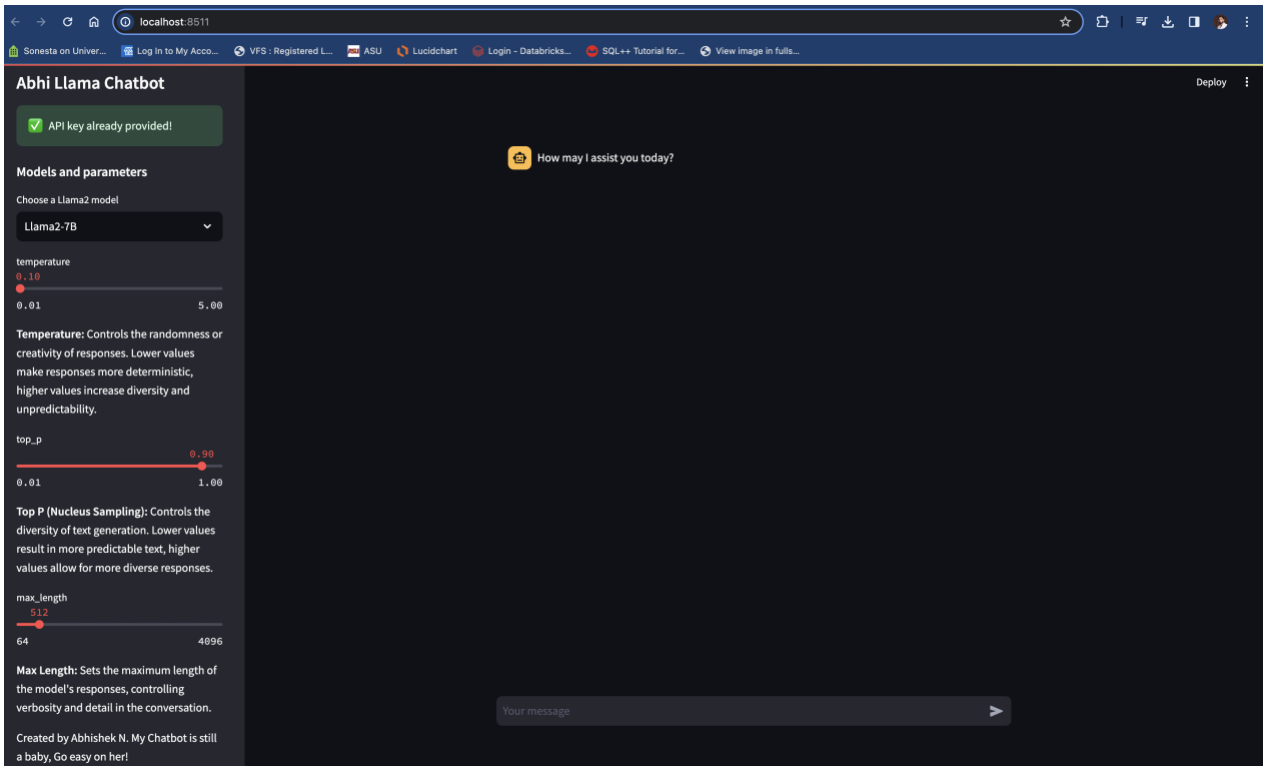
- Credential Securitization: The strategic use of environment variables for API key management precludes the risk of credential exposure, thereby upholding stringent security standards.

- Adaptability for Cloud Integration: The foundational design principles of the project advocate for adaptability, ensuring that transitioning to a cloud-based deployment model can be accomplished with minimal friction.

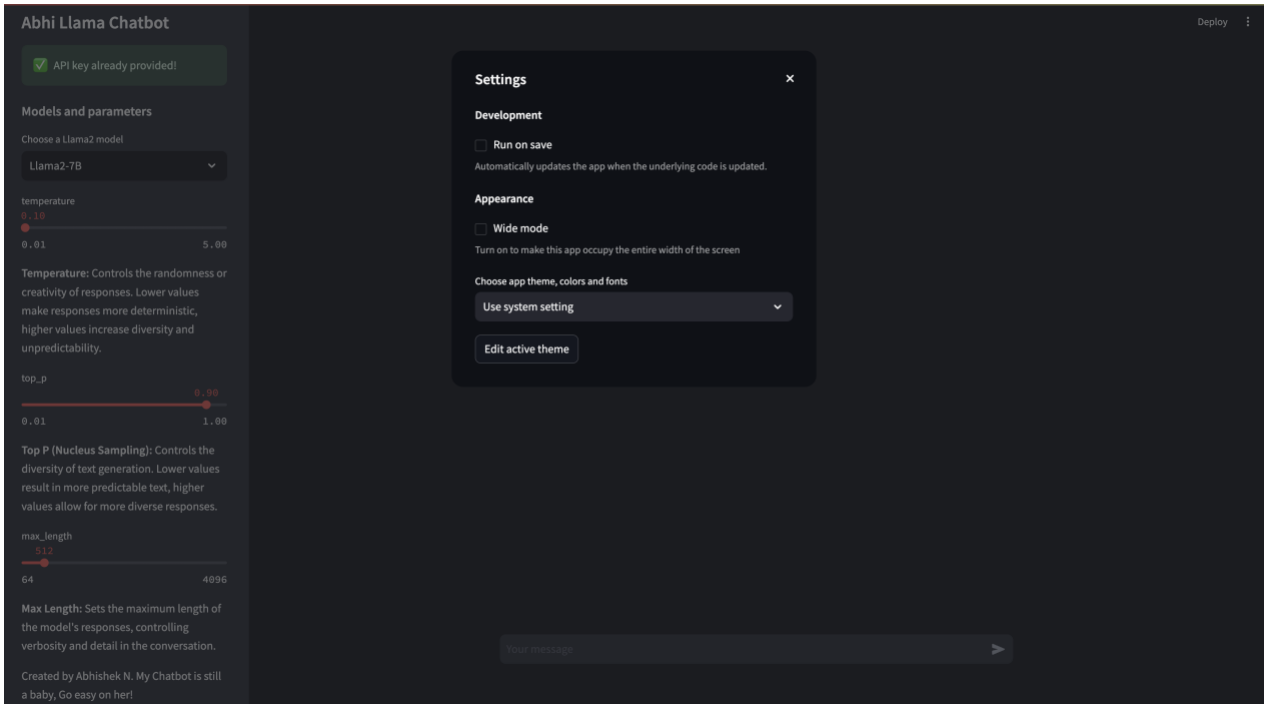
Interface and User Interaction:

The user interface is engineered for maximal usability and minimal cognitive load, ensuring that users can focus on interaction rather than navigation.

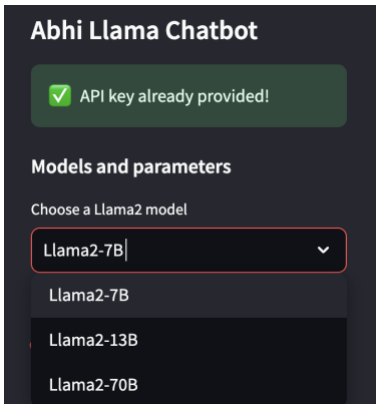
Dialogue Interface Visual:



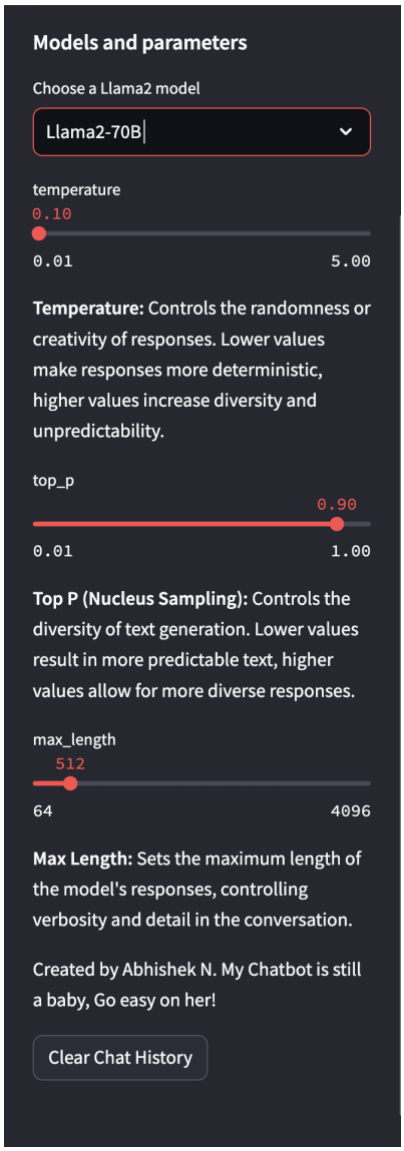
Configuration Sidebar Visual:



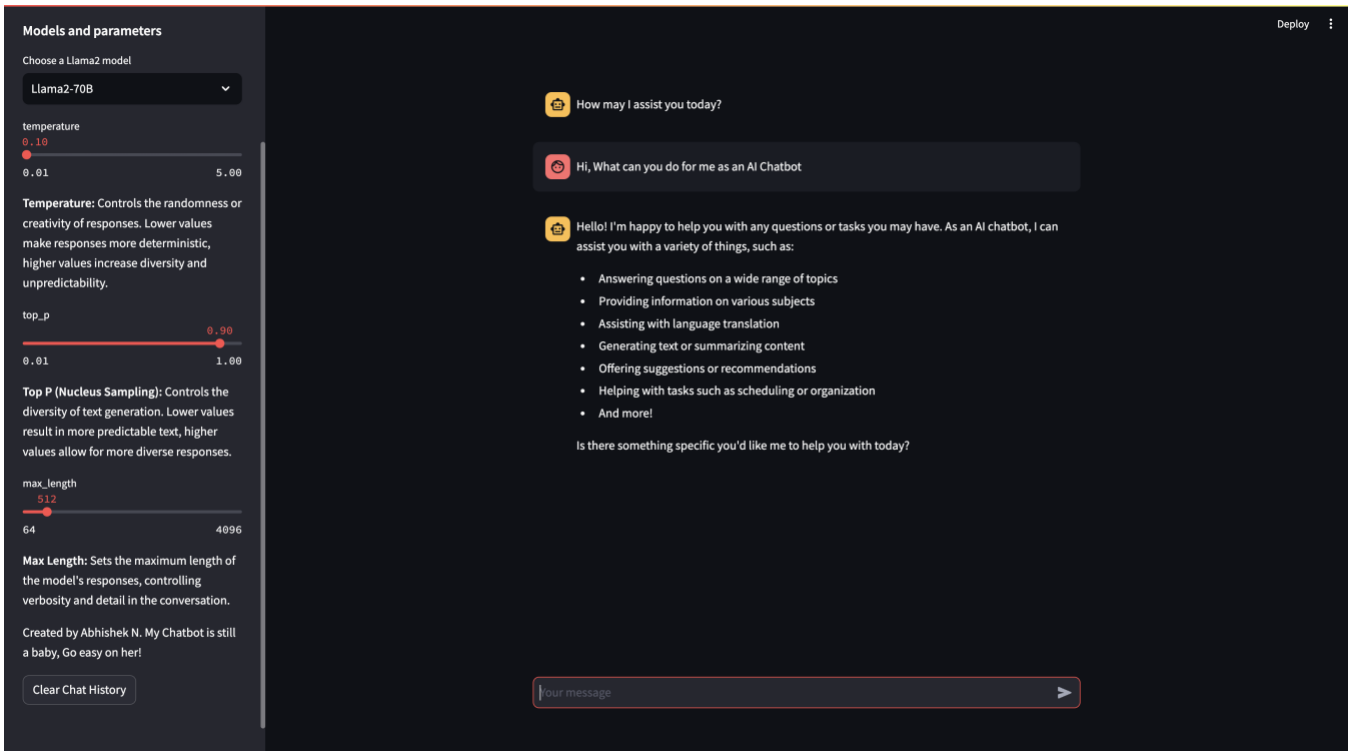
Model Selection among Llama2 – 7B, 13B, 70B



Hyper Parameters:



Output Screenshot:



Code:

```
import streamlit as st
import replicate
import os

# App title
st.set_page_config(page_title="Abhi's Llama Chatbot")

# Replicate Credentials
with st.sidebar:
    st.title('Abhi Llama Chatbot')
    # Use os.getenv to get the environment variable directly
    replicate_api = os.getenv('REPLICATE_API_TOKEN')
    if replicate_api:
        st.success('API key already provided!', icon='✅')
    else:
        # If the API token is not found in the environment, allow the user to input it
        replicate_api = st.text_input('Enter Replicate API token:', type='password')
        if not replicate_api:
            st.warning('Please enter your credentials!', icon='⚠')
        else:
            os.environ['REPLICATE_API_TOKEN'] = replicate_api  # Set the token as an environment variable for the current session
            st.success('Proceed to entering your prompt message!', icon='👉')
```

```
    # Refactored from https://github.com/a16z-infra/llama2-chatbot
    st.subheader('Models and parameters')
    selected_model = st.sidebar.selectbox('Choose a Llama2 model', ['Llama2-7B', 'Llama2-13B',
'Llama2-70B'], key='selected_model')
    if selected_model == 'Llama2-7B':
        llm = 'a16z-infra/llama7b-v2-
chat:4f0a4744c7295c024a1de15e1a63c880d3da035fa1f49bfd344fe076074c8eea'
    elif selected_model == 'Llama2-13B':
        llm = 'a16z-infra/llama13b-v2-
chat:df7690f1994d94e96ad9d568eac121aecf50684a0b0963b25a41cc40061269e5'
    else:
        llm = 'replicate/llama70b-v2-
chat:e951f18578850b652510200860fc4ea62b3b16fac280f83ff32282f87bbd2e48'

    temperature = st.sidebar.slider('temperature', min_value=0.01, max_value=5.0, value=0.1, step=0.01)
    st.markdown('**Temperature:** Controls the randomness or creativity of responses. Lower values
make responses more deterministic, higher values increase diversity and unpredictability.')

    top_p = st.sidebar.slider('top_p', min_value=0.01, max_value=1.0, value=0.9, step=0.01)
    st.markdown('**Top P (Nucleus Sampling):** Controls the diversity of text generation. Lower values
result in more predictable text, higher values allow for more diverse responses.')

    max_length = st.sidebar.slider('max_length', min_value=64, max_value=4096, value=512, step=8)
    st.markdown('**Max Length:** Sets the maximum length of the model\'s responses, controlling
verbosity and detail in the conversation.')

    st.markdown('Created by Abhishek N. My Chatbot is still a baby, Go easy on her!')

os.environ['REPLICATE_API_TOKEN'] = replicate_api

# Store LLM generated responses
if "messages" not in st.session_state.keys():
    st.session_state.messages = [{"role": "assistant", "content": "How may I assist you today?"}]

# Display or clear chat messages
for message in st.session_state.messages:
    with st.chat_message(message["role"]):
        st.write(message["content"])

def clear_chat_history():
    st.session_state.messages = [{"role": "assistant", "content": "How may I assist you today?"}]
st.sidebar.button('Clear Chat History', on_click=clear_chat_history)

# Function for generating LLaMA2 response
def generate_llama2_response(prompt_input):
```

```
    string_dialogue = "You are a helpful assistant. You do not respond as 'User' or pretend to be 'User'.
You only respond once as 'Assistant'."
    for dict_message in st.session_state.messages:
        if dict_message["role"] == "user":
            string_dialogue += "User: " + dict_message["content"] + "\n\n"
        else:
            string_dialogue += "Assistant: " + dict_message["content"] + "\n\n"
    output = replicate.run(llm,
                           input={"prompt": f"{string_dialogue} {prompt_input} Assistant: ",
                                  "temperature":temperature, "top_p":top_p, "max_length":max_length,
"repetition_penalty":1})
    return output

# User-provided prompt
if prompt := st.chat_input(disabled=not replicate_api):
    st.session_state.messages.append({"role": "user", "content": prompt})
    with st.chat_message("user"):
        st.write(prompt)

# Generate a new response if last message is not from assistant
if st.session_state.messages[-1]["role"] != "assistant":
    with st.chat_message("assistant"):
        with st.spinner("Thinking..."):
            response = generate_llama2_response(prompt)
            placeholder = st.empty()
            full_response = ''
            for item in response:
                full_response += item
                placeholder.markdown(full_response)
            placeholder.markdown(full_response)
    message = {"role": "assistant", "content": full_response}
    st.session_state.messages.append(message)
```

Encountered Challenges:

Rate Limit Constraints: Initial encounters with API rate limitations were systematically addressed through the strategic implementation of request caching and optimization of API interactions.

Model Tuning Complexities: The calibration of model parameters to achieve an optimal equilibrium between response quality and resource consumption necessitated an iterative approach, involving extensive experimentation and fine-tuning.

Evolutionary Roadmap:

Cloud Platform Transition: Future iterations will explore the migration of the application to a cloud platform to enhance its scalability and accessibility.

Authentication Mechanisms: The introduction of authentication protocols will pave the way for personalized user experiences and the preservation of conversational histories.

Advanced Linguistic Features: The integration of sophisticated NLP functionalities, such as sentiment analysis, is anticipated to further enrich the conversational capabilities of the chatbot, making interactions more nuanced and context-aware.