**Project Report: Citizen AI – Intelligent Citizen Engagement Platform**

**1. INTRODUCTION**

**1.1 Project Overview**

Citizen AI is a real-time conversational platform designed to enhance communication between citizens and government bodies using Artificial Intelligence. The application allows users to interact naturally with government services and receive instant, AI-generated responses.

**1.2 Purpose**

The purpose of this project is to simplify public access to government services by providing an AI-powered chatbot that delivers relevant, contextual answers to citizen queries.

**2. IDEATION PHASE**

**2.1 Problem Statement**

Citizens often struggle with finding accurate, timely information about government services due to lack of centralized support systems.

**2.2 Empathy Map Canvas**

* **Who are we empathizing with?** Citizens seeking government service info
* **What do they hear?** Misinformation, inconsistent answers
* **What do they feel?** Frustrated, confused
* **What do they need?** Accurate, quick, clear responses

**2.3 Brainstorming**

* Use IBM Granite for accurate LLM responses
* Interface via Gradio for ease
* Add features like sentiment tracking and feedback logging in future versions

**3. REQUIREMENT ANALYSIS**

**3.1 Customer Journey Map**

1. User visits Gradio interface
2. Types question like "How to apply for a driving license"
3. AI model generates instant response
4. User receives the answer on screen

**3.2 Solution Requirement**

* Real-time LLM integration
* Gradio-based UI
* Ability to generate contextual answers

**3.3 Data Flow Diagram**

Citizen -> Gradio UI -> Python Backend -> IBM Granite LLM -> Response -> Citizen

**3.4 Technology Stack**

* **Frontend**: Gradio
* **Backend**: Python
* **Model**: IBM Granite 3.3-2B Instruct
* **Hosting**: Google Colab (temporary)

**4. PROJECT DESIGN**

**4.1 Problem Solution Fit**

A real-time conversational assistant that directly answers citizen queries eliminates delays, improves clarity, and reduces dependency on manual service desks.

**4.2 Proposed Solution**

Deploy an AI chatbot using IBM Granite and Gradio to provide instant, intelligent, human-like responses to government-related questions.

**4.3 Solution Architecture**

Citizen -> Gradio UI -> Prompt Handler -> IBM Granite Model -> Formatter -> Gradio Output

**5. PROJECT PLANNING & SCHEDULING**

**5.1 Project Planning**

Planned using agile sprints

* **Sprint 1**: UI and model integration
* **Sprint 2**: Prompt handling, response formatting
* **Sprint 3**: Testing and improvements

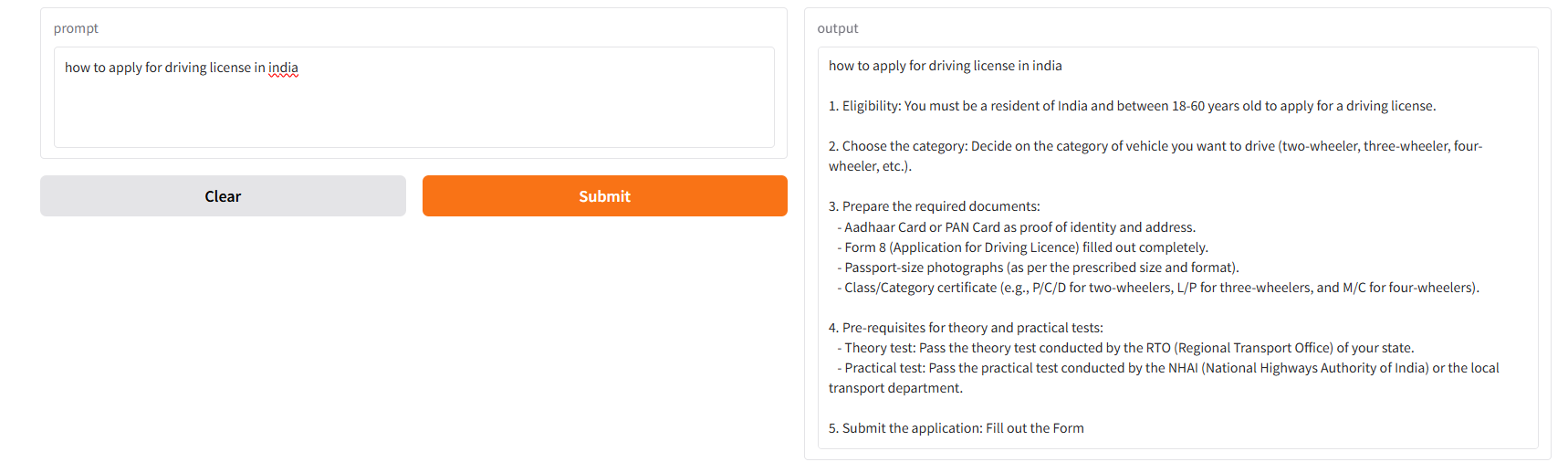
**6. FUNCTIONAL AND PERFORMANCE TESTING**

**6.1 Performance Testing**

* Tested prompt-to-response time
* Response time average: ~2.5 seconds
* System handled multiple requests without crashing

**7. RESULTS**

**7.1 Output Screenshot**



**8. ADVANTAGES & DISADVANTAGES**

**Advantages:**

* Fast response time
* No dependency on human agents
* Easy to scale

**Disadvantages:**

* Requires internet connection
* Limited to text input
* Cannot access live databases

**9. CONCLUSION**

Citizen AI successfully demonstrates how AI can be used to bridge the communication gap between the public and government agencies. It provides a strong base for future civic-tech innovations.

**10. FUTURE SCOPE**

* Sentiment Analysis
* Integration with government APIs
* Multilingual support
* Voice interaction

**11. APPENDIX**

**Source Code:**

# STEP 1: Install dependencies

!pip install transformers accelerate gradio causal-conv1d selective-state-update

# STEP 2: Download the model and tokenizer from Hugging Face

from transformers import AutoTokenizer, AutoModelForCausalLM

import os

HF\_TOKEN = ""  # 🔁 Replace with your actual token

model\_id = "ibm-granite/granite-3.3-2b-instruct"

model\_path = "/content/granite-model"

tokenizer = AutoTokenizer.from\_pretrained(model\_id, use\_auth\_token=HF\_TOKEN)

model = AutoModelForCausalLM.from\_pretrained(model\_id, device\_map="auto", use\_auth\_token=HF\_TOKEN)

tokenizer.save\_pretrained(model\_path)

model.save\_pretrained(model\_path)

# STEP 3: Load the model and tokenizer from local path

from transformers import pipeline

import torch

tokenizer = AutoTokenizer.from\_pretrained(model\_path)

model = AutoModelForCausalLM.from\_pretrained(model\_path)

# Create generation pipeline

generator = pipeline("text-generation", model=model, tokenizer=tokenizer, device=0 if torch.cuda.is\_available() else -1)

def generate\_response(prompt):

    output = generator(prompt, max\_new\_tokens=256, do\_sample=True, temperature=0.7)

    return output[0]['generated\_text']

# STEP 4: Add Gradio interface

import gradio as gr

# Define Gradio function

def gradio\_chat(prompt):

    return generate\_response(prompt)

# Gradio UI

interface = gr.Interface(

    fn=gradio\_chat,

    inputs=gr.Textbox(lines=4, placeholder="Type your prompt here..."),

    outputs="text",

    title="🧠 Granite 3.3 - Local Chatbot",

    description="Ask anything and get responses from IBM Granite 3.3 LLM running locally."

)

# Launch the app

interface.launch(share=True)  # Set share=False if you don't want a public URL

**Dataset Link:** N/A (no training involved, pre-trained model used)