Citizen AI: Intelligent Citizen Engagement Platform Using IBM Granite

Project Documentation

Introduction

Project title: Citizen AI: Intelligent Citizen Engagement Platform Using IBM Granite

Team members:

Team member : Thiru murugan K
Team member : NIRANJAN A
Team member : THARUN A
Team member : SIVAKUMAR E

Project Overview

Purpose:

The purpose of Citizen AI is o provide users with an intelligent, AI-powered assistant capable of answering queries related to government services, civic issues, and public engagement. By leveraging IBM's Granite LLM and real-time AI processing, the platform aims to deliver accurate, accessible, and user-friendly guidance while supporting authorities in improving transparency and citizen satisfaction.

Features:

1.Conversational Interface:

Key Point: Natural language interaction
Functionality: Users can ask questions about government services, policies, or civic issues and receive AI-generated responses.

2.Service Guidance:

Key Point: Citizen-focused information
Functionality: Provides step-by-step guidance for services like Aadhaar application, Voter ID registration, RTI queries, and transportation-related information.

3.Quick Question Buttons:

Key Point: Easy access to common queries
Functionality: Pre-set buttons for frequently asked topics such as Aadhaar, RTI Act, Voter ID, and Railway ticket booking.

4.Chat Management

Key Point: Enhanced user control
Functionality: Options to send queries, clear chat history, and download conversations as a text file.

5.Gradio UI

Key Point: User-friendly interface
Functionality: Gradient-based chatbot design with text input, quick action buttons, and real-time responses.

Architecture:

Frontend (Gradio):

Chat-based web interface with input textbox and functional buttons (Send, Clear, Download, Quick Questions).

Inputs: User queries in natural language.

Outputs: AI-generated responses displayed in the chat, with option to download as text.

Backend (Python + Transformers):

1.Processes user queries and generates AI responses using IBM Granite LLM.

2.Handles model loading and GPU optimization if available.

LLM Integration (IBM Granite – Hugging Face Model):

Model: ibm-granite/granite-1b-code-instruct (lightweight and efficient for Colab).

1.Performs natural language understanding and response generation. Deployment (Google Colab):

2.Runs in Google Colab with T4 GPU for smooth AI processing.

3. Application can be shared publicly using launch(share=True).

Setup Instructions Prerequisites:

1.Python 3.9 or later

2.pip for package installation

3.Internet connection (for downloading model)

4.GPU recommended for faster response (T4 GPU in Colab)

Installation Process:

1. Clone the repository.

2. Install dependencies: pip install transformers torch gradio

3. Run the Gradio app: python citizen_ai.py

4. Open the provided local URL or use the public share link

Folder Structure:

citizen_ai.py	– Main Gradio app and UI layout
requirements.txt	– Python dependencies

Running the Application:

1. Launch the Python script citizen_ai.py or run the notebook in Colab.

2. Gradio interface opens in the browser.

3. User flow: - Enter a query   Click Send   View AI response.

4. Use Quick Question Buttons for instant responses.

5. Use Clear Chat to reset conversation.

API Documentation:

(Note: Current version runs as a Gradio interface; no REST API implemented. Optional future enhancement could add FastAPI backend.)

Inputs: User query (Textbox or Quick Question button)

Outputs: AI response (Text), Chat history (Downloadable file)

Authentication:

1.Current version runs in an open environment.

2.No authentication implemented.

3.Future enhancement: Add login system and role-based access.

User Interface:

1.Minimalist chatbot interface with Gradio.

2.Includes: Text input field Action buttons (Send, Clear)

3.Quick Question buttons (RTI, Aadhaar, Voter ID, Railway)

4.Gradient-based background for improved visual appeal.

Testing:

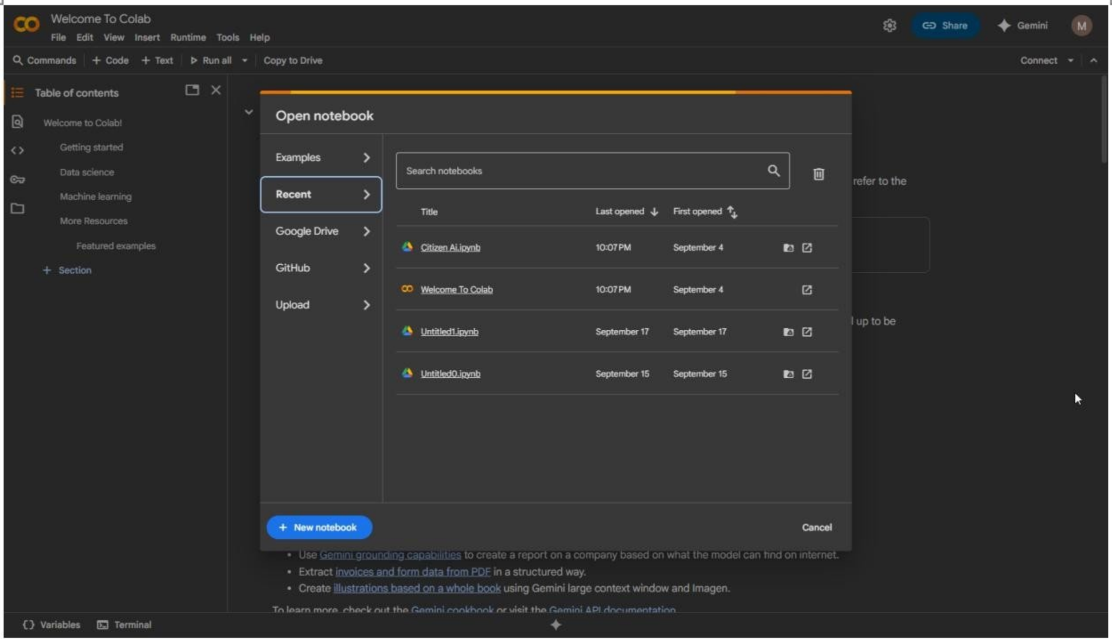
Unit Testing:
Validate input handling and model response.

Manual Testing: Ask queries on government services and verify responses.

Edge Case Handling: Empty input, repeated queries, long text.

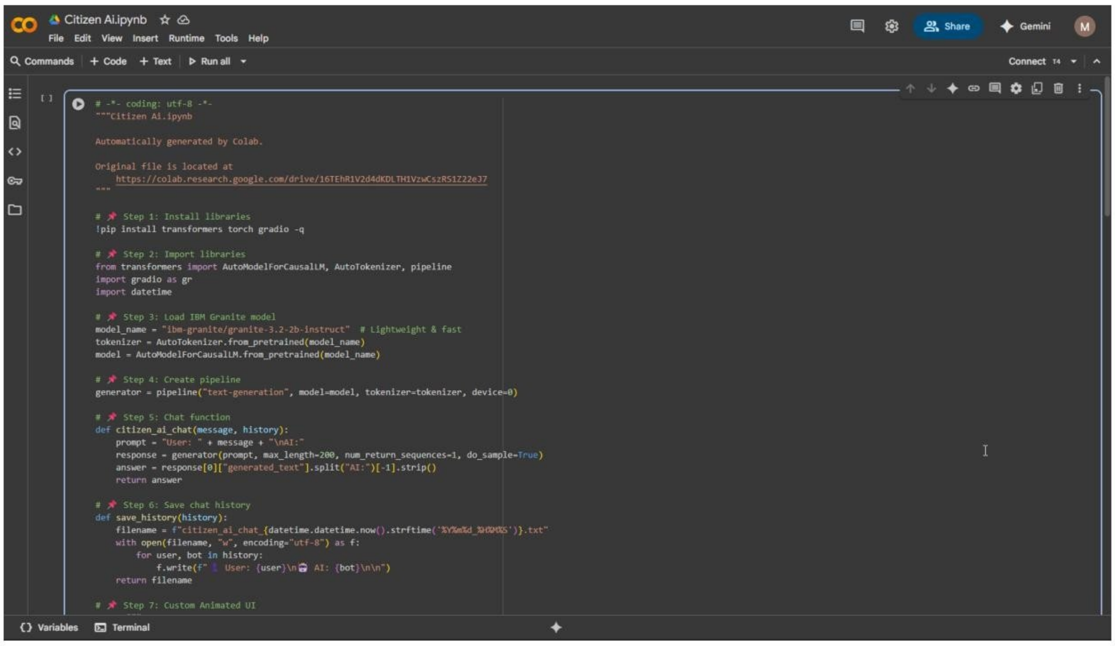
Screenshots:

TOOLS:

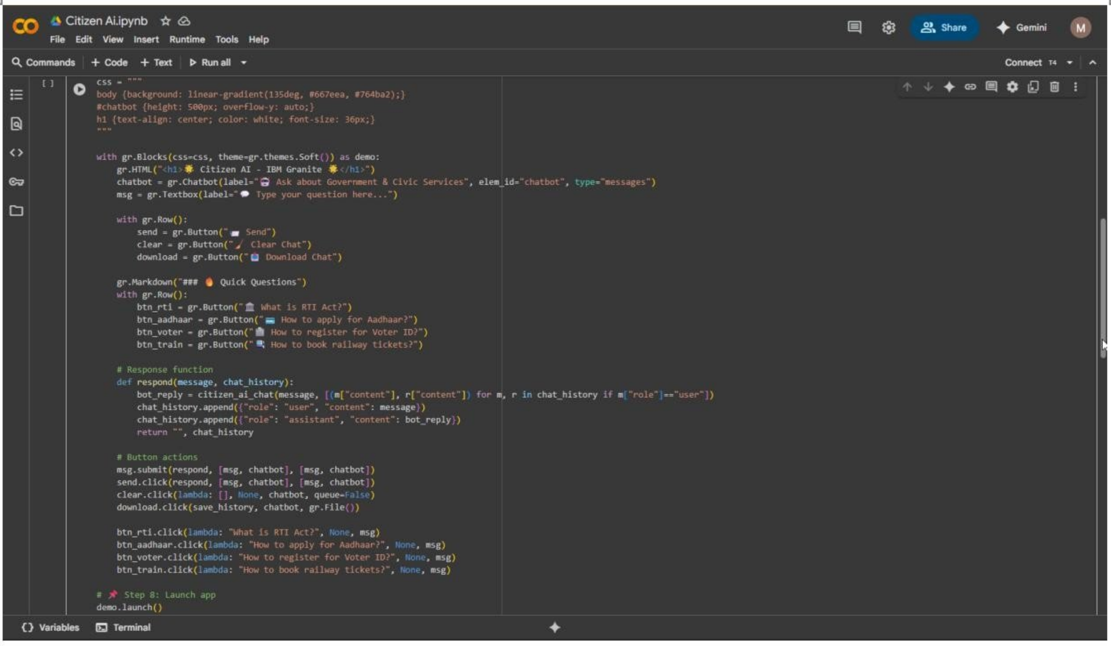


Title	Last opened	First opened
Citizen Ai.ipynb	10:07 PM	September 4
Welcome To Colab	10:07 PM	September 4
Untitled1.ipynb	September 17	September 17
Untitled0.ipynb	September 15	September 15

CODING:


Citizen Ai.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Connect
Variables Terminal

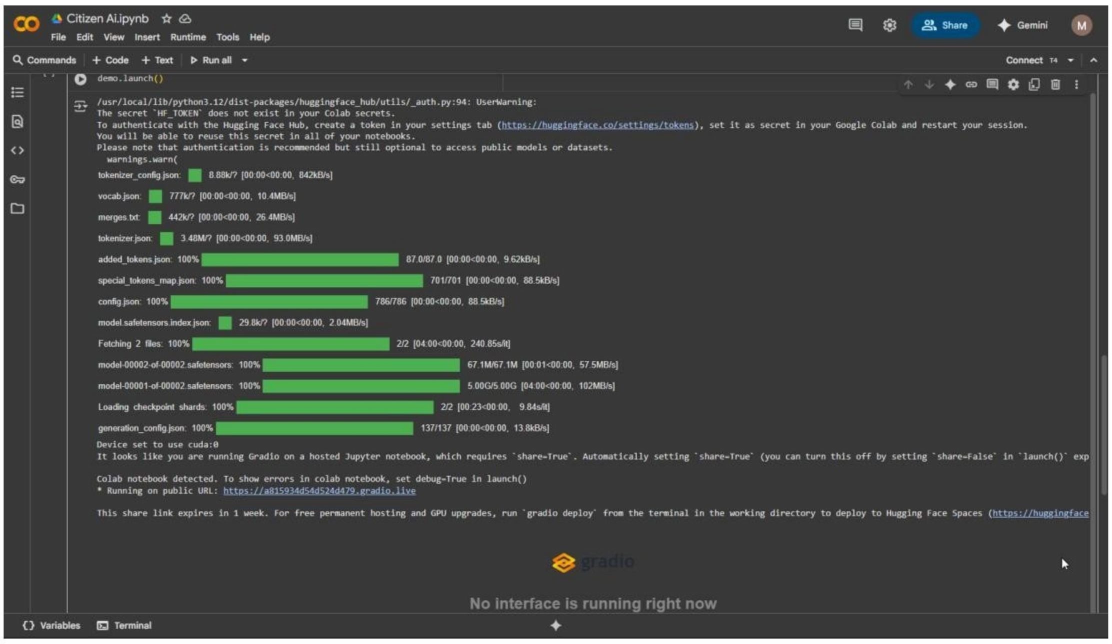
```python
# -*- coding: utf-8 -*-
"""Citizen Ai.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/16TEhR1V2d4dKDLTH1VzwCszRS1Z22eJ7
"""

# 📌 Step 1: Install libraries
!pip install transformers torch gradio -q

# 📌 Step 2: Import libraries
from transformers import AutoModelForCausalLM, AutoTokenizer, pipeline
import gradio as gr
import datetime

# 📌 Step 3: Load IBM Granite model
model_name = "ibm-granite/granite-3.2-2b-instruct"  # Lightweight & fast
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(model_name)

# 📌 Step 4: Create pipeline
generator = pipeline("text-generation", model=model, tokenizer=tokenizer, device=0)

# 📌 Step 5: Chat function
def citizen_ai_chat(message, history):
    prompt = "User: " + message + "\nAI:"
    response = generator(prompt, max_length=200, num_return_sequences=1, do_sample=True)
    answer = response[0]["generated_text"].split("AI:")[-1].strip()
    return answer

# 📌 Step 6: Save chat history
def save_history(history):
    filename = f"citizen_ai_chat_{datetime.datetime.now().strftime('%Y%m%d_%H%M%S')}.txt"
    with open(filename, "w", encoding="utf-8") as f:
        for user, bot in history:
            f.write(f"👤 User: {user}\n🤖 AI: {bot}\n\n")
    return filename

# 📌 Step 7: Custom Animated UI
```



Citizen Ai.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text ▷ Run all
Share
Gemini
Connect T4
Variables
Terminal

```
css = """
body {background: linear-gradient(135deg, #667eea, #764ba2);}
#chatbot {height: 500px; overflow-y: auto;}
h1 {text-align: center; color: white; font-size: 36px;}
"""

with gr.Blocks(css=css, theme=gr.themes.Soft()) as demo:
    gr.HTML("<h1>🌟 Citizen AI - IBM Granite 🌟</h1>")
    chatbot = gr.Chatbot(label="🤖 Ask about Government & Civic Services", elem_id="chatbot", type="messages")
    msg = gr.Textbox(label="💬 Type your question here...")

    with gr.Row():
        send = gr.Button("📨 Send")
        clear = gr.Button("🧹 Clear Chat")
        download = gr.Button("📥 Download Chat")

    gr.Markdown("### 🔥 Quick Questions")
    with gr.Row():
        btn_rti = gr.Button("🏛 What is RTI Act?")
        btn_aadhaar = gr.Button("💳 How to apply for Aadhaar?")
        btn_voter = gr.Button("🗳 How to register for Voter ID?")
        btn_train = gr.Button("🚆 How to book railway tickets?")

    # Response function
    def respond(message, chat_history):
        bot_reply = citizen_ai_chat(message, [(m["content"], r["content"]) for m, r in chat_history if m["role"]=="user"])
        chat_history.append({"role": "user", "content": message})
        chat_history.append({"role": "assistant", "content": bot_reply})
        return "", chat_history

    # Button actions
    msg.submit(respond, [msg, chatbot], [msg, chatbot])
    send.click(respond, [msg, chatbot], [msg, chatbot])
    clear.click(lambda: [], None, chatbot, queue=False)
    download.click(save_history, chatbot, gr.File())

    btn_rti.click(lambda: "What is RTI Act?", None, msg)
    btn_aadhaar.click(lambda: "How to apply for Aadhaar?", None, msg)
    btn_voter.click(lambda: "How to register for Voter ID?", None, msg)
    btn_train.click(lambda: "How to book railway tickets?", None, msg)

# 📌 Step 8: Launch app
demo.launch()
```

OUTPUT LINK:

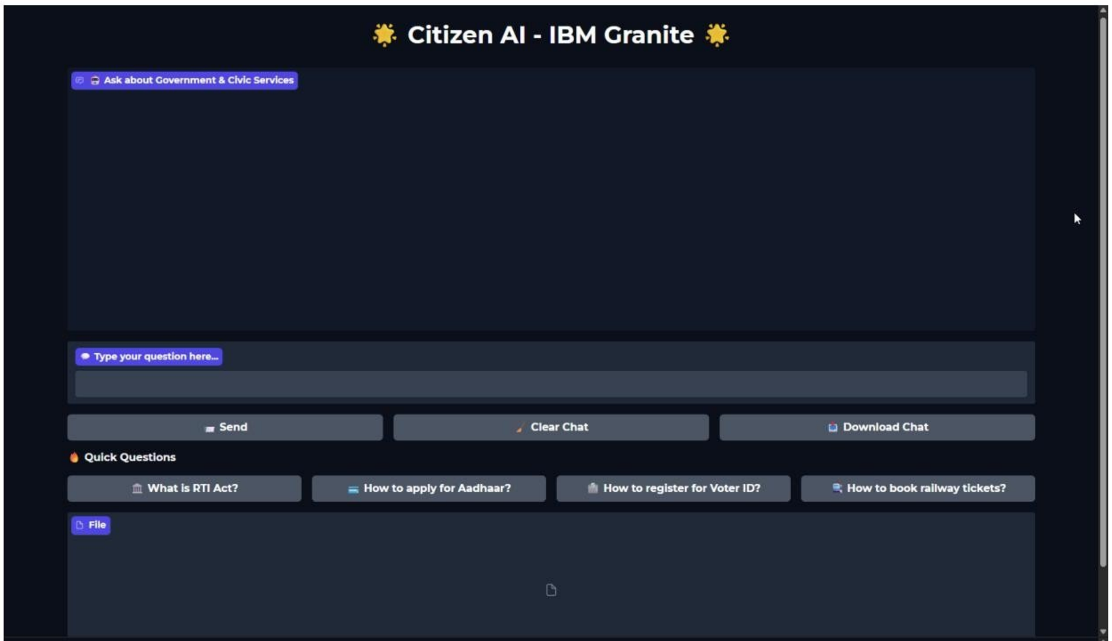


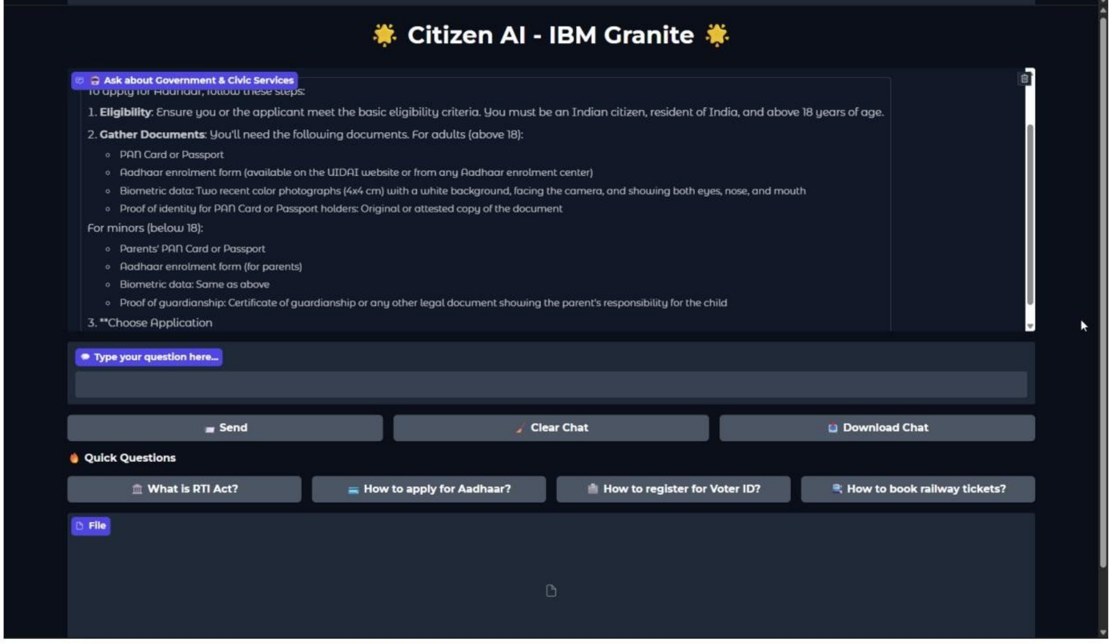
Citizen Ai.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text ▷ Run all
Share
Gemini
Connect T4
gradio
No interface is running right now
Variables
Terminal

```
demo.launch()
```

```
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secret in your Google Colab and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
tokenizer_config.json:  8.88k/? [00:00<00:00, 842kB/s]
vocab.json:  777k/? [00:00<00:00, 10.4MB/s]
merges.txt:  442k/? [00:00<00:00, 26.4MB/s]
tokenizer.json:  3.48M/? [00:00<00:00, 93.0MB/s]
added_tokens.json: 100%  87.0/87.0 [00:00<00:00, 9.62kB/s]
special_tokens_map.json: 100%  701/701 [00:00<00:00, 88.5kB/s]
config.json: 100%  786/786 [00:00<00:00, 88.5kB/s]
model.safetensors.index.json:  29.8k/? [00:00<00:00, 2.04MB/s]
Fetching 2 files: 100%  2/2 [04:00<00:00, 240.85s/it]
model-00002-of-00002.safetensors: 100%  67.1M/67.1M [00:01<00:00, 57.5MB/s]
model-00001-of-00002.safetensors: 100%  5.00G/5.00G [04:00<00:00, 102MB/s]
Loading checkpoint shards: 100%  2/2 [00:23<00:00, 9.84s/it]
generation_config.json: 100%  137/137 [00:00<00:00, 13.8kB/s]
Device set to use cuda:0
It looks like you are running Gradio on a hosted Jupyter notebook, which requires `share=True`. Automatically setting `share=True` (you can turn this off by setting `share=False` in `launch()` exp

Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://a815934d54d524d479.gradio.live

This share link expires in 1 week. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in the working directory to deploy to Hugging Face Spaces (https://huggingface
```

FINAL OUTPUT:


Citizen AI - IBM Granite
Ask about Government & Civic Services
Type your question here...
Send
Clear Chat
Download Chat
Quick Questions
What is RTI Act?
How to apply for Aadhaar?
How to register for Voter ID?
How to book railway tickets?
File



To apply for Aadhaar, follow these steps:

1. **Eligibility**: Ensure you or the applicant meet the basic eligibility criteria. You must be an Indian citizen, resident of India, and above 18 years of age.
2. **Gather Documents**: You'll need the following documents. For adults (above 18):
   - PAN Card or Passport
   - Aadhaar enrolment form (available on the UIDAI website or from any Aadhaar enrolment center)
   - Biometric data: Two recent color photographs (4x4 cm) with a white background, facing the camera, and showing both eyes, nose, and mouth
   - Proof of identity for PAN Card or Passport holders: Original or attested copy of the document

For minors (below 18):

   - Parents' PAN Card or Passport
   - Aadhaar enrolment form (for parents)
   - Biometric data: Same as above
   - Proof of guardianship: Certificate of guardianship or any other legal document showing the parent's responsibility for the child

3. **Choose Application

Type your question here...
Send
Clear Chat
Download Chat
Quick Questions
What is RTI Act?
How to apply for Aadhaar?
How to register for Voter ID?
How to book railway tickets?
File

Known Issues:

1.Large queries may exceed model’s maximum token limit.

2.Some responses may be generic due to limited domain training.

3.Free Colab session may disconnect, stopping the app.

Future Enhancements:

1.Deploy permanently on Hugging Face Spaces or Streamlit Cloud.

2.Add voice input and voice output (speech-to-text and text-to-speech).

3.Add dashboard for sentiment analysis of citizen feedback. - Enable multilingual support (e.g., Tamil, Telugu, Hindi, etc.).