TARDIS Prototype:
A Serverless Conversational AI Assistant

Pranav Kumar Kaliaperumal

1 Project Motivation

The goal of this project was to design and deploy a lightweight, conversational AI assistant that could be accessed instantly by non-technical users through a web browser. Inspired by the fictional TARDIS from Doctor Who, the assistant emphasizes interaction, memory, and decoding capabilities rather than raw knowledge retrieval. A core constraint was to avoid heavyweight infrastructure while still demonstrating production-style AI integration.

2 High-Level Architecture

The system is divided into a static frontend and a serverless backend. The frontend is hosted on GitHub Pages and contains all user-facing logic, including chat rendering, speech interfaces, and cipher decoding. The backend is implemented using Cloudflare Workers and acts as a secure proxy between the client and Hugging Face's inference services.

This separation ensures API credentials are never exposed to the browser while keeping deployment costs minimal.

3 Frontend Implementation

3.1 welcome.html

The `welcome.html` file serves as a landing screen designed to provide a clean, non-intimidating entry point for users. It uses a navy-blue background and a typewriter animation implemented in vanilla JavaScript to introduce the assistant. The page contains no framework dependencies and links directly to the main assistant interface, ensuring fast load times and maximum compatibility.

3.2 index.html

The `index.html` file defines the structure of the AI assistant interface. It includes controls for model selection, personality sliders, voice interaction toggles, memory management, and cipher tools. Semantic HTML elements are used to ensure accessibility and clarity. The page loads all application logic from `app.js` and styling from `styles.css`.

3.3 styles.css

The stylesheet establishes a light grey, blue, and white theme intended to feel professional and unobtrusive. Components such as cards, buttons, sliders, and chat bubbles are styled consistently. Additional styles support expandable sections for memory controls and cipher tools without overwhelming the main chat interface.

3.4 app.js

The `app.js` file contains the majority of the frontend logic. It manages:

- Rendering and updating the chat log
- Managing a rolling conversation history
- Persisting memory using `localStorage`
- Generating system prompts that steer conversational behavior
- Integrating browser speech recognition and synthesis
- Executing client-side Caesar and ROT13 decoding

Rather than relying on a backend database, conversation memory is simulated by sending recent messages with each request. A summarization mechanism periodically compresses long conversations into a short memory block, allowing longer interactions without exceeding model context limits.

4 Backend Implementation

4.1 Cloudflare Worker

The backend is implemented as a single Cloudflare Worker located in `tardis-prototype-proxy/src/index.js`. Its primary responsibility is to accept sanitized chat requests from the frontend, attach the appropriate authorization headers, and forward them to Hugging Face's inference router using an OpenAI-compatible API format.

4.2 Security Considerations

The Worker ensures:

- Hugging Face tokens are never exposed client-side
- CORS headers are applied correctly
- Input payloads are size-limited and validated

Secrets are managed using Wrangler's secret storage, keeping credentials out of version control.

5 Model Selection

The project uses the `google/gemma-2-2b-it` model, selected for its balance of instruction-following capability and low inference latency. The system prompt and few-shot examples are carefully constructed to encourage conversational responses rather than encyclopedic explanations.

6 Challenges Encountered

Several challenges were addressed during development:

- Migrating from deprecated Hugging Face inference endpoints to the Router API
- Managing browser caching issues during GitHub Pages deployment
- Designing conversational memory without a database
- Balancing model verbosity and personality through prompt engineering

Each issue required iterative debugging and architectural refinement.

7 Conclusion

This project demonstrates how a production-style AI assistant can be built using only static hosting and serverless infrastructure. By combining prompt engineering, client-side tooling, and secure backend proxies, the system achieves a responsive, conversational experience without relying on heavy frameworks or paid platforms.