**CitizenAI – Intelligent Citizen Engagement Platform**

**Powered by IBM Granite**

**Project Description:**

**CitizenAI** is a smart citizen engagement platform powered by **IBM Granite** and built using **Flask**. It allows citizens to:

* Interact with government services through a conversational assistant
* Submit feedback and receive **AI-powered sentiment analysis**
* Report issues and concerns, visible to administrators
* View real-time analytics on a dashboard

The application uses the **Granite-3b-code-instruct** model from Hugging Face to generate context-aware responses and analyze sentiment. Built with Flask for the backend and HTML/CSS for the frontend, the system runs locally or via Colab using GPU/CPU-based inference.

**Scenarios:**

**Scenario 1: Real-Time Conversational AI Assistant**

This is the core functionality of CitizenAI. When a user types a question in the chatbot interface—such as:

“How do I apply for a ration card?”  
“What are the working hours of the local municipal office?”

the query is passed to the AI backend (IBM Granite model), processed, and a human-like response is generated. The Flask server handles this routing seamlessly in real-time, providing instant access to public service information.

**Scenario 2:** **Sentiment Analysis**

Citizens submit feedback like:

“The water supply in my area has improved greatly.”  
The platform classifies this as **Positive**, **Neutral**, or **Negative**, helping the government monitor satisfaction trends.

**Scenario 3:** **Concern Reporting**

Users can log issues like:

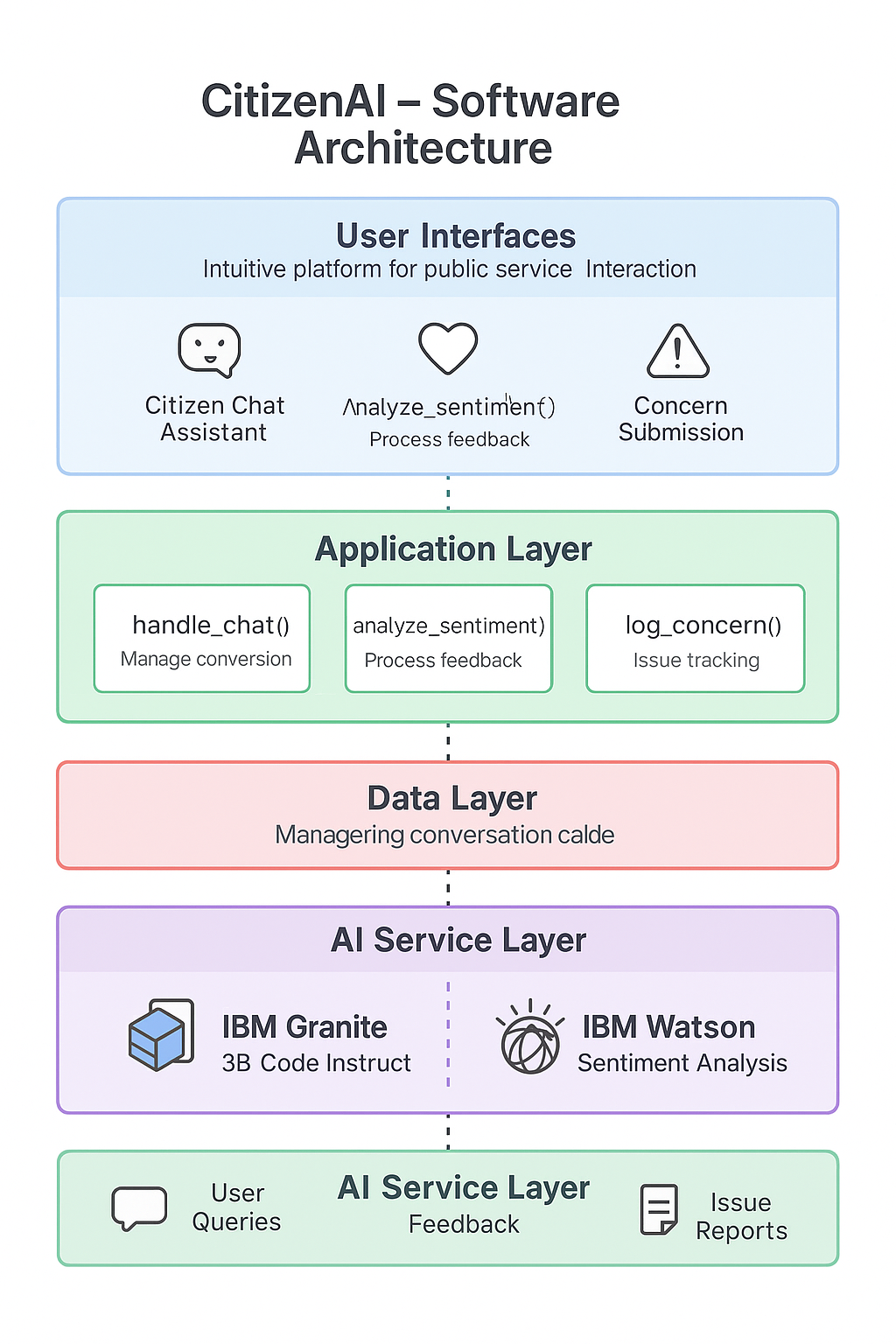
“Garbage collection not happening in street 5.”  
These are recorded and shown in the dashboard to help authorities prioritize.

**Scenario 4: Dashboard Analytics**

Admin dashboard displays:

* Total Positive, Neutral, Negative sentiments
* Recent concerns
* Number of interactions  
  (Charts and metrics rendered in HTML)

**Technical Architecture:**



**Pre-requisites:**

1. Python 3.7+
2. Flask
3. Torch
4. Transformers
5. Accelerate
6. bitsandbytes (optional for quantized models)
7. Browser (Chrome/Edge)

**Activity 1: Model Selection and Architecture**

* **Activity 1.1:** Set up the project architecture using Flask as the backend framework and Hugging Face Transformers for integrating the IBM Granite model.
* **Activity 1.2:** Installed essential libraries including Flask, torch, transformers, accelerate, and bitsandbytes for AI integration and model quantization**.**
* **Activity 1.3:** Defined a layered architecture (User Interface, Flask Application Layer, AI Processing Layer, and Data Management Layer).
* **Activity 2: Core Functionalities Development**
* **Activity 2.1:** Developed core features including:

Real-Time AI Chat Assistant

Sentiment Analysis on citizen feedback

Concern Reporting System

Interactive Analytics Dashboard

* **Activity 2.2:** Implemented sentiment classification logic (Positive, Neutral, Negative) using NLP utilities and linked it with dashboard metrics.
* **Activity 3: App.py Development**
* **Activity 3.1:** Wrote the main application logic inside app.py, handling Flask routes, model integration, and user request processing.
* **Activity 3.2:** Implemented AI helper functions:

granite\_generate\_response() for generating replies

analyze\_sentiment() for text sentiment analysis

Data aggregation logic for dashboard updates

**Activity 4: Frontend Development**

* **Activity 4.1**: Designed HTML templates for all pages (index.html, about.html, services.html, chat.html, dashboard.html, login.html) using a clean and responsive layout.
* **Activity 4.2:** Created dynamic input forms and visual feedback areas using Jinja2 templating in Flask.
* **Activity 4.3:** Styled the interface with styles.css (in static/css) ensuring an intuitive and accessible user experience.

**Activity 5: Deployment**

* **Activity 5.1:** The application was tested locally using http://127.0.0.1:5000 to ensure all features—chat, sentiment analysis, concern reporting, and dashboard—worked correctly. The IBM Granite model was optimized to run on CPU using lightweight settings. All Flask routes were verified, and the project was prepared for future deployment on platforms like Render or Railway.

# Model Selection and Architecture

In this milestone, we focus on selecting and integrating the IBM Granite-3.3-2b-instruct model for our language learning needs. This involves configuring the model with appropriate parameters, ensuring optimal performance for educational content generation, and establishing the foundation for multilingual language instruction capabilities.

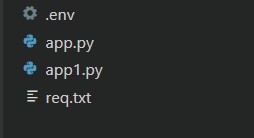
**Activity 1.1: Set up the development environment**

1. Install Python and Pip: Ensure Python is installed along with pip for managing dependencies.
2. Create a Virtual Environment: Set up a virtual environment to isolate project dependencies.
3. Install Required Libraries:

bash

pip install gradio transformers torch matplotlib langid accelerate

1. Set Up Application Structure: Create the initial directory structure for the Language Guru application.



**Core Functionalities Development**

**Activity 2.1: Develop core functionalities**

1. **Real-Time Corrections System:** 
   1. Implement text analysis interface for grammar, spelling, and punctuation

○ Create prompting system for the IBM Granite model to provide detailed corrections

○ Develop word count validation and feedback mechanisms

1. **Explanatory Notes System:** 
   1. Create automatic language detection using langid library

○ Develop contextual language rule breakdown functionality ○ Structure output format to show detailed linguistic analysis

1. **Adaptive Quiz Generator:** 
   1. Build dynamic quiz generation for multiple languages

○ Create three quiz categories: Grammar, Tenses, and Parts of Speech

○ Implement structured question formatting with answer keys

1. **Multilingual Learning Exercises:** 
   1. Implement interactive exercise generation across supported languages

○ Create three exercise types: Grammar Exercises, Sentence Formation, and Tense Exercises

○ Develop comprehensive practice activities with explanations

**Activity 2.2: Implement data management utilities**

1. **Language Detection Integration:** 
   1. Integrate langid library for automatic language identification

○ Support for English, Spanish, Chinese, French, German, and Hindi

○ Implement confidence scoring for detection accuracy

1. **Text Analysis Metrics:** 
   1. Calculate word count, average word length, and sentence complexity

○ Generate language competency visualizations using matplotlib

○ Create radar charts for comprehensive skill assessment

1. **Model Response Processing:** 
   1. Implement intelligent response parsing for different content types

○ Structure outputs for educational clarity and organization

○ Handle model-generated content formatting and presentation

**App.py Development**

**Activity 3.1: Write the main application logic** The app.py file is organized into several key sections:

1. **Imports and Setup:** 
   1. Import necessary libraries (Gradio, Transformers, Torch, Matplotlib, LangID)

○ Load IBM Granite-3.3-2b-instruct model with optimal configuration

○ Initialize tokenizer and model with appropriate device mapping

1. **Core Functions:** 
   1. generate\_response(): Handle AI text generation with the Granite model

○ detect\_language(): Automatic language identification functionality

○ real\_time\_correction(): Process text for comprehensive error analysis

○ explanatory\_notes(): Generate detailed language rule explanations ○ generate\_quiz(): Create adaptive quiz questions for selected languages

○ multilingual\_learning(): Generate structured learning exercises

○ analyze\_text(): Comprehensive text analysis with visualizations

1. **UI Components:** 
   1. Tabbed interface using Gradio Blocks for organized navigation

○ Custom input validation and error handling

○ Interactive visualizations integrated within the interface

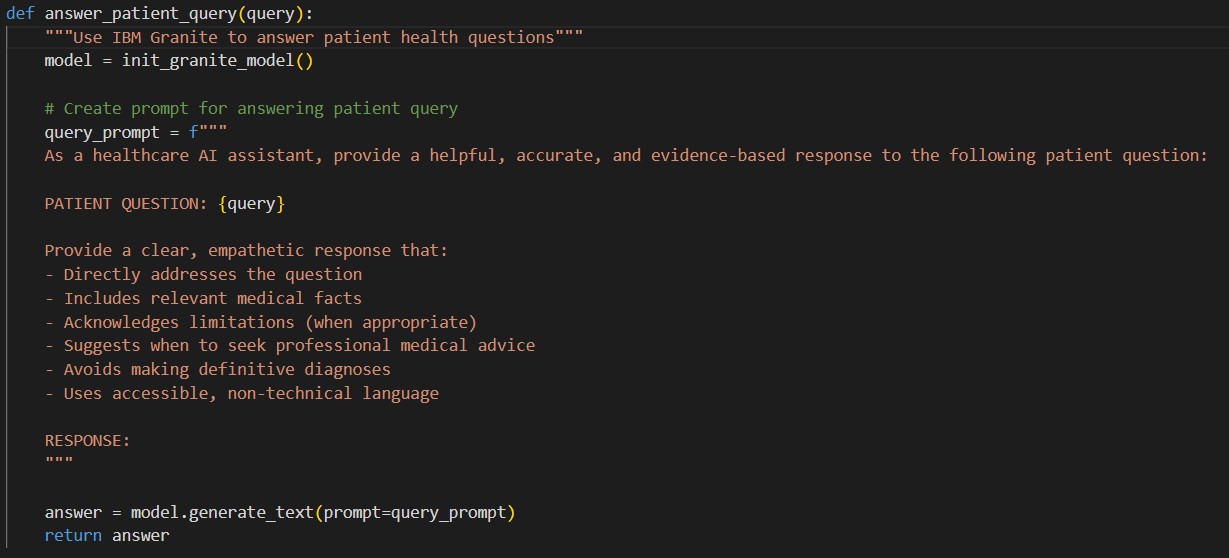
1. **Feature Implementation:** 
   1. display\_patient\_chat(): Chatbot interface for health questions

○ display\_disease\_prediction(): Symptom analysis system

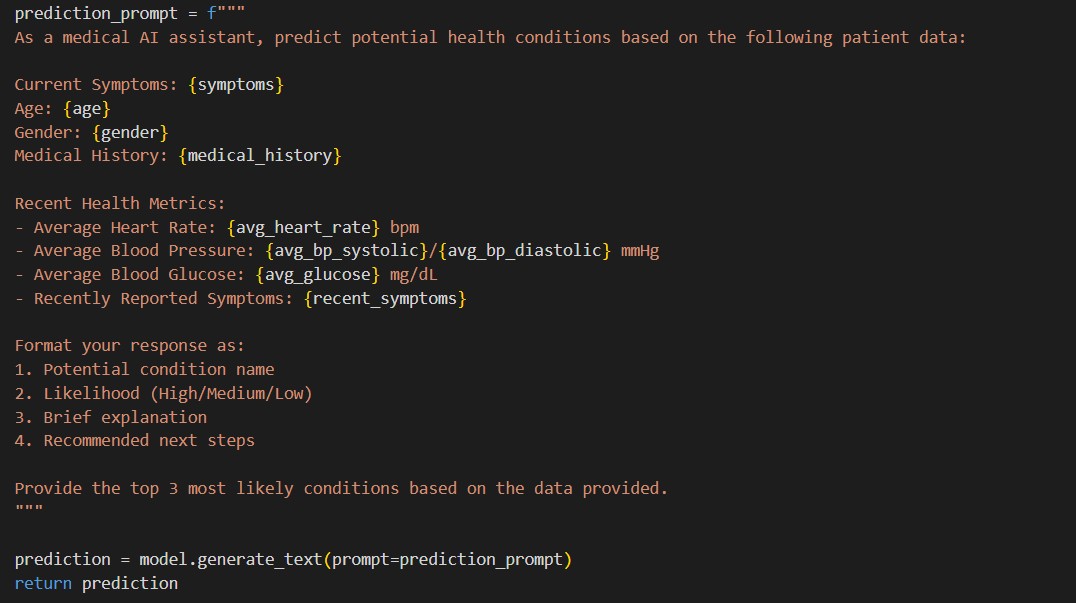
○ display\_treatment\_plans(): Treatment plan generator

○ display\_health\_analytics(): Interactive health dashboard

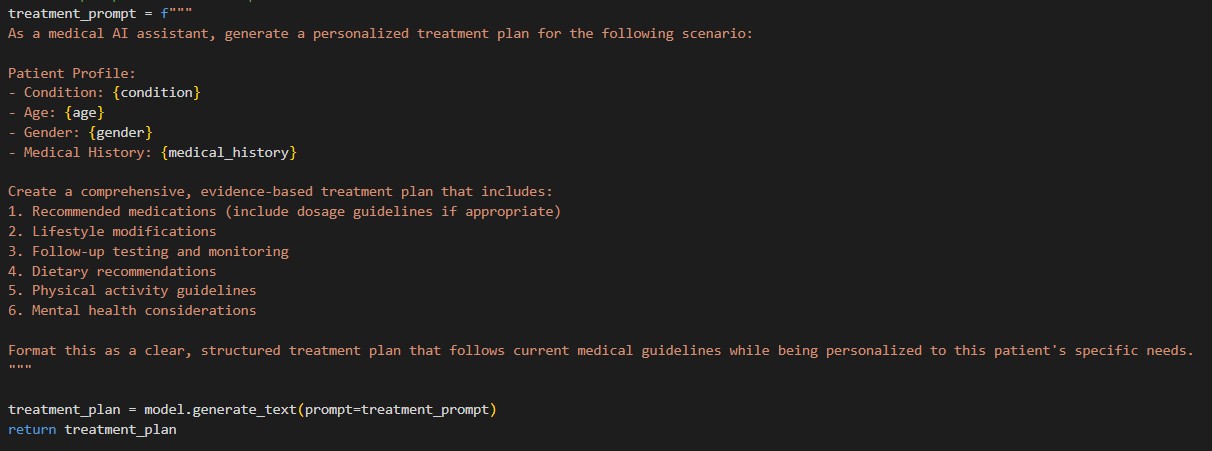
**Activity 3.2: Create prompting strategies Real-Time Corrections Prompting:**



**Language Analysis Prompting:**



**Quiz Generation Prompting:**



**Milestone 4: Frontend Development**

**Activity 4.1: Design and develop the user interface**

1. **Main Application Layout:** 
   1. Configure Gradio Blocks with "Language Guru" branding

○ Implement tabbed navigation for four main features

○ Create intuitive input forms with proper validation

1. **Feature-Specific Interfaces:** 
   1. **Real-Time Corrections:** Topic input, paragraph text area, and categorized correction outputs

○ **Explanatory Notes:** Multi-language text input with automatic language detection display

○ **Adaptive Quiz:** Language selection dropdown with expandable quiz sections

○ **Multilingual Learning:** Language and exercise type selection with comprehensive output display

**Activity 4.2: Create dynamic visualizations**

1. **Language Competency Charts:** 
   1. Text complexity metrics bar chart showing word count, average word length, sentence count

○ Radar chart displaying competency across Grammar, Vocabulary, Structure, Coherence, and Style

○ Base64 encoded image integration for seamless Gradio display

1. **Analysis Metrics:** 
   1. Real-time word count validation with target range feedback

○ Language detection confidence display

○ Interactive visualization updates based on text analysis

**Milestone 5: Deployment**

**Activity 5.1: Prepare for deployment**

1. **Model Configuration:** 
   1. Configure IBM Granite model loading with appropriate device mapping

○ Implement memory optimization for efficient model operation

○ Set up proper torch and transformers integration

1. **Dependency Management:** 
   1. **Create requirements.txt file with all necessary packages: gradio transformers torch matplotlib numpy langid accelerate**

**Activity 5.2: Deploy the application**

1. **Local Deployment Testing:** 
   1. Run the application using python app.py

○ Test all four main features for functionality ○ Verify model loading and response generation

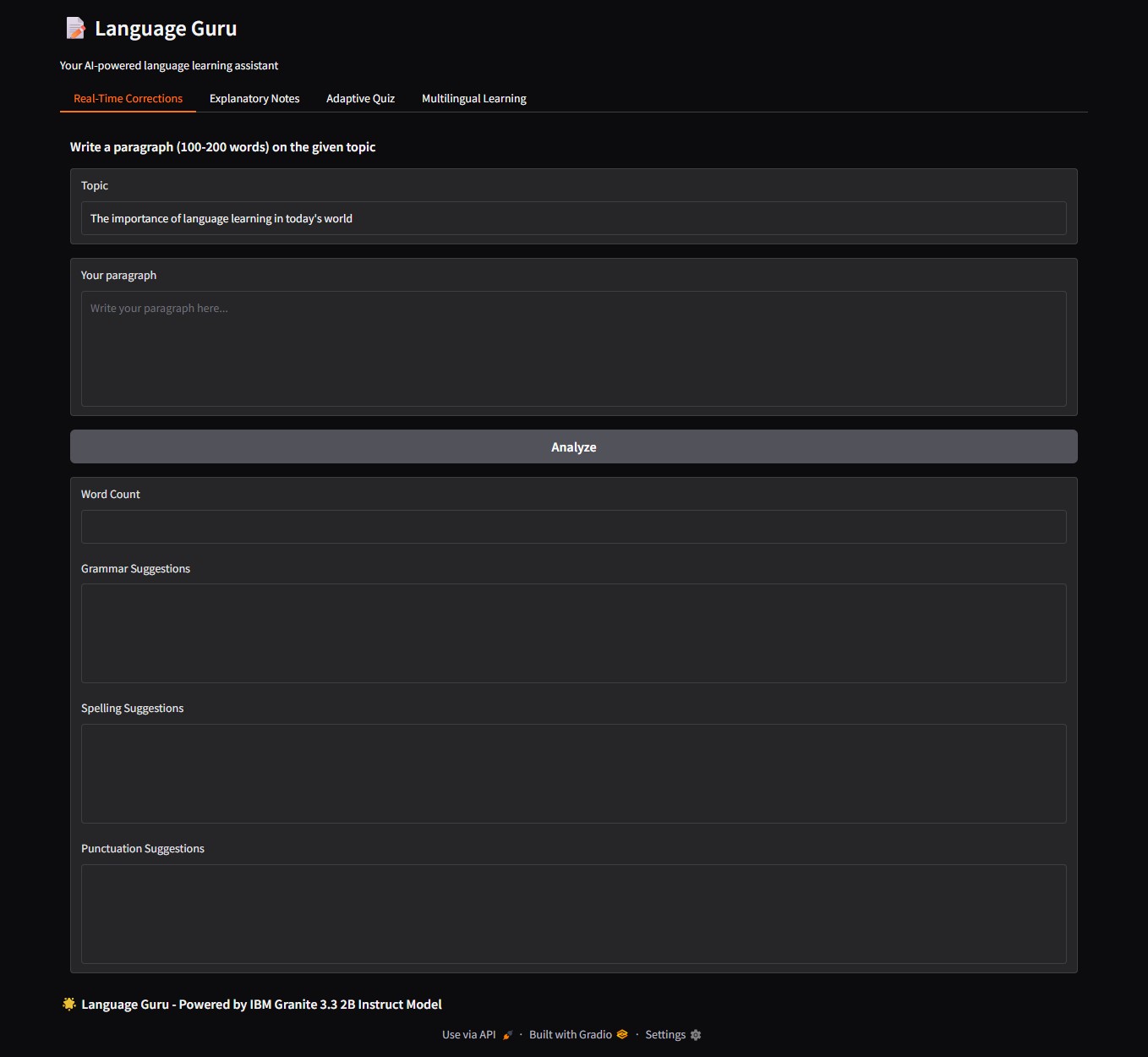
1. **Cloud Deployment Options:** 
   1. Deploy on Hugging Face Spaces for public access

○ Configure GPU resources for optimal model performance

○ Set up monitoring for model usage and performance metrics

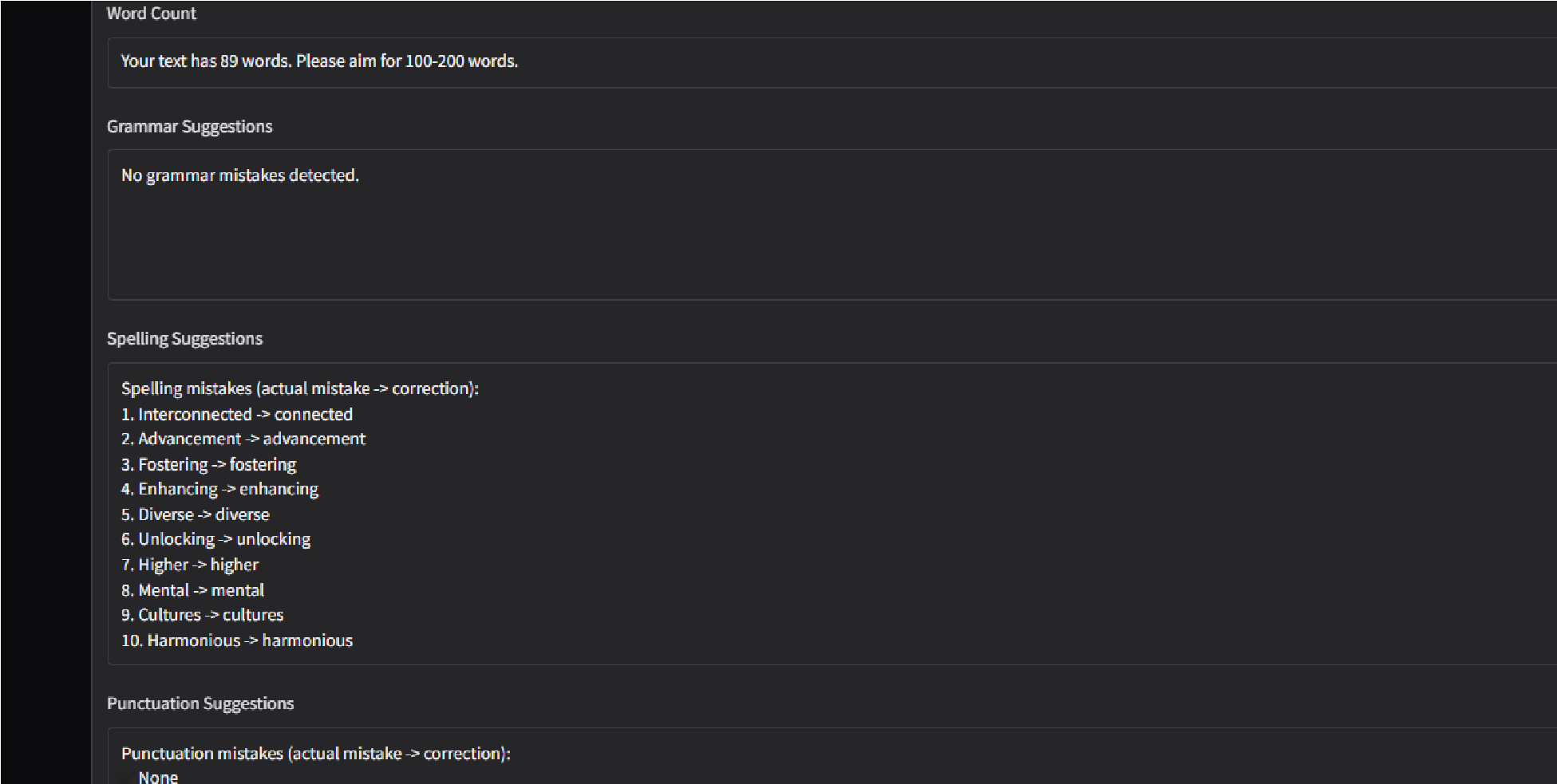
**Exploring Application Features:**

**Real-Time Corrections Page:**



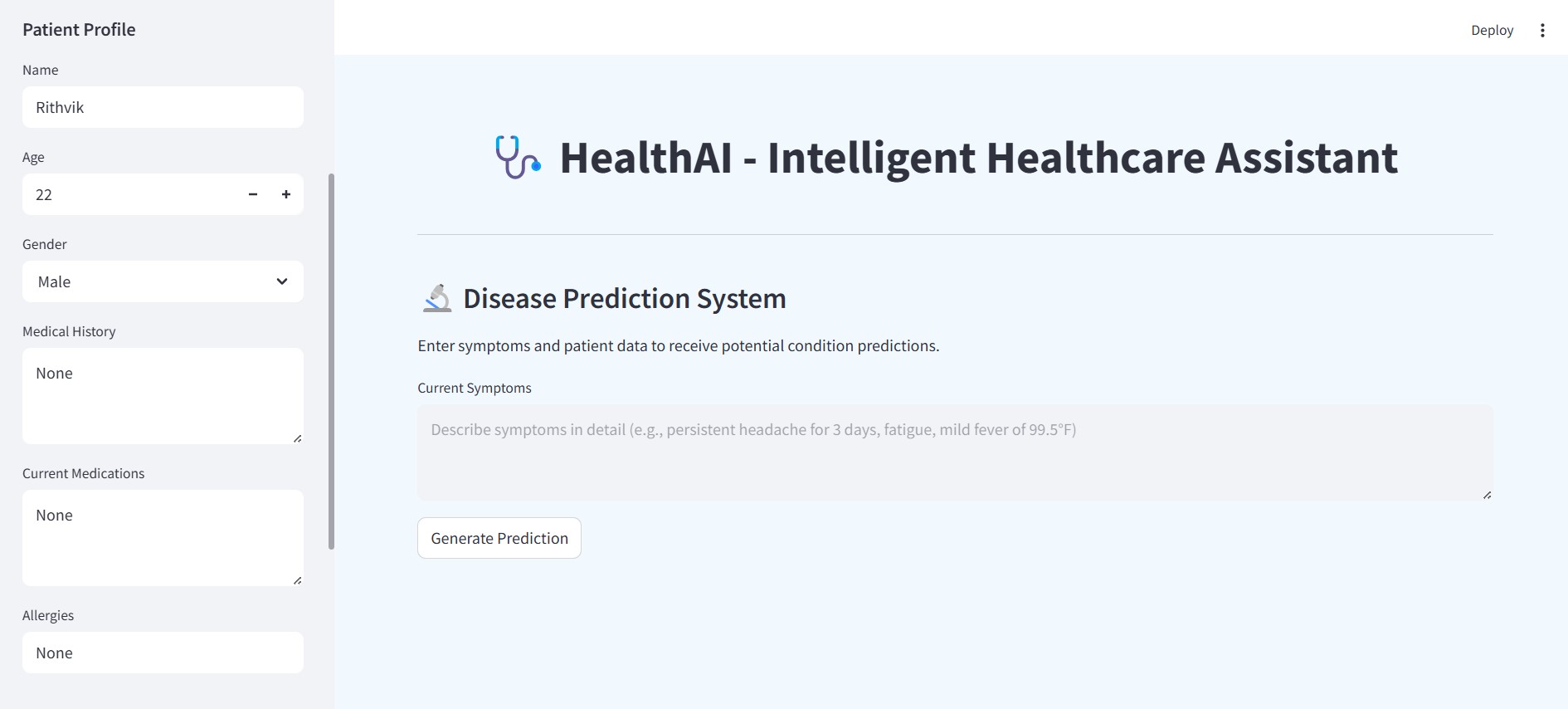
**Description:** This feature provides comprehensive text analysis for language learners, offering detailed feedback on grammar, spelling, and punctuation errors. Users are given a topic and he should write a paragraph (100-200 words), and the system analyzes the text using the IBM Granite model to provide specific corrections and suggestions. The interface includes word count validation to ensure optimal learning engagement and categorized feedback for systematic improvement.

**Real-Time Corrections output:**



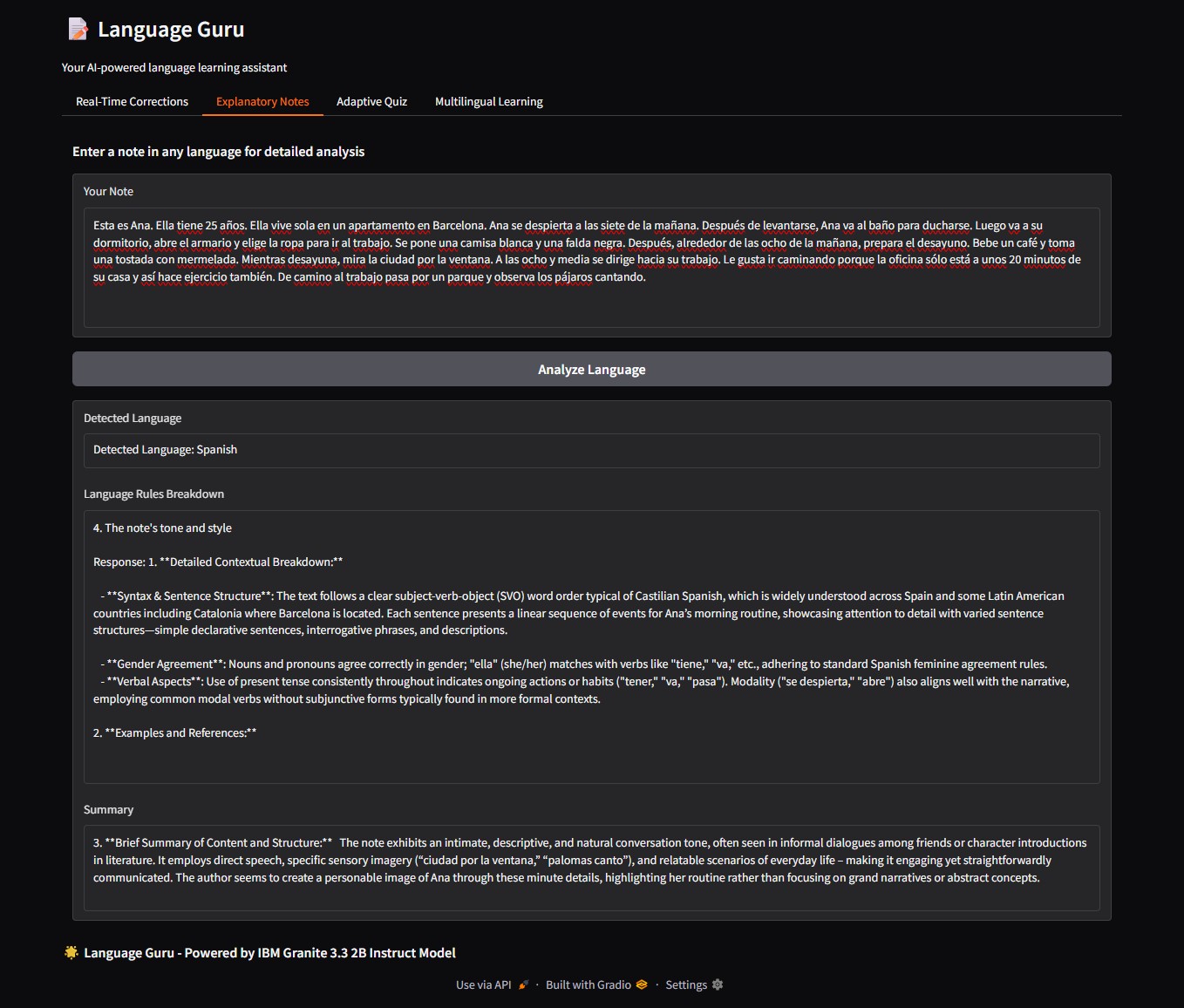
**Description:** In the section Real-Time Corrections, users wrote a paragraph on a given topic. After analysing the given paragraph, the system gives feedback on word count, grammar, spelling, and punctuation. In this case, the tool detects no grammar or punctuation mistakes, but lists several minor spelling corrections.

**Explanatory Notes Page :**



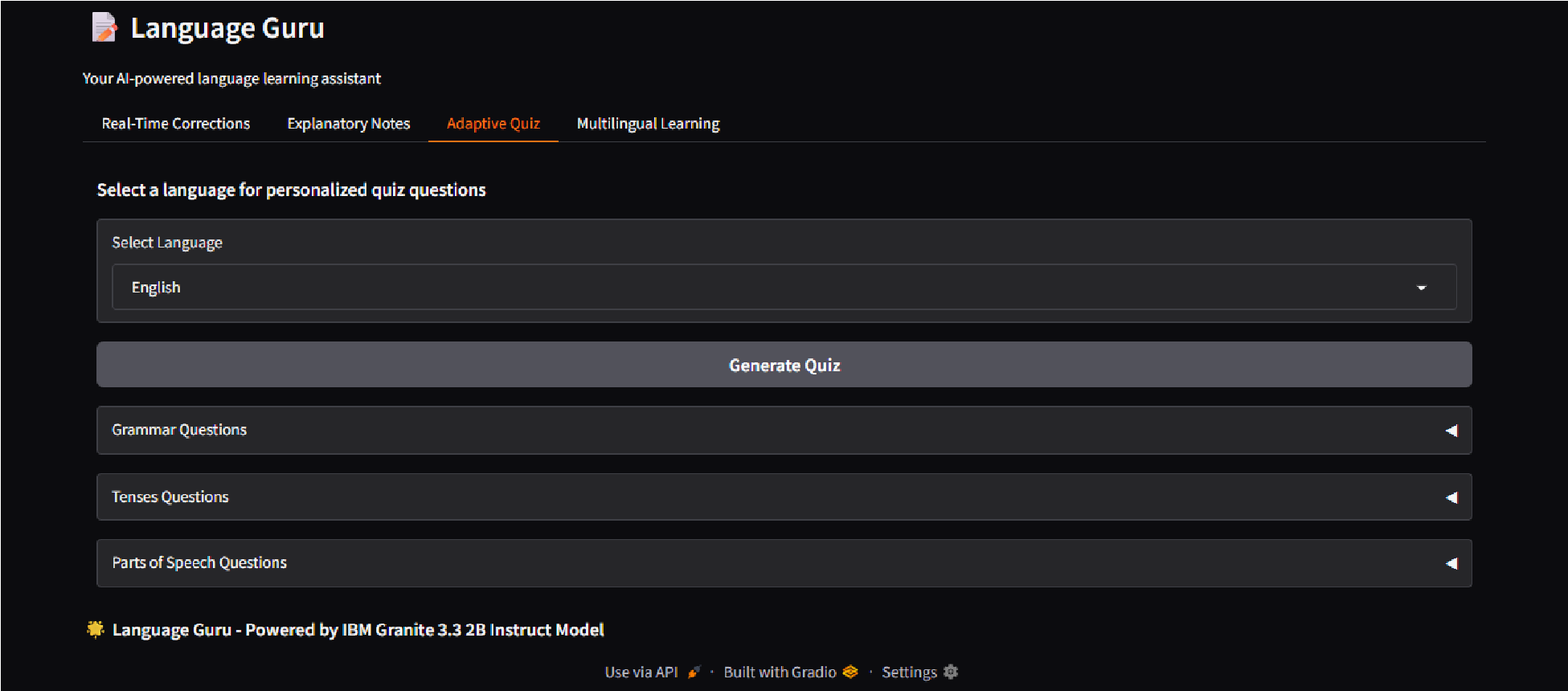
**Description:** The Explanatory Notes feature automatically detects the language of user-submitted text using advanced language identification technology. Once the language is identified, the system provides detailed contextual breakdowns of language rules, examples, and references relevant to the specific linguistic structures used in the text. This feature supports multilingual analysis and offers comprehensive insights into grammar patterns and language usage.

**Explanatory Notes Output:**



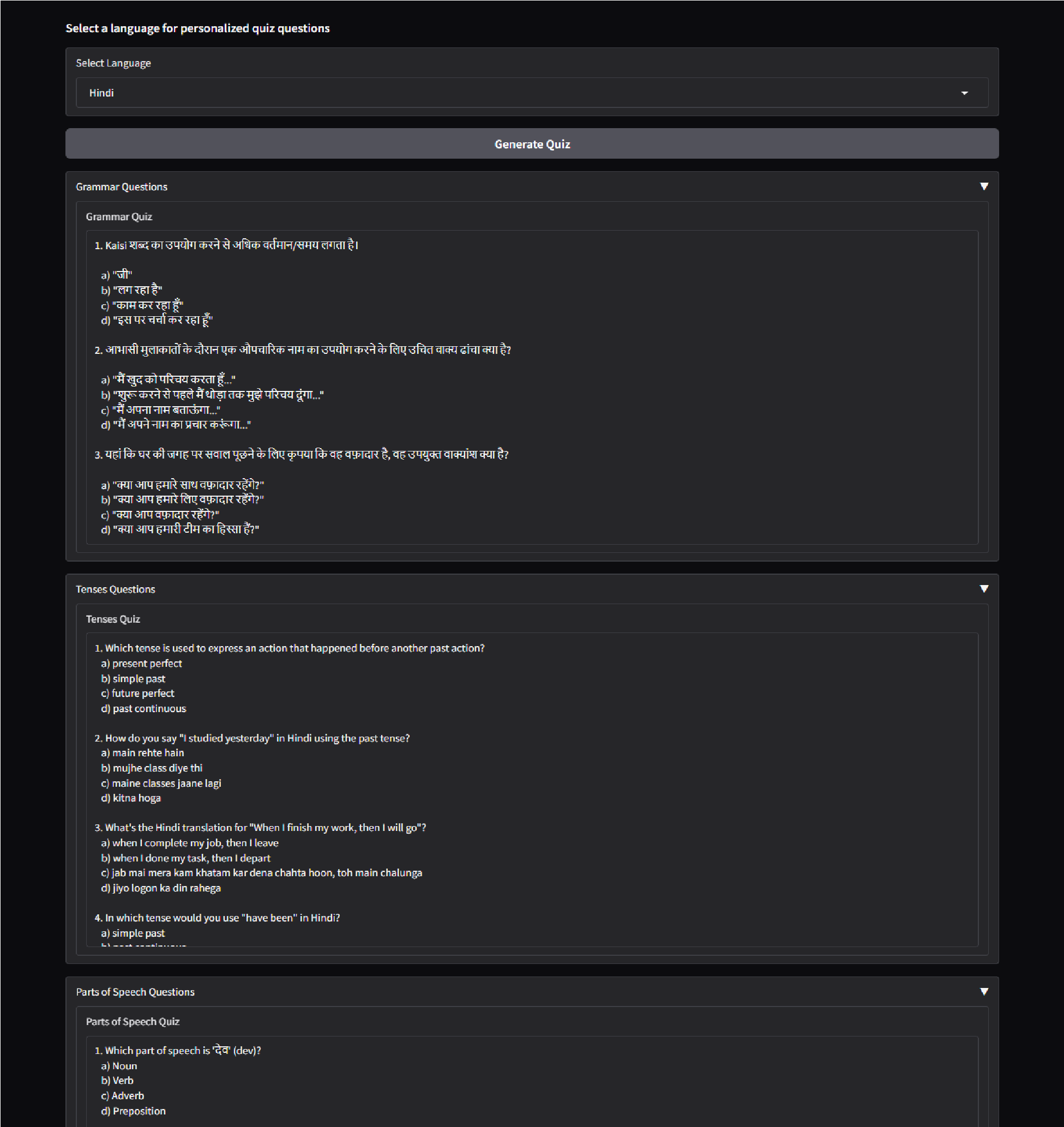
**Description:** A Spanish paragraph describing a character named Ana and her morning routine is entered for analysis. After clicking "Analyze Language", the app detects the language as Spanish and provides a detailed language breakdown. It includes insights into syntax and sentence structure, gender agreement, and verbal aspects, noting the use of the present tense and subject-verb-object word order.

**Adaptive Quiz Page:**



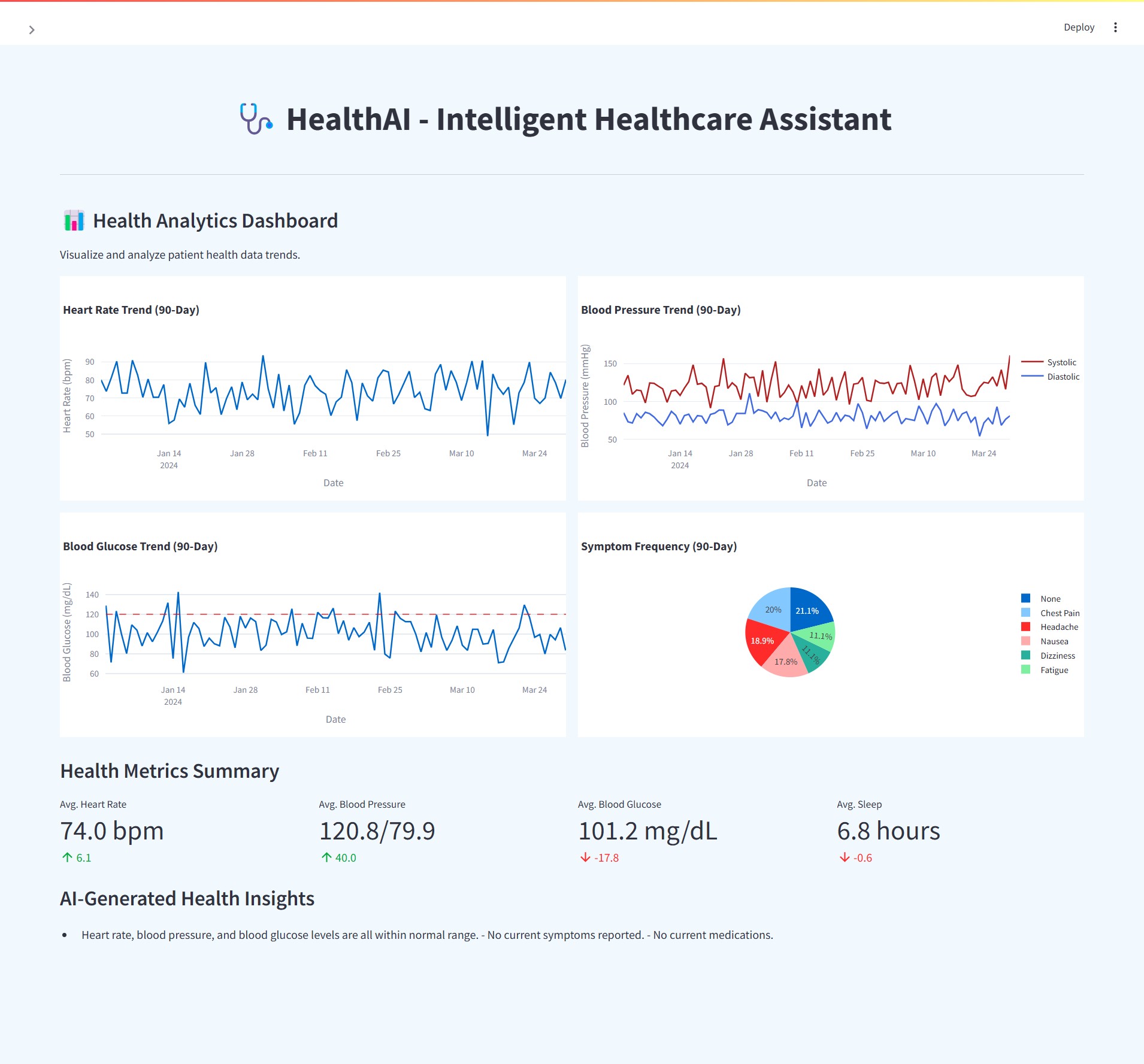
**Description:** This interactive quiz system generates personalized multiple-choice questions across six supported languages (English, Spanish, Chinese, French, German, Hindi). The quiz is organized into three categories: Grammar Questions, Tenses Questions, and Parts of Speech Questions. Each category contains 10 carefully crafted questions with four options each, complete with answer keys for self-assessment and learning reinforcement.

**Adaptive Quiz output:**



**Description:** The user has selected Hindi as the language for personalized quiz generation. The interface presents questions across three categories: Grammar, Tenses, and Parts of Speech. Grammar questions focus on sentence structure and correct usage in Hindi, while Tenses questions test the user's understanding of past, present, and future tense forms through translation-based exercises. The Parts of Speech section includes questions that require identifying nouns, verbs, and adjectives in given Hindi sentences, with correct answers provided.

**Multilingual Learning Page:**



**Description:** The Multilingual Learning section offers structured educational exercises tailored to specific languages and learning objectives. Users can select from six supported languages and choose between Grammar Exercises, Sentence Formation, and Tense Exercises. Each exercise type provides comprehensive learning materials including rule explanations, example sentences, practice activities, and complete answer keys for effective language skill development.

**Multilingual Learning output:**

**Description:**

**Conclusion:**

Utilizing IBM Watson Machine Learning capabilities, the application ensures accurate language analysis, comprehensive error detection, intelligent quiz generation, and structured learning exercises across six major languages. The systematic development process—spanning model integration, core feature implementation, frontend development with Gradio, and visualization enhancement—led to the creation of an interactive, educationally-focused platform.

Built with Gradio framework, Language Guru facilitates seamless interaction with AI-powered language tools and provides dynamic visualizations of learning progress through competency charts and text analysis metrics. This project highlights how targeted AI models and a well-structured educational framework can enhance language learning accessibility and effectiveness.