Technical Scoping Document: Funny Storyteller Chatbot

Project Title: Funny Storyteller Chatbot from Classic Literature

Date: 27-05-2025

Author/Developer: Shantanu Dhadwe

1. Introduction & Project Goal

This document outlines the technical design and implementation of a chatbot prototype designed to engage users with stories from classic public domain literature. The primary goal was to build a backend system and a user interface capable of:

- Understanding user queries related to specific books ("Alice in Wonderland," "Gulliver's Travels," "The Arabian Nights").
- Retrieving relevant story snippets from a knowledge base.
- Generating a humorous and engaging story based on the retrieved context.
- Providing an idea for an image related to the generated story.
- Generating and displaying this image.
- Handling queries unrelated to the source material with a funny "I don't know" response.
- Accepting user input via text or speech.
- Providing bot responses as text and synthesized speech.

The prototype demonstrates these capabilities through a Streamlit-based chat interface (app.py).

2. System Architecture & Technology Stack

The application follows a Retrieval Augmented Generation (RAG) architecture, with separate scripts for data ingestion and the main application logic.

Core Technologies Used:

- **Programming Language:** Python 3.x
- **Frontend Interface:** Streamlit (implemented in app.py)
    - *Why:* Rapid prototyping of interactive web applications, excellent for building chat interfaces, managing session state, and displaying text, images, and audio.
- **LLM Interaction (Langchain):** (used in app.py and data ingestion process)
    - langchain: Core framework for building LLM applications.
    - langchain_community, langchain_huggingface, langchain_chroma: For specific integrations.
    - *Why:* Simplifies LLM integration, prompt management, document loading (PyPDFLoader), text splitting (RecursiveCharacterTextSplitter), and vector store interactions.

- **Large Language Model (LLM):** mistralai/Mistral-7B-Instruct-v0.3 (or specified model) via HuggingFaceEndpoint in langchain_huggingface. (configured in app.py)
    - *Why:* A capable open-access instruction-tuned model suitable for complex generation (storytelling, humor) and classification (relevance check) tasks. HuggingFaceEndpoint allows usage without local GPU hosting for this model size.
- **Embedding Model:** sentence-transformers/paraphrase-multilingual-mpnet-base-v2 via HuggingFaceEmbeddings in langchain_huggingface. (used in data ingestion and app.py)
    - *Why:* A strong multilingual model, enabling user queries in various languages to be matched against the English-only source documents. Provides robust semantic understanding for retrieval.
- **Vector Database:** ChromaDB (langchain_chroma). Persistent storage configured in db/chroma_db/. (Created by an ingestion script like simple_rag.py or retrieve.py if it contains ingestion logic, loaded by app.py).
    - *Why:* Open-source, lightweight, and easy to integrate for storing document embeddings and performing semantic similarity searches.
- **Environment Management:** python-dotenv (used in app.py and helper scripts) for managing API keys from a .env file.
- **Audio Handling (audio_handler.py):**
    - **Speech-to-Text (STT):** speech_recognition library (utilizing Google Web Speech API via recognizer.recognize_google(audio)).
        - *Why:* Widely used, accessible library for converting spoken language to text. Includes ambient noise adjustment.
    - **Text-to-Speech (TTS):** ElevenLabs API via elevenlabs Python client. Voice ID pNInz6obpgDQGcFmaJgB (Adam) and model eleven_multilingual_v2 are used, streaming audio to BytesIO.
        - *Why:* ElevenLabs provides high-quality, natural-sounding voices and multilingual support, enhancing user experience.
- **Image Generation (image_generation.py):**
    - Hugging Face InferenceClient (huggingface_hub.InferenceClient).
    - Model: stabilityai/stable-diffusion-xl-base-1.0.
    - *Why:* Stable Diffusion XL is a powerful open-access text-to-image model. The InferenceClient allows API access to models hosted on the Hugging Face Hub. Generates PIL Image objects.
- **Data Ingestion (Separate Script, e.g., based on simple_rag.py or the ingestion part of retrieve.py):**
    - Loads PDF documents from a local /documents directory.

- Uses PyPDFLoader to extract text.
- Employs RecursiveCharacterTextSplitter (chunk size 1000, overlap 150) for document chunking.
- Generates embeddings for chunks using the specified multilingual model.
- Creates and persists a ChromaDB vector store in db/chroma_db/ if one doesn't already exist.

3. Application Flow (app.py)

The Streamlit application orchestrates the user interaction in two main phases: initial setup (once per session) and per-query processing.

A. Initial Setup & Caching (on app start/first interaction):

1. **Environment Variables:** load_dotenv() loads API keys.
2. **Resource Initialization:** @st.cache_resource is used to load/initialize:
   - HuggingFaceEmbeddings model.
   - ChromaDB vector store (from persistent_dir).
   - HuggingFaceEndpoint and ChatHuggingFace model for LLM interactions.
   - Langchain retriever from the ChromaDB instance.
3. **Error Checks:** Critical components (DB, LLM) are checked; if loading fails, an error is displayed, and the app stops.
4. **Session State:** st.session_state initializes/maintains:
   - messages: List to store and display chat history.
   - memory: Langchain ConversationBufferMemory for contextual LLM calls.
   - audio_to_play: For handling TTS playback.

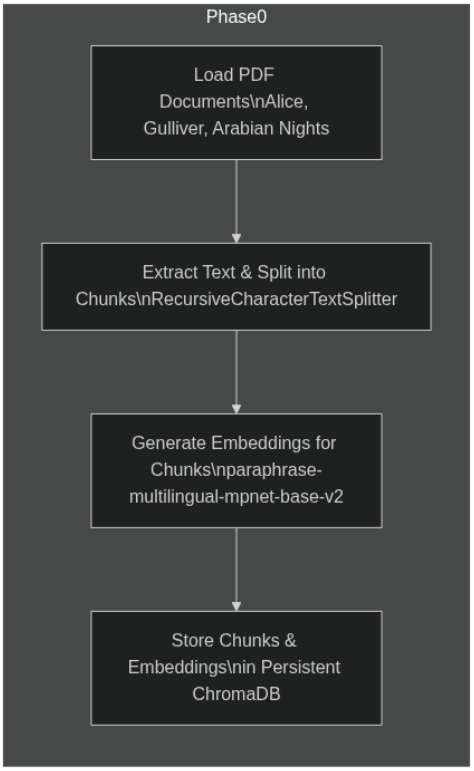
B. Per-Query Processing (triggered by user input):

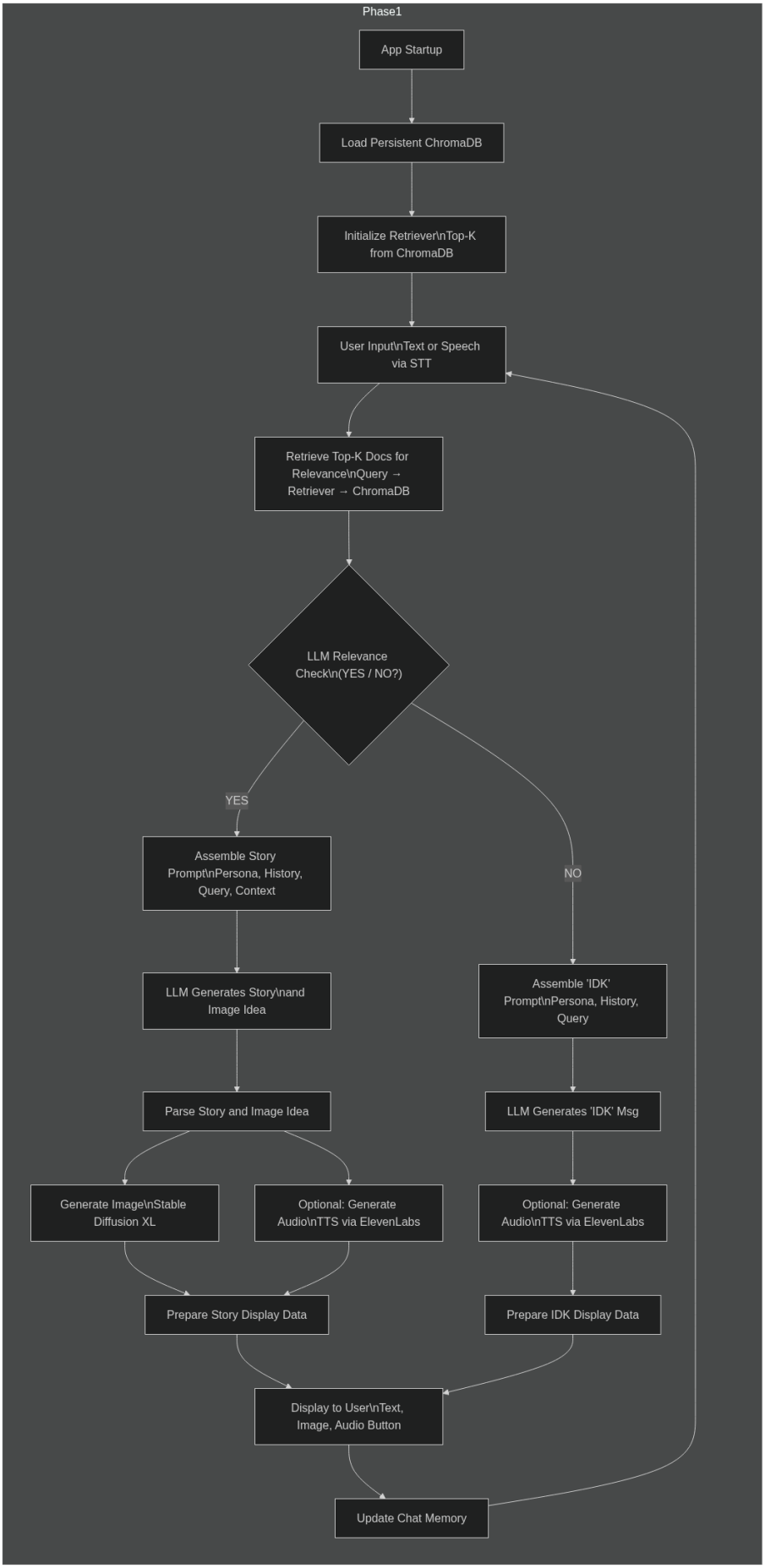
1. **User Input Acquisition:**
   - Text input via st.text_input (form submission).
   - Speech input (if AUDIO_ENABLED): "Speak" button triggers audio_handler.speech_to_text_from_mic.
   - The recognized/typed query is added to st.session_state.messages.
2. **Context Retrieval for Relevance Check:**
   - The user_query_from_input is passed to the global retriever.
   - Top K (currently 3 or 5, as per retriever config) document chunks are fetched from ChromaDB.
   - format_docs prepares context_for_relevance_check_str.

3. **LLM-Based Relevance Classification:**
    - RELEVANCE_SYSTEM_PROMPT_TEXT (few-shot) and the user_query_from_input + context_for_relevance_check_str are formed into SystemMessage and HumanMessage.
    - These messages are sent to model.invoke() to get a "YES"/"NO" relevance classification.
    - The LLM's raw string output is processed with a heuristic to determine a boolean is_relevant.
4. **Conditional Response Generation:**
    - **Path A: Query is Relevant (is_relevant == True):**
        1. **Story Prompting:** story_prompt_messages_builder creates messages including system persona, current chat_history (from st.session_state.memory), the user_query_from_input, and the context_for_relevance_check_str (re-used).
        2. **LLM Story Generation:** model.invoke() generates the story.
        3. **Parsing:** parse_story_response extracts the "Witty Tale" and "Whimsical Image Idea".
        4. **Image Generation:** image_generation.generate_img() is called with the image idea.
        5. **Audio Output:** audio_handler.text_to_speech_elevenlabs() synthesizes the story text.
    - **Path B: Query is Not Relevant (is_relevant == False):**
        1. **IDK Prompting:** idk_prompt_messages_builder creates messages for a funny "I don't know" response, including chat history and the query.
        2. **LLM IDK Generation:** model.invoke() generates the IDK message.
        3. **Audio Output:** The IDK message is synthesized by audio_handler.text_to_speech_elevenlabs().
5. **Output Display & Memory Update:**
    - The bot's textual response, generated image (PIL object), and audio data (BytesIO object) are packaged into a dictionary.
    - This dictionary is appended to st.session_state.messages.
    - User input and bot text output are saved to st.session_state.memory.
    - st.rerun() updates the Streamlit UI to display new messages, image, and audio play button.

4. Key Design Decisions & Logic

- **Data Ingestion:** Handled by a separate script (e.g., simple_rag.py / retrieve.py containing ingestion logic) to create a persistent ChromaDB, making the main app.py load faster. Uses RecursiveCharacterTextSplitter for effective chunking of literary texts.
- **LLM for Relevance Classification:** A few-shot prompted LLM call is used to determine if retrieved context is suitable for answering the query, aiming for more semantic relevance detection than simple score thresholds.
- **Context Re-use:** Retrieved context for the relevance check is passed directly to the story generation step if the query is relevant, avoiding a redundant retrieval call.
- **Chat History:** Langchain's ConversationBufferMemory and Streamlit's session_state are used to maintain conversation context, which is included in prompts to the LLM, enabling more natural (though currently simple) follow-up potential.
- **Modular Features:** Speech input/output (audio_handler.py) and image generation (image_generation.py) are designed as optional modules, allowing the core text-based chatbot to function even if these dependencies or their API keys are unavailable.
- **Streamlit Caching & State:** @st.cache_resource optimizes loading of models and DB. st.session_state is integral for the interactive chat experience.
- **User Experience:** A clear chat interface with text input, a "Speak" button, image display, and audio playback buttons for an engaging multimedia experience. A "Clear Chat" feature is provided.




Phase1
App Startup
Load Persistent ChromaDB
Initialize Retriever\nTop-K from ChromaDB
User Input\nText or Speech via STT
Retrieve Top-K Docs for Relevance\nQuery → Retriever → ChromaDB
LLM Relevance Check\n(YES / NO?)
YES
NO
Assemble Story Prompt\nPersona, History, Query, Context
LLM Generates Story\nand Image Idea
Parse Story and Image Idea
Generate Image\nStable Diffusion XL
Optional: Generate Audio\nTTS via ElevenLabs
Prepare Story Display Data
Assemble 'IDK' Prompt\nPersona, History, Query
LLM Generates 'IDK' Msg
Optional: Generate Audio\nTTS via ElevenLabs
Prepare IDK Display Data
Display to User\nText, Image, Audio Button
Update Chat Memory

6. Potential Future Improvements

- **Refined Relevance Logic:** Improve parsing of the LLM's YES/NO relevance output or explore alternative relevance detection methods if current heuristics are insufficient.
- **Advanced RAG:** Implement techniques like Parent Document Retriever for better context.
- **Error Handling:** Enhance error handling for API calls and external services.
- **Streaming:** Implement streaming for LLM responses and TTS for better perceived UI responsiveness.

7. Conclusion

This prototype effectively demonstrates a RAG-based chatbot with a humorous persona, multimedia output, and voice interaction capabilities. It successfully integrates various open-source tools and APIs within a Langchain and Streamlit framework, providing an engaging way to interact with classic literature.
