Chefbot Multimodal Cooking Assistant

BY

Lama Alwuthaynani

Renad Asiri

Abctract

Chefbot is an advanced multimodal ai assistant designed to enhance the cooking experience through video analysis and natural language interaction. By combining transcript processing, embedding techniques, and retrieval-augmented generation (rag), chefbot delivers contextually relevant cooking guidance based on video content and general culinary knowledge. The system processes youtube cooking videos, extracts key information, and enables users to ask specific questions about recipe steps, ingredients, techniques, and general cooking advice.

i. INTRODUCTION

1.1 Background

Traditional recipe platforms often lack interactive guidance and contextual understanding. While cooking videos provide visual instruction, they require constant pausing and rewinding to follow along. Text-based assistants lack visual context, while video platforms lack intelligent q&a capabilities. Chefbot bridges this gap by creating a multimodal experience that combines video content analysis with conversational ai.

1.2 Problem Statement

- Users struggle to follow cooking videos efficiently
- Text-only recipe applications lack visual demonstration
- Finding specific information within cooking videos is time-consuming
- Existing cooking solutions fail to offer personalized guidance tailored to individual needs.

1.3 Objectives

1. Build a multimodal assistant capable of processing cooking videos.
2. Create a conversational interface for recipe-specific questions.
3. Develop a knowledge retrieval system optimized for cooking content.
4. Provide contextually relevant answers from video transcripts.
5. Enhance cooking experience with supplementary features (shopping list, unit conversion).

1.4 Scope

- Audio processing and transcription from youtube videos.
- Text extraction and embedding from video transcripts.
- Text extraction and embedding from video transcripts.
- Language model integration for conversational responses.
- Web interface for user interaction.

II. PROJECT GOALS

The goal of chefbot is to assist individuals in better understanding and following cooking recipes from youtube videos by transforming passive video content into an interactive and accessible experience.

By extracting transcripts and enabling users to ask questions about the video, chefbot allows users to easily find specific steps, ingredients, cooking times, and techniques without having to watch the entire video. This is especially helpful for people who prefer reading over watching, who may be in noisy environments, or who need to quickly revisit a part of the recipe.

The aim is to make cooking more efficient, inclusive, and enjoyable by leveraging ai to bridge the gap between video content and user needs in real time.

III. TOOLS & TECHNOLOGIES

- **Openai gpt-3.5**: core language processing and generation
- **Whisper**: audio transcription from video content
- **Text-embedding-ada-002**: vector embeddings for semantic search
- **Streamlit**: web application interface and deployment
- **Langchain**: framework for llm application development
- **Pinecone**: vector database for efficient similarity search
- **Yt-dlp**: youtube video downloading
- **Ffmpeg**: audio format conversion
- **Langsmith**: tracing and monitoring for model outputs
- **Langdetect**: language detection for multilingual support
- **Pinecone vector database**: cloud-based vector store
- **Openai api**: model inference and embedding generation

IV. SYSTEM ARCHITECTURE

Chefbot is built using a modular architecture that supports the flow from video ingestion to intelligent q&a. The main components are:

4.1 Transcript processing module:

This module extracts the transcript of a youtube cooking video. The system uses openai whisper to transcribe the video content. The transcript is then processed to ensure it's clean and usable for the next steps, which includes segmenting the text into manageable chunks using **recursivecharactertextsplitter** from langchain. This text splitter is particularly effective for chunking large amounts of text into smaller, contextually meaningful pieces, making it easier for the system to retrieve relevant information later.

4.2 Vector storage module:

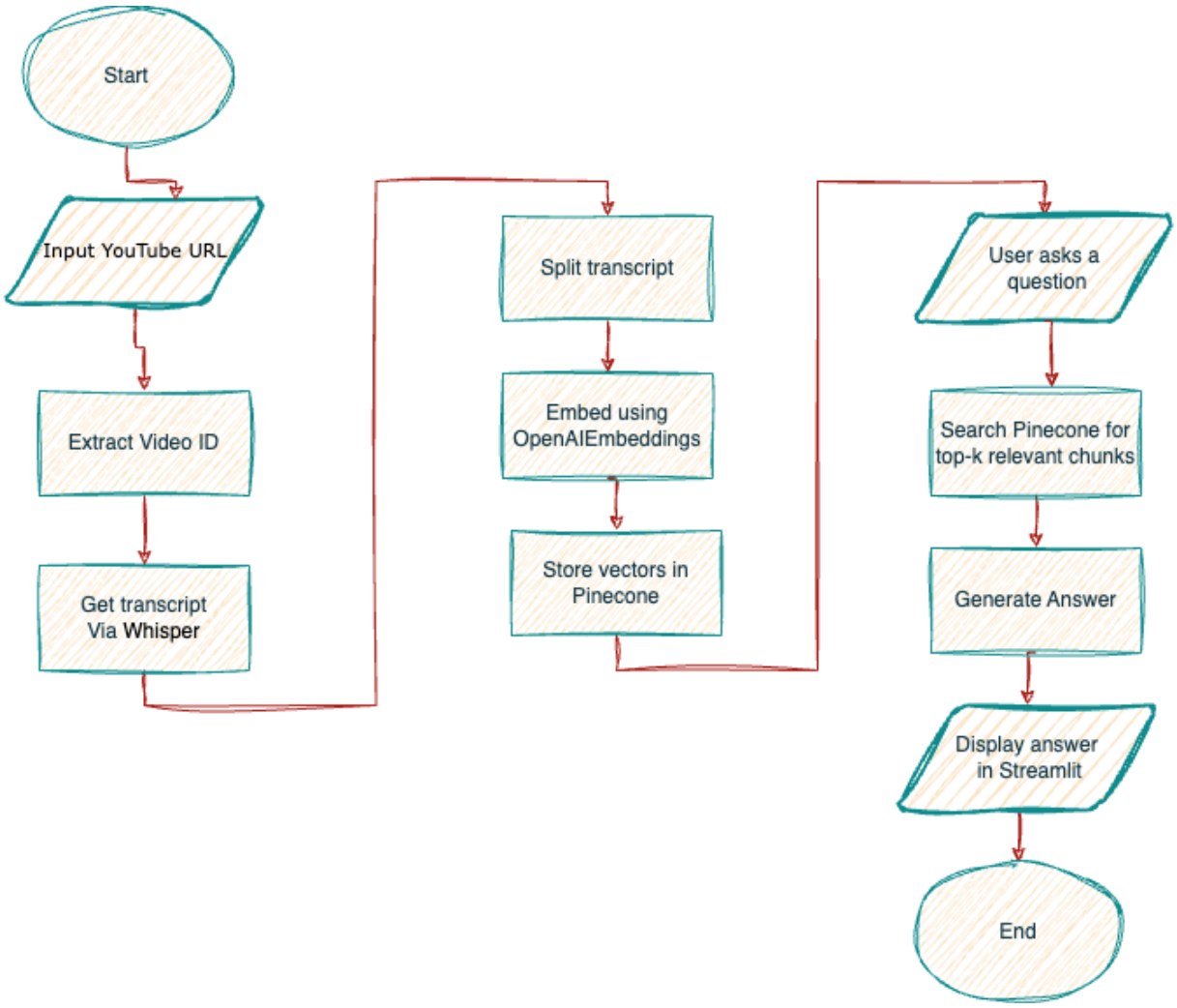
After chunking, each chunk is embedded using openaiembeddings(model="text-embedding-ada-002") and stored in pinecone, a vector database optimized for fast, semantic search. This enables efficient retrieval of relevant transcript chunks when the user asks a question.

4.3 Rag system (retrieval-augmented generation):

The rag system is the core of chefbot's intelligence. When a user asks a question, the system performs a similarity search in the pinecone vector database to retrieve the most relevant transcript chunks. These chunks are then passed into openai gpt for context-aware response generation.

4.4 User interface (streamlit interface):

The user interacts with chefbot through a streamlit-based web interface. The interface allows users to input youtube video urls, ask recipe-related questions, and receive answers generated by the system in real-time. This interface is designed to make the interaction smooth and intuitive.



V. METHODOLOGY & WORKFLOW

5.1 Development Process

This project was developed as a cooking assistant powered by retrieval-augmented generation (rag), built with python and streamlit. It integrates youtube transcript analysis with natural language interaction to help users explore recipes, ask cooking questions, and generate shopping lists.

5.2 Data Flow Or Pipelines

1. Transcript Collection: Transcripts Are Obtained From Cooking Videos And Stored In A Vector Store (Pinecone) After Preprocessing.

2. Vector Embedding: Each Transcript Is Embedded Using Openai's Embedding Model And Indexed

3. User Query Handling:
   - The User Inputs A Cooking-Related Question.
   - The System Searches Pinecone Using The Query Embedding To Find Relevant Transcript Chunks.
   - Results Are Reranked And Formatted As Context.

4. LLM Response Generation:
   - A Prompt With System Instructions, User Question, And Transcript Context Is Built.
   - The Prompt Is Passed To Openai's Chatgpt (Via Langchain) To Generate A Response

5. Post-Processing:
    - Ingredients Are Extracted.
    - Units Are Converted.

5.3 Algorithms- Models Used

- Embedding model: openai's text-embedding-ada-002
- LLM: openai's gpt-3.5-turbo
- Language detection: langdetect
- Rag framework: langchain
- Vector search: pinecone
- Reranking: custom function prioritizing most relevant transcript chunks

VI. IMPLEMENTATION & FEATURES

6.1 Key Modules

- Transcript_processor.py: handles cleaning and segmenting video transcripts.
- Vector_store.py: adds video data to pinecone, queries vector store, formats results.
- Rag.py: main logic for asking questions with context injection, model prompting, and response post-processing.
- Streamlit_app2.py: streamlit interface for user interaction (chat, input, display).
- Evaluate.ipynb: notebook for manual and automated evaluation of system answers.

6.2 Highlighted Features

- Rag-enabled chatbot: answers based on video content with fallback to general culinary knowledge.
- Ingredient extractor: builds a shopping list from generated answers.
- Multilingual support: detects user's language and responds accordingly.
- Unit conversion: standardizes measurement units in responses.

VII. CHALLENGES & SOLUTIONS

7.1 Transcript Quality And Structure

⇒Challenge:
youtube transcripts often come in messy formats, with missing punctuation or sentence structure. This made it hard to extract clear steps or accurate information.

⇒Solution:
we applied chunking with overlap using a smart text splitter. This preserved context across the transcript and ensured better input for embeddings and the language model.

7.2 Gtting Relevant Results From The Vector Store

⇒Challenge:
sometimes, the pinecone vector search returned chunks that weren't highly relevant to the user's question.

⇒Solution:
we implemented a custom reranking method that combined the original similarity score from pinecone with a term-overlap score. This improved the relevance of the top results significantly.

7.3 Keeping Answers In The User's Langchallenge:

⇒Challenge:
Since users might ask questions in different languages, we needed a way to make sure the answers were returned in the same language.

⇒Solution:
we used the langdetect library to detect the language of the user's question and passed this information to the prompt. This allowed the assistant to respond in the appropriate language automatically

VIII. EVALUATION

8.1 Evaluation Setup

The model was evaluated using the langsmith evaluation framework. A small dataset named recipe_qa_dataset_v3 was created with three examples each example included:

- A question
- A youtube video id
- An expected answer

Sample questions included:

- “what is the ingredient for the recipe?”
- “how long and at what temperature should i bake the pie?”

These examples were uploaded to langsmith to test the rag-based qa system's ability to retrieve and generate accurate responses.

8.2 Evaluation Metrics

Three rounds of evaluation were conducted using different evaluators and gpt models:

1. Answer presence (is answered)
    - **Evaluator name:** is_answered
    - **Model used:** gpt-4
    - **Goal:** ensure the model outputs a non-empty and contextually relevant answer.
    - **Result:** all model responses were marked as meaningful attempts (score: 1).
    - **Experiment name:** test-recipe-qa-eval-is-answered-llm-gpt-4
2. Answer correctness (manual grading with explanation):

- **Evaluator name:** answer_correctness
- **Model used:** gpt-4
- **Scoring:** 0 or 1 based on factual correctness, aligned with provided ground-truth answers
- **Outcome:**
    - Responses were factually correct and aligned with expected answers.
    - Explanations were automatically generated using gpt-4.
- **Experiment name:** correctness-eval-gpt-4
- **Score summary:** all responses received a correctness score of 1.

This round tested the rag system's ability to generate factually accurate answers using a custom evaluator.

8.3 Methodology:

- Target function: the ask_question() function was used as the model under test.
- LLM evaluation: a GPT-4 model was used to verify if the generated answer was a "real attempt" at answering the question.
- Metric used: binary scoring system
- Evaluator logic:
    - Score = 0 if the answer was empty or irrelevant
    - Score = 1 if the response was a genuine attempt to answer the question

IX. CONCLUSION

This project demonstrates a robust ai-powered cooking assistant that combines retrieval-augmented generation with real-world youtube cooking videos. Through semantic search, dynamic prompting, and intuitive ui features like ingredient extraction and favorites saving, the system offers a helpful, friendly, and efficient tool for anyone interested in cooking. The modular design and high evaluation performance validate the effectiveness of combining langchain, openai, and pinecone for real-world applications.