Cook Book

Human Presentator: From Voice to Animated Avatar

Markus Siegert & Lucas Buchholz

Student ID: D533 & D484

Bachelor of Science Informatik

Primary Examiner: Richard Beetz

Deadline: 22.08.2025

Contents

List of Figures **iv**

Acronyms **v**

1 Introduction **1**

2 Methodology **1**

2.1 Usage Recipe 2

2.2 Development Recipe 4

3 Results **5**

4 Discussion **6**

List of Figures

1 User Interface of the Human Presentor 3
2 Architecture of the Human Presentor 6

Acronyms

API	Application Programming Interface.
CLI	Command-Line Interface.
DTIT	Deutsche Telekom IT GmbH.
memo	memoavatar/memo.
Zonos	Zyphra/Zonos.

1 Introduction

This cookbook presents a fully local pipeline for generating talking-head animations from static images and arbitrary text, leveraging Zyphra/Zonos (Zonos) for voice cloning and text-to-speech and memoavatar/memo (memo) for precise lip-synchronization. Designed to run entirely on your own hardware, this application ensures data privacy, offline functionality, and the flexibility to customize every component.

In today's media landscape, immersive and personalized audiovisual content drives engagement across educational platforms, virtual assistants, and marketing campaigns. By combining high-fidelity voice cloning with accurate mouth animation, creators can produce dynamic videos without relying on cloud services or external APIs. This guide walks you through setting up the environment, preparing your data, and executing each stage of the pipeline, from extracting a speaker's voice characteristics to animating a still image's mouth movements to match the synthesized speech.

At the core of this pipeline are two specialized tools: Zonos and memo. Zonos provides an end-to-end solution for voice cloning and speech synthesis, enabling you to input a short audio sample and generate natural-sounding speech for any text. memo handles the visual side, mapping the generated audio waveform onto a static image to produce lipid-accurate mouth movements. You'll find detailed instructions for installing dependencies, configuring models, and running inference scripts, alongside code snippets and troubleshooting advice.

While several open-source and commercial solutions address parts of this workflow, most depend on cloud infrastructure or separate services. Projects like Real-Time-Voice-Cloning and Mozilla TTS excel at local voice synthesis, and Wav2Lip and SyncNet offer speaker-agnostic lip-sync capabilities. Commercial tools such as Adobe Character Animator or CrazyTalk let users animate images without code, whereas platforms like Synthesia and D-ID provide end-to-end video creation online. This cookbook demystifies the underlying algorithms, empowering you to implement a cohesive, offline solution that you can extend for research or production purposes.

2 Methodology

The human-presentator project was initially conceived as a streamlined, single-script Python application designed to generate talking-head animations from static images and text input. The original approach emphasized simplicity and cohesion, with all functionality consolidated into one comprehensive Python script that would handle the entire pipeline from text input to final video output.

Rather than developing proprietary solutions from scratch, the project methodology centered on leveraging established open-source software for the core voice generation and image animation components. This approach was chosen to reduce development time by building upon proven, well-tested codebases, ensure compatibility with existing workflows and standards and maintain transparency and auditability through open-source implementations.

An integral component of the original methodology included implementing automated validation mechanisms for the generated MP4 output files. The automated validation approach aimed to ensure consistent output quality while reducing manual review overhead during the development and testing phases.

The initial architectural approach emphasized a monolithic design pattern, consolidating all processing stages within a single executable unit. This methodology was selected to minimize deployment complexity, reduce inter-component communication overhead, and provide a clear, linear processing flow that could be easily understood and modified by developers working on the project.

2.1 Usage Recipe

Usecase: Generate a talking-Presenter animation from a static image, text and audio input.

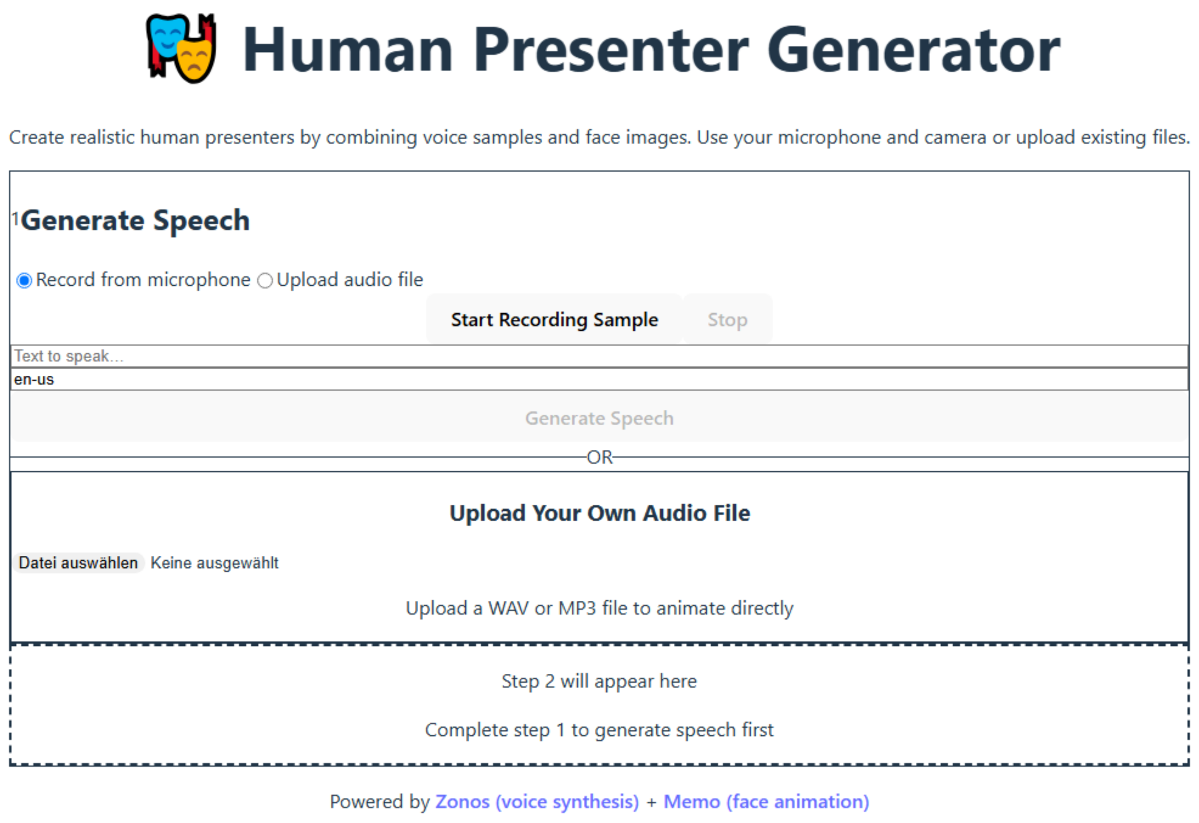
Prequisites: Linux, Docker or Podman, NVIDIA GPU with CUDA support.

Time: Approximately 10 minutes for setup and 5min-24h in execution.

1. Clone Repository

   ```
   git clone https://github.com/Squad-Mandalore/human-presentator.git
   ```

2. Change directory to the project root: `cd human-presentator`
3. Build the Application: `docker compose up`
4. Open the User-Interface in your browser: `http://localhost:8000`



Human Presenter Generator

Create realistic human presenters by combining voice samples and face images. Use your microphone and camera or upload existing files.

1Generate Speech

● Record from microphone ○ Upload audio file

Upload Your Own Audio File

Upload a WAV or MP3 file to animate directly

Step 2 will appear here

Complete step 1 to generate speech first

Powered by Zonos (voice synthesis) + Memo (face animation)

Figure 1: User Interface of the Human Presentor

1. Prepare your data:
2. (optional) Step 1: Generate Audio
    - Select Record from microphone or Upload Audio File
    - Insert the text you want the speaker to say.
    - Select `Generate Speech`
    - Wait for the audio file to be generated.
3. Step 2: Generate Lip-Sync
    - Upload Generated Audio File or any pre-recorded audio file.
    - Upload a static image or use Capture Face Snapshot for a presentor image.
    - Select `Generate Video`
    - Wait for the video to be generated.

2.2 Development Recipe

How to use the application components for own development or research purposes. The Structure of the human-presentator is defined into multiple directories, each containing a specific component of the application. This Recipe can also be used to rebuild the human-presentator project from scratch.

Usecase: Develop or extend the human-presentator project for research or production purposes.

Prequisites: Linux, Python, NodeJS, Docker or Podman, NVIDIA GPU with CUDA support, Python.

Time: Unkown, depends on the development task.

The main components are:

- `zonos-api`: Voice cloning and text-to-speech components, utilizing Zonos.
- `memo-api`: Lip-sync and animation components, utilizing memo.
- `presentator`: Development Script for experimenting with zonos- and memo-api.
- `presentor-vue`: Web user interface for interacting with the zonos- and memo-api.
- `presentation`: Presentation of the human-presentator project.
- `documentation`: Documentation of the human-presentator including this cookbook.

zonos-api: The zonos-controller is used as endpoint for utilizing zonos-service for voice cloning and text-to-speech components. The zonos-service is responsible for handling the voice cloning and text-to-speech requests, processing the audio / text input, and returning the generated audio output. For this purpose Zonos have been forked and installed as dependency to implement into the zonos-service.

memo-api: The memo-controller is used as endpoint for utilizing memo-service for lip-sync and animation components. The memo-service is responsible for handling the lip-sync and animation requests, processing the audio / image input, and returning the generated video output. For this purpose memo have been forked and installed as dependency to implement into the memo-service.

presentor-vue: Contains the Vue.js frontend application that provides a user interface for interacting with the zonos-api and memo-api services. The frontend allows users to upload audio files, static images, and text input, and displays the generated video output.

3 Results

The implementation of the human-presentator project deviated significantly from the initially planned monolithic approach described in the methodology section. During development, several technical constraints and compatibility issues necessitated a fundamental architectural redesign that resulted in a modular, service-oriented system.

The most significant change from the original methodology was the transition from a single-script Python application to a distributed 'microservice' like architecture. This shift was primarily driven by Python version incompatibilities between the core dependencies, specifically Zonos for voice synthesis and Memo for face animation. Rather than attempting to reconcile these version conflicts within a monolithic structure, the implementation adopted a containerized approach using Docker Compose to isolate each service in its own environment with appropriate Python versions and dependencies.

The final implementation consists of three primary services: a Vue.js frontend (presentor-vue) serving as the user interface, a Zonos API service for voice recognition and synthesis running on port 8001, and a Memo API service for face animation processing on port 8002. This modular architecture allows each component to operate independently while maintaining clear communication channels through RESTful APIs.

A critical implementation decision involved forking both the Zonos and Memo projects to adapt their system APIs for the specific requirements of the human-presentator use case. The original Zonos and memo implementation was modified to better integrate with the distributed architecture. The Memo fork in particular was adapted to try optimize face animation processing but without real achievements.

The containerization strategy employed Docker with NVIDIA GPU runtime support to leverage hardware acceleration for the computationally intensive AI operations. The memo-api service specifically utilizes NVIDIA CUDA capabilities for face animation processing, while the zonos-api service can optionally leverage GPU acceleration when sufficient VRAM is available. As for now it is disabled because it then would take up to 32GB VRAM.

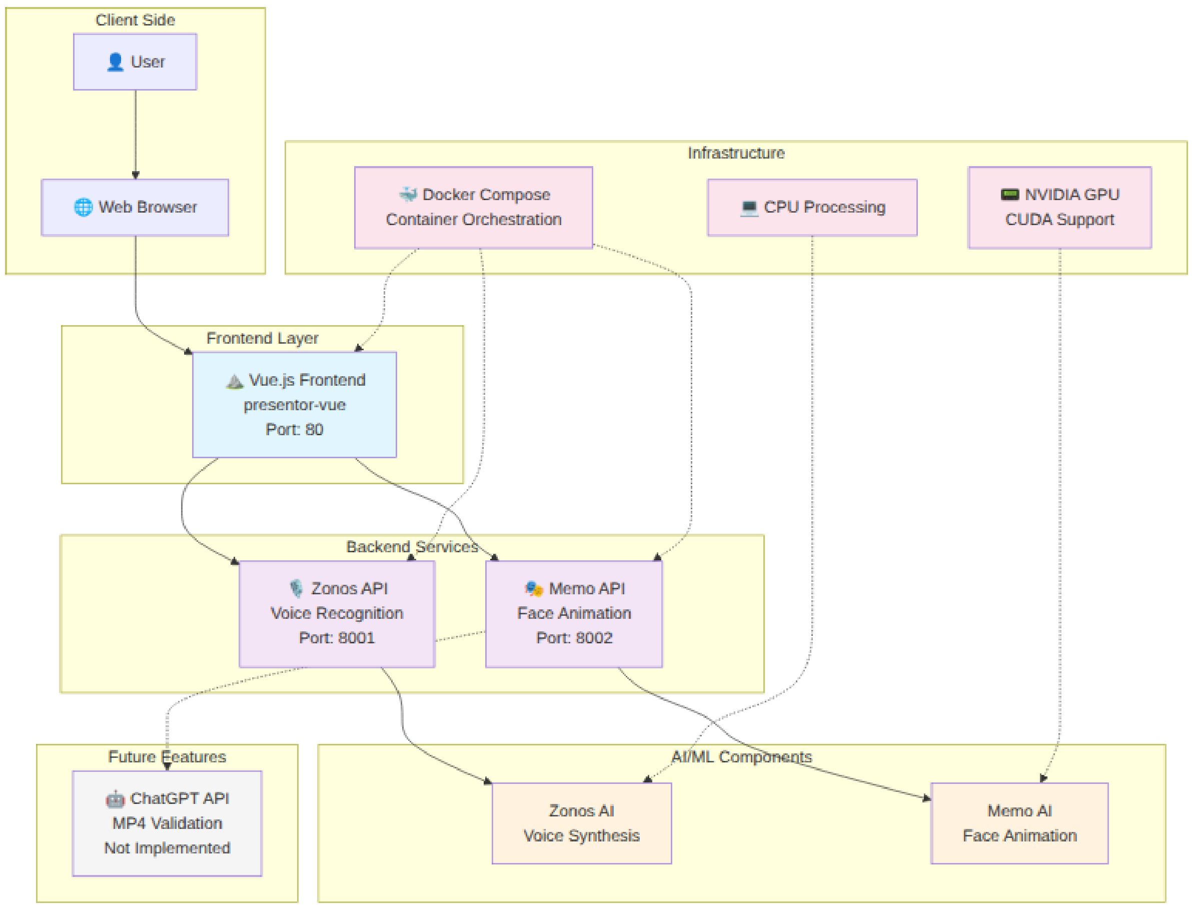


Figure 2: Architecture of the Human Presentor

4 Discussion

The Human Presenter Generator represents a sophisticated pipeline for creating realistic talking-head animations, combining Zonos for voice synthesis and memo for facial animation. While technically impressive, the system's practical usability presents both opportunities and significant challenges for end users.

From a usability perspective, the application provides an intuitive two-step workflow through its Vue.js-based web interface. Users begin by either recording voice samples directly through their microphone or uploading existing audio files to the Zonos component. The interface guides users through entering the desired text and language parameters before generating synthetic speech. Subsequently, the memo component allows users to capture face images via webcam or upload static photographs, which are then animated to match the generated audio. This streamlined approach makes the complex underlying technology accessible to non-technical users through clear visual feedback and straightforward controls.

However, the system's hardware requirements pose substantial barriers to widespread adoption. The implementation demands significant computational resources, particularly GPU memory, with the documentation indicating that even systems with 20GB of VRAM proved insufficient for running both Zonos and memo simultaneously. The Docker configuration reveals that while the memo component requires NVIDIA GPU acceleration for acceptable performance, the Zonos service had to be configured to run without GPU acceleration due to memory constraints. This hardware limitation forces users to either invest in high-end graphics hardware or accept significantly degraded performance, making the system impractical for typical consumer hardware configurations.

The application's architecture, built around separate FastAPI microservices for each AI component and a Vue.js frontend, demonstrates good separation of concerns but introduces additional complexity for deployment and maintenance. Users must manage multiple Docker containers and ensure proper GPU driver integration through the NVIDIA Container Toolkit. The reliance on local processing, while ensuring privacy and offline functionality, places the entire computational burden on the user's system rather than distributing it across cloud infrastructure.

Despite these hardware challenges, the system offers valuable advantages for users with appropriate computational resources. The fully local processing ensures complete data privacy, as no voice samples or images are transmitted to external services. The modular architecture allows for customization and extension of individual components, making it suitable for research applications or specialized use cases. The continuous workflow from voice synthesis to facial animation within a single application eliminates the need for users to coordinate multiple separate tools and manual file transfers between processing stages.