**Mid – Term : Audio-Audio GenAI Usecase - Voice-Cloning — Technical Documentation**

**Table of contents**

1. Project overview
2. What this repo contains (file / folder map)
3. How it works (workflow) — step-by-step
4. Agents: what they are here (implementation details)
5. How to run (development environment & commands)
6. Key implementation notes & snippets (transcription, Gemini polishing, tuning UI)
7. Troubleshooting (common errors we encountered and fixes)
8. Appendix: quick reference commands & tips

**1 — Project overview**

This project implements a local voice-cloning pipeline using the Chatterbox TTS system plus custom preprocessing and transcript processing. The UI is Streamlit. You upload a raw WAV, provide a YouTube link (or video id), and the pipeline:

* Cleans/denoises the uploaded audio (Agent A1).
* Downloads and (optionally) cleans/transforms the transcript (Agent A2).
* Feeds the cleaned audio + script to Chatterbox to generate cloned speech (Agent A3).

**2 — File / folder structure (current snapshot)**

chatterbox/

├─ agents/

│ ├─ agent\_a1\_audio\_cleaner.py # A1: denoiser / normalizer

│ ├─ agent\_a2\_script\_processor.py # A2: orchestrates transcriber → filler remover → polisher (can call Gemini)

│ ├─ agent\_a3\_voice\_cloner.py # A3: wraps ChatterboxTTS usage

│ └─ orchestrator\_run.py # optional orchestrator script (not required)

│

├─ utils/

│ ├─ audio\_tools.py # helpers: load/save/denoise, pitch/tempo functions

│ ├─ filler\_remover.py # simple regex-based filler removal

│ ├─ polisher.py # Gemini-polisher wrapper (optional)

│ ├─ transcriber.py # YouTube transcript helper (extract id + fetch)

│ └─ utils.py # misc pipeline helpers (optional)

│

├─ src/ # chatterbox package sources (Chatterbox TTS)

│ └─ chatterbox/...

│

├─ testing/

│ ├─ app\_agentic.py # lightweight agentic demo (sequential, no LangChain)

│ └─ test.py # import-check helper script

│

├─ app.py # main Streamlit app (the UI you used)

├─ voice\_clone.py # standalone script you used for testing Chatterbox TTS

├─ pyproject.toml

├─ requirements.txt # (keep a stable copy; see instructions)

└─ env/ # your virtualenv (kept outside to avoid breaking)

Note: your virtual environment env/ is fine being outside chatterbox/. You do **not** need to copy env into the repo.

**3 — How it works (workflow)**

High-level steps (what the UI does when you click **Generate**):

1. **Upload audio** (WAV) — stored as ./<uploaded-filename> temporarily.
2. **Agent A1 (Audio Cleaner)**
   * Denoise + normalize using librosa / noisereduce (or a denoising method in audio\_tools.py).
   * Save cleaned output (cleaned\_audio.wav or cleaned\_audio\_agentic.wav).
3. **Fine-tune audio (optional, before final generation)**
   * UI exposes preset styles and manual sliders (pitch, speed, gain, bass, treble, reverb).
   * Tuned output saved (e.g., tuned\_audio.wav) and previewed.
4. **Agent A2 (Transcriber/Processor)**
   * Use utils/transcriber.py to extract the transcript from YouTube (either full URL or video id — your final code accepts full URL).
   * Remove fillers: utils/filler\_remover.py.
   * Optionally polish with Gemini (google.generativeai or langchain\_google\_genai) — **optional** because it may fail due to missing package/import issues or network/timeouts.
5. **Agent A3 (Voice Cloner)**
   * Feed cleaned/tuned audio and polished transcript into ChatterboxTTS.from\_pretrained(...) and generate(...).
   * Save final cloned audio (e.g., cloned\_final.wav or final\_cloned\_agentic.wav).
6. **UI shows**: cleaned audio preview, transcript (raw + polished), and cloned audio preview.

**4 — Agents (what they are here)**

* AgentA1AudioCleaner (agents/agent\_a1\_audio\_cleaner.py)
  + Methods: process\_audio(input\_path, output\_path)
  + Behavior: librosa.load → noisereduce.reduce\_noise → sf.write(output)
  + Notes: requires noisereduce (pip) and librosa, soundfile.
* AgentA2ScriptProcessor (agents/agent\_a2\_script\_processor.py)
  + Methods: process\_youtube(youtube\_url\_or\_id)
  + Behavior: calls utils/transcriber.get\_transcript(...) → utils/filler\_remover.remove\_fillers(...) → utils/polisher.polish\_text(...) (polisher can call Gemini).
  + We recommend leaving Gemini call optional, since polish can also be simple local cleanup.
* AgentA3VoiceCloner (agents/agent\_a3\_voice\_cloner.py)
  + Methods: clone\_voice(cleaned\_audio, text, output\_path)
  + Behavior: ChatterboxTTS.from\_pretrained(device) and tts.generate(text=..., audio\_prompt\_path=...) and save with torchaudio.save or ta.save.
  + Notes: heavy GPU memory usage. Optionally allow device="cpu" to avoid CUDA OOM.

**5 — How to run (safe, repeatable steps)**

**Important**: Do not reinstall or modify your working environment unless necessary. If you do install packages, prefer a new copy of the project (e.g., chatterbox\_test/) to avoid breaking the existing working setup.

**1) Activate your existing env (the one you already used successfully)**

Windows PowerShell example (you used env):

PS> cd C:\Users\ayaan\OneDrive\Desktop\tts-testing

PS> .\env\Scripts\Activate.ps1 # or: source env/bin/activate on Linux/macOS

**2) (Optional) Check imports quickly**

We used a testing/test.py that checks key imports. Running it helps verify the environment:

(env) PS> python testing/test.py

**3) Run the Streamlit main app (your working app)**

From the chatterbox folder:

(env) PS> streamlit run app.py

# or, for agent demo:

(env) PS> streamlit run testing/app\_agentic.py

**4) What to upload / fields to fill in**

* Upload the WAV file (e.g., sample-2.wav).
* Paste the **full YouTube URL** (recommended) or the **video id** (11 chars). Your final transcriber supports full URL and extracts ID.
* (Optional) Gemini API key if you want to use cloud polishing.

**6 — Key implementation notes & code highlights**

**Transcriber (utils/transcriber.py)**

Final working pattern: accept full URL, extract id, use an instance YouTubeTranscriptApi() then fetch(video\_id).

from youtube\_transcript\_api import YouTubeTranscriptApi

from youtube\_transcript\_api.formatters import TextFormatter

import re

def extract\_video\_id(youtube\_url: str) -> str:

pattern = r"(?:v=|\/)([0-9A-Za-z\_-]{11}).\*"

match = re.search(pattern, youtube\_url)

if not match:

raise ValueError("Invalid YouTube URL")

return match.group(1)

def get\_transcript(youtube\_url: str) -> str:

video\_id = extract\_video\_id(youtube\_url)

try:

ytt\_api = YouTubeTranscriptApi()

transcript\_list = ytt\_api.fetch(video\_id)

formatter = TextFormatter()

plain\_text = formatter.format\_transcript(transcript\_list)

return plain\_text

except Exception as e:

raise RuntimeError(f"Transcript error: {e}")

Use full URL in UI; the helper extracts the id internally.

**Gemini polishing**

You used a snippet with google.genai or langchain\_google\_genai. Both work in different environments. Example using google.generativeai style (if installed):

from google import genai

client = genai.Client(api\_key=GEMINI\_API\_KEY)

response = client.models.generate\_content(

model="gemini-2.5-flash",

contents=f"polish this raw transcript ... : {raw\_text}"

)

polished = response.text

**Caveats**:

* Several import variants were attempted in the project (google.generativeai, langchain\_google\_genai, google.genai). Make sure the package you install matches the import you use.
* Network/timeouts and API errors can happen — handle exceptions gracefully and fall back to local polishing.

**Audio tuning (UI placement)**

You implemented tuning controls *before* clicking Generate, stored tuning parameters in st.session\_state["tune\_params"], and applied tuning after cleaning. This avoids page refresh / duplicate-element-id issues.

Tuning examples used librosa.effects.pitch\_shift and librosa.effects.time\_stretch and plain gain multiply. Keep a small safety limit on pitch and length changes to avoid artifacts.

**Agentic demo (testing/app\_agentic.py)**

* This is a minimal sequential orchestrator (A1 → A2 → A3) that **does not use LangChain**. It runs the agents one after another and reports progress.
* Use this to demonstrate agent-like organization without changing the working app.py.

**7 — Troubleshooting — common errors & fixes**

This is an actionable log of the errors we ran into and what to do.

**ModuleNotFoundError: No module named 'torchaudio'**

* Ensure the environment where Streamlit runs has torchaudio installed.
* If you installed torchaudio with a CUDA variant, make sure the runtime Python matches that environment.

**StreamlitDuplicateElementId / UI refresh when switching presets**

* Use key= for widgets or store tuning parameters in st.session\_state.
* Place tuning widgets outside the part of code that re-runs repeatedly or before the main Generate button.

**ModuleNotFoundError: No module named 'utils' or No module named 'agents'**

* Ensure you run Streamlit from repo root (chatterbox/) so relative imports like from agents... and from utils... resolve.
* Good pattern: cd C:\... \chatterbox then streamlit run app.py.

**YouTubeTranscriptApi attribute errors (e.g., .get\_transcript, .list\_transcripts, .fetch)**

* Different versions of youtube-transcript-api expose different APIs. The robust approach:
  + Extract video id (via regex).
  + Use YouTubeTranscriptApi() instance and fetch(video\_id) where available.
  + If not available, fallback to YouTubeTranscriptApi.get\_transcript(video\_id) (older code) or check package docs.
* We settled on ytt\_api = YouTubeTranscriptApi(); ytt\_api.fetch(video\_id) which worked in your environment.

**google / genai import errors for Gemini**

* There are multiple client packages/versions: confirm which you installed.
  + pip install google-generativeai (package google.generativeai) — then import google.generativeai as genai.
  + Alternatively pip install langchain-google-genai (if using LangChain wrapper).
* If you see cannot import name 'genai' from 'google', adjust the import to the correct package you installed.

**CUDA OOM / device-side assert / invalid device memory**

* Chatterbox TTS models are heavy for small GPUs (e.g., GTX 1650 with 4GB). Fixes:
  + Load the model on CPU: change device argument to "cpu" in AgentA3VoiceCloner and ChatterboxTTS.from\_pretrained("cpu"). It will be slower but avoids OOM.
  + Or move to a larger GPU (if available).
  + Free GPU memory: torch.cuda.empty\_cache() before large loads; set env PYTORCH\_CUDA\_ALLOC\_CONF=expandable\_segments:True.
  + Reduce sequence/text length: pass safe\_text = transcript[:800] to avoid very long prompts.
  + If device-side assert persists, set CUDA\_LAUNCH\_BLOCKING=1 for debugging.

**langchain import / API incompatibilities (if you try LangChain)**

* LangChain v1 changed many imports (hub, create\_agent, agent interfaces). If you try to use LangChain, install the matching version and langchain-classic / langchain\_community packages as appropriate, but be careful — these may introduce new dependencies and conflict with the Chatterbox environment.
* Because we had many version mismatches and runtime errors, the safe choice was to **not** force LangChain for the working pipeline. You can experiment in a separate copy of the repo.

**8 — Appendix — quick commands & tips**

**Activate environment**

cd C:\Users\ayaan\OneDrive\Desktop\tts-testing

.\env\Scripts\Activate.ps1

**Run main Streamlit app (working one)**

cd chatterbox

streamlit run app.py

**Run agentic (sequential) demo (safe, no LangChain)**

streamlit run testing/app\_agentic.py

**Test imports (testing/test.py)**

* Use a test.py that tries to import all key modules and prints a report — run it to confirm Python path and installed packages.

**If GPU OOM:**

* Load Chatterbox on CPU for testing:

device = "cpu"

tts = ChatterboxTTS.from\_pretrained(device)

* Or before running, set:

set PYTORCH\_CUDA\_ALLOC\_CONF=expandable\_segments:True

# or in PowerShell:

$env:PYTORCH\_CUDA\_ALLOC\_CONF="expandable\_segments:True"

**Keep a backup**

**Example of testing/app\_agentic.py (what to expect)**

You already have this file. It runs agents sequentially and reports progress. Use it if you want an agent-like demo without external dependencies.