🎙️ The Offline AI VTuber Handbook

Chapter 1 — Setting Up Your Environment (No Experience Needed)

Welcome to your first step toward building your own fully offline AI VTuber[2]. You don't need to know how to code[3]. You don't even need to understand AI yet. We'll take this *one click at a time* — everything on your PC, no cloud, no paid tools[4].

Your setup — **RTX 5070, i5 12400F, 32 GB RAM** — is perfect for running every tool locally[5]. Let's prepare your system so it's stable, clean, and ready for AI work[6].

🧰 Step 1 — Make a Project Folder

You'll keep all your files in one place so nothing gets lost[7].

1. Go to your **Documents** folder.
2. Right-click → **New** → **Folder** → name it:

AI_VTuber_Project

3.
4. Inside that, make subfolders:

/models
/voice
/avatar
/scripts
/assets
/outputs

5.

This keeps everything organized when you start connecting systems later[8].

⚙️ Step 2 — Update Your GPU Drivers

A modern NVIDIA driver unlocks CUDA acceleration — that's what lets your GPU run AI models fast[9].

- Go to the official NVIDIA site:
  👉 https://www.nvidia.com/Download/index.aspx
- Select your GPU model (RTX 5070)[10].
- Download and install the latest **Game Ready Driver**[11].
- When asked, choose **Clean Installation** to reset old settings[12].
- Restart your PC afterward[13].

🧩 Step 3 — Install Python (the Language AI Tools Speak)

Most AI software runs on Python behind the scenes.

1. Visit the official download page:
   👉 https://www.python.org/downloads/
2. Download **Python 3.11 (or newer)** for Windows[14].
3. When installing:
   - ✅ Check **"Add Python to PATH."**
   - Choose **Customize installation → Install for all users**[15].

To confirm it's working:

Press Windows + R, type cmd, hit Enter, then type:

python --version

You should see something like Python 3.11.6[16].

🧱 Step 4 — Install Git (for Downloading AI Tools)

Git is the tool used to grab AI projects from GitHub[17].

1. Download from the official site:
   👉 https://git-scm.com/downloads
2. Install with default settings — just keep clicking **Next**[18].
3. To verify, open **Command Prompt** again and type:

git --version

4.

5. You'll see something like git version 2.46.0[19].

💻 Step 5 — Install Visual Studio Code (for Editing Files)

Think of VS Code as your control center — where you'll open small scripts later[20].

1. Download here:
   👉 https://code.visualstudio.com/
2. Install with defaults[21].
3. Open it once to be sure it launches.
4. In the Extensions tab (left toolbar), install:
   - **Python**
   - **Code Runner**
   - **Markdown Preview Enhanced** [22]

You don't have to understand them yet — they'll just make things easier later[23].

⚡ Step 6 — Get CUDA & cuDNN (GPU Brains)

CUDA lets Python use your GPU; cuDNN speeds it up[24].

1. Go to NVIDIA CUDA Toolkit:
   👉 https://developer.nvidia.com/cuda-downloads
   - Choose **Windows → x86_64 → 11.8 or 12.x**
2. Install with defaults[25].
3. Then grab cuDNN (you'll need a free NVIDIA Developer account):
   👉 https://developer.nvidia.com/cudnn
   - Download for the same CUDA version you installed[26].
4. Extract the files and copy them into your CUDA installation folder (the installer guide shows where)27.
   You only need to do this once28.

🧱 Step 7 — Create a Python Virtual Environment

A virtual environment isolates your AI tools so they don't conflict.

1. Open **Command Prompt** (or VS Code Terminal).
2. Navigate to your project folder:

```
cd %USERPROFILE%\Documents\AI_VTuber_Project
```

3.

4. Create the environment:

```
python -m venv .venv
```

5.
6. Activate it:

```
.venv\Scripts\activate
```

7.
8. (You'll know it worked when you see (.venv) at the start of your prompt.) [29]

🧩 Step 8 — Install Core Python Packages

These are the universal ones every AI project uses.

While your environment is active:

```
pip install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu121
pip install numpy pillow requests tqdm
pip install fastapi uvicorn
```

If you see "Successfully installed …" — you're ready[30].

🧰 Step 9 — Install FFmpeg (for Audio Handling)

FFmpeg handles voice conversion, sound trimming, and lip-sync later[31].

1. Download from the official site:
   👉 https://www.gyan.dev/ffmpeg/builds/
2. Choose **Full build → Windows ZIP**[32].
3. Extract it to C:\ffmpeg
4. Add it to your PATH:
   - Open *Windows Search → Edit the system environment variables → Environment Variables*[33].
5. Under **Path**, click **New**, paste:

```
C:\ffmpeg\bin
```

6.
   - Click **OK** all the way out[34].
7. Verify with:

```
ffmpeg -version
```

35

🧠 Step 10 — Quick Health Check

Let's test that Python, CUDA, and FFmpeg all respond correctly.

Run this in your terminal:

```
Python
import torch
print(torch.cuda.is_available())
```

[36]If it prints **True**, congratulations — your GPU is connected and ready for AI[37].

✅ You've Finished Chapter 1

You now have a rock-solid foundation: Python, Git, VS Code, CUDA, and FFmpeg all installed and working[38]. Everything else you build — from the voice to the avatar — will depend on this setup[39].

🧭 Coming Up Next:

Chapter 2 — Gathering Your Free Tools and Installing the AI Toolkit

We'll download and set up Ollama, Unsloth, VRoid Studio, Live2D Cubism, VSeeFace, OBS, Audacity, and connect them neatly40.

Chapter 2 — Installing All the Free Tools (Your AI Toolkit)

You now have the foundation ready — Python, Git, CUDA, and FFmpeg are working[41]. Now we'll gather and install every major tool that makes your AI VTuber possible[42]. Everything in this chapter is **free**, **offline**, and compatible with your RTX 5070 and Windows 11 setup[43].

🧠 Step 1 — Install Ollama (Your Local AI Brain)

Ollama is the easiest way to run large language models (LLMs) completely offline44. It's like ChatGPT that lives on your computer, using your GPU instead of the cloud45.

Download:

👉 https://ollama.com/download 46

Install steps:

1. Download the **Windows installer**.
2. Run it and follow the prompts[47].
3. After installation, open **Command Prompt** and type:

ollama run llama3

4.
5. This command downloads and runs the Llama 3 model (around 4–8 GB)[48].
6. When it says >>>, type a question like:

Hello! How are you?

[49]If you get a reply, congratulations — your first offline AI model is working[50].

Why Ollama?

It handles all the complex setup automatically (CUDA, weights, quantization)51. It can also host fine-tuned models locally (which you'll create later with Unsloth)52.

⚙️ Step 2 — Add More Models to Ollama

You can swap or test models easily:

Model	**Command**	**Strength**
Llama 3 8B	ollama run llama3 [56]	**Balanced, fast** [57]
Mistral 7B	ollama run mistral [58]	Creative, lightweight [59]
Phi-3 Mini	ollama run phi3 [60]	**Tiny & fast** [61]

Neural Chat 7B	ollama run neural-chat [62]	Conversational tone [63]

Models are stored under:

C:\Users\<YourName>\.ollama\models

Each can be customized later — like swapping the “brain” of your VTuber.

🎨 Step 3 — Install VRoid Studio (3D Avatar Creator)

VRoid Studio is a free, anime-style 3D character maker64. You can customize hair, clothes, eyes, body, and export a model ready for animation65.

Download:

👉 https://vroid.com/en/studio 66

Steps:

1. Install and open the program.
2. Click **“New Character”** → choose male or female[67].
3. Customize face, hair, clothing[68].
4. Go to **Export** → **Export as VRM** and save it in your project folder:

AI_VTuber_Project/avatar/your_model.vrm
5.

That .vrm file will be used later in VSeeFace or Unity[69].

🖼️ Step 4 — (Optional) Install Live2D Cubism (2D Avatar Path)

If you prefer 2D avatars (like Hololive-style streamers), use Live2D Cubism70.

Download:

👉 https://www.live2d.com/en/download/cubism/ 71

Use-case:

- You draw your character in layers (PSD).
- Cubism rigs it into an animated 2D model (.moc3)72.
  You can later use VTube Studio to animate it live73.

🧍 Step 5 — Install VSeeFace (Avatar Animator)

VSeeFace makes your 3D model blink, talk, and move in real time using your webcam or tracking software74.

Download:

👉 https://www.vseeface.icu/ 75

Setup:

1. Download and extract the ZIP.
2. Open VSeeFace.exe[76].
3. Load your .vrm avatar → check mouth movement and expressions[77].
4. Enable lip-sync from microphone → test speaking78.
   You'll later replace your microphone input with your AI voice output so the avatar "talks" automatically79.

🎥 Step 6 — Install OBS Studio (for Streaming & Recording)

OBS captures your VSeeFace window and sends it to Twitch, YouTube, or wherever you stream80.

Download:

👉 https://obsproject.com/download 81

Setup:

1. Install OBS → launch it.
2. Click "+ → Window Capture" → select VSeeFace[82].
3. Add "Audio Input Capture" → choose your system audio or virtual cable[83].
4. Click Start Virtual Camera84.
   Now any app (Zoom, Discord, etc.) can see your VTuber as a virtual webcam85.

🔌 Step 7 — Install OBS WebSocket Plugin

This lets scripts control OBS automatically — for example, to switch scenes or trigger animations when the AI speaks86.

Download:

👉 https://github.com/obsproject/obs-websocket/releases 87

Install:

1. Run the installer.
2. In OBS: Tools → WebSocket Server Settings → check “Enable WebSocket Server”[88].
3. Note the port (usually 4455) and password (you’ll use it later)[89].

🎤 Step 8 — Install Audacity (for Voice Editing)

You’ll use this to trim samples or tweak your AI voice90.

Download:

👉 https://www.audacityteam.org/download/ 91

It’s simple: drag audio in, cut, fade, export as WAV or MP3[92].

🧬 Step 9 — Install RVC (Real-Time Voice Conversion)

RVC lets you turn one voice into another — for example, converting your AI TTS voice into your VTuber’s custom voice93.

Download:

👉 https://github.com/RVC-Project/Retrieval-based-Voice-Conversion-WebUI 94

Quick start:

1. Click **Code** → **Download ZIP**.
2. Extract to AI_VTuber_Project/voice/RVC.
3. Run go-web.bat[95].
4. Open http://127.0.0.1:7865/ in your browser[96].
5. Train a voice model with a few minutes of clean audio (WAV files)97.
   Later, your AI will generate speech → RVC will convert it to your character’s voice98.

🧠 Step 10 — Install Unsloth (For Local LLM Fine-Tuning)

Unsloth makes fine-tuning LLMs simple, even for beginners[99]. It's how you'll give your AI its unique personality[100].

Guide: https://docs.unsloth.ai/ [101]

Install Steps:

1. Activate your virtual environment (.venv\Scripts\activate).
2. Run:

pip install unsloth

3.

102

3. Once installed, you can use their tutorial datasets to train your model's personality (like "friendly streamer," "sarcastic robot," etc.)103.

We'll cover fine-tuning in Chapter 8104.

🔊 Step 11 — Install XTTS v2 (Offline Text-to-Speech)

XTTS is a free offline TTS engine that runs well on your GPU105. It can generate natural voices with good intonation and emotion106.

Repo: https://github.com/coqui-ai/TTS 107

Install:

pip install TTS

108

To test:

Python
```
from TTS.api import TTS
tts = TTS(model_name="tts_models/multilingual/multi-dataset/xtts_v2")
tts.tts_to_file(text="Hello! I'm your AI VTuber voice.", file_path="test.wav")
```

Play test.wav — if you hear a voice, it works! [109]

🎯 Step 12 — Test Everything Together

Let's do a mini-integration test to prove all your tools are working110.

Open VS Code → new file → test_pipeline.py:

```
Python
from TTS.api import TTS
import os

os.system("start cmd /k ollama run phi3")

tts = TTS(model_name="tts_models/multilingual/multi-dataset/xtts_v2")
text = "This is a quick test of your offline AI system."
tts.tts_to_file(text=text, file_path="voice_test.wav")

111
```

Save and run:

```
python test_pipeline.py
```

You'll hear your first AI-generated voice and see your LLM respond in the background[112].

You've officially assembled the offline AI toolkit! 113

✅ End of Chapter 2

You now have every essential tool installed and tested locally[114]. Your PC is now a self-contained AI studio[115].

🧭 Coming Up Next:

Chapter 3 — Building Your AI Brain (Teaching Your VTuber How to Think)

We'll create a personality prompt, load models into Ollama, and make the AI talk like a character[116].

🧠 Chapter 3 — Building Your AI Brain (Personality & Local LLM Setup)

By now, you have the *body* (avatar tools) and *voice* (TTS)[117]. Now it's time to build the **mind** — the part that decides how your AI speaks, reacts, and stays consistent[118]. Everything here is **offline**, using **Ollama**, **Llama 3**, and optionally **Unsloth** for fine-tuning later[119].

🪄 Step 1 — Understanding the "Brain Loop"

Your VTuber's brain works like this:

User message → AI thinks (LLM) → Text output → TTS converts to speech → Avatar speaks

You already have the tools for each step[120]. Now, we'll configure the "AI thinks" part — the **LLM (Large Language Model)** that drives everything[121].

⚙️ Step 2 — Open Ollama and Pick a Model

Ollama makes it dead simple to swap models like changing game characters122.

Open Command Prompt and type:

ollama list

123

You'll see something like:

NAME	SIZE
llama3	4.7GB
mistral	4.2GB
phi3	2.1GB

These are your AI "brains." [124] Each one has a personality style:

- **Llama 3** → Balanced, good general tone.
- **Mistral** → Slightly creative and fun.
- **Phi-3** → Lightweight and snappy for fast responses[125].

For VTubing, Mistral is a great start (friendly and expressive).

Run this:

ollama run mistral

126

You'll get a chat prompt (>>>). Try typing:

You are now an energetic VTuber named Nova.
Greet your viewers!

[127] If it answers in character — your brain works! [128]

🧩 Step 3 — Create a Custom Personality Prompt

A “prompt” is just text that tells your AI *who* it is and *how* to behave[129]. We’ll make one you can easily reuse.

In your project folder, create:

AI_VTuber_Project/brain/personality.txt

[130]

Open it in VS Code and paste this:

SYSTEM:
You are an AI VTuber.
Your name is {{vtuber_name}}. You are friendly, expressive, and funny but never rude or inappropriate[cite: 105].
You stream games, chat with viewers, and sometimes joke or tease playfully[cite: 106].
You always keep your tone casual and upbeat.
If you don’t understand something, make a lighthearted joke instead of giving a boring answer[cite: 107].
Stay in character no matter what.

USER: {{user_input}}

[131]

Replace {{vtuber_name}} with your chosen character name.

Example:

You are an AI VTuber named Nova. You are upbeat, a little chaotic, and love gaming references.

[132]

Save it.

🧠 Step 4 — Test the Personality File in Ollama

You can feed Ollama your custom prompt[133].

In the terminal:

ollama run mistral -f "brain/personality.txt"

134It’ll start a chat using your prompt as its personality core135.

Now type something like:

How do you feel today, Nova?

136If it responds playfully or matches your tone — you’ve just defined your AI’s character!
137

🧪 Step 5 — Add Memory (Simple Offline Context)

You can give your AI short-term memory so it remembers what was said in the same session138.

Make a file:

AI_VTuber_Project/scripts/memory_chat.py

139

Paste this simple script:

Python
```
import subprocess

memory = []

def chat_with_ollama(prompt):
    global memory
    memory.append(f"User: {prompt}")
    context = "\n".join(memory[-6:])  # last 6 lines memory
    result = subprocess.run(
        ["ollama", "run", "mistral"],
        input=context.encode(),
        capture_output=True,
        text=True
    )
    response = result.stdout.strip()
    memory.append(f"AI: {response}")
    print(f"\nAI: {response}\n")

while True:
    msg = input("You: ")
```

```
    if msg.lower() in ["quit", "exit"]:
        break
    chat_with_ollama(msg)
```

140

Run it:

```
python scripts/memory_chat.py
```

You now have a mini local chatbot with short-term memory — and it's 100% offline[141].

🎨 Step 6 — Teach Tone & Style (Fine-Tuning Prep)

This is where you *teach* your AI how to sound more natural — like a human streamer[142].

Create your dataset folder:

AI_VTuber_Project/brain/training_data/

143

Inside it, make a few .txt files, e.g.:

```
stream_chatter.txt
funny_replies.txt
calm_mode.txt
```

Each one should have sample dialogues144.

Example:

```
User: Hey Nova, what's your favorite game?
AI: Definitely Elden Ring! I love suffering in HD[cite: 118].
---
User: What's your opinion on Mondays?
AI: Mondays are like lag spikes in real life. Nobody asked for them[cite: 119].
```

The more examples, the better your AI will match your tone later when fine-tuned[145].

⚙️ Step 7 — Fine-Tuning with Unsloth (Local and Easy)

Fine-tuning helps your model learn your VTuber's personality permanently — no prompts needed146.

Make sure your environment is active:

.venv\Scripts\activate

147

Then, install Unsloth if you haven't:

pip install unsloth

148

Next, create a config file:

AI_VTuber_Project/brain/unsloth_config.yaml

149

Paste:

YAML
model: mistral-7b-instruct
dataset: ./brain/training_data
epochs: 2
learning_rate: 5e-5
output_dir: ./brain/fine_tuned
quantization: 4bit

Then run:

unsloth train --config ./brain/unsloth_config.yaml

It'll use your GPU to train a LoRA personality file (in a few hours, depending on dataset size)150.

When done, load it into Ollama:

ollama create nova-mistral -f ./brain/fine_tuned/adapter.safetensors
ollama run nova-mistral

Now your AI has its own memory, tone, and humor — it's *truly* your VTuber brain[151].

🗣️ Step 8 — Connect Brain to Voice (Simple Pipeline)

Let's do a small test — have your AI speak with its new voice152.

Make a Python file:

scripts/brain_to_voice.py

153

Paste:

```
Python
from TTS.api import TTS
import subprocess

tts = TTS(model_name="tts_models/multilingual/multi-dataset/xtts_v2")

while True:
    user_input = input("You: ")
    if user_input.lower() == "exit":
        break
    ai_output = subprocess.run(
        ["ollama", "run", "nova-mistral"],
        input=user_input.encode(),
        capture_output=True,
        text=True
    ).stdout.strip()
    print(f"AI: {ai_output}")
    tts.tts_to_file(text=ai_output, file_path="voice/ai_line.wav")
```

154

Run it:

```
python scripts/brain_to_voice.py
```

Now your AI will generate text → convert it into a voice clip[155].

Next chapter, you'll sync that voice with your avatar[156].

✅ Chapter Summary

You've now created:

- A **local conversational AI** using Ollama[157].
- A **personality prompt** that defines your VTuber's behavior[158].
- A **simple memory system**[159].
- A **fine-tuning pipeline** for permanent character traits[160].

- A text-to-voice bridge that turns replies into speech161.
  Your VTuber now thinks and talks162.
  Next, we'll make it look alive163.

🧭 Coming Up Next:

Chapter 4 — Giving Your AI a Voice (TTS, RVC, and Lip Sync Integration)

We'll make your AI's words flow out of your avatar's mouth — in real time, with emotion164.

🎤 Chapter 4 — Giving Your AI a Voice (Offline TTS, Voice Conversion, and Lip Sync)

Up until now, your AI has been text-only165. It can think, remember, and joke — but all inside the console166. This chapter brings it to life by adding a voice and teaching it how to move its mouth when speaking167.

You'll use three local tools:

Purpose	**Tool**	**Why It's Great**
Text-to-Speech	**XTTS v2 (Coqui TTS)** [171]	**Natural, multilingual, fast** [172]
Voice Customization	**RVC (Retrieval-based Voice Conversion)** [173]	Lets you create your own custom voice [174]
Lip-Sync [175]	**OpenSeeFace + VSeeFace** [176]	**Syncs the voice to your avatar's mouth** [177]

🧠 Step 1 — The Voice Flow Explained

Your AI's speech follows this chain:

Text → (TTS) → Audio file → (RVC) → Transformed voice → (VSeeFace) → Mouth movement

You'll set up each piece now, then link them all together[178].

🔊 Step 2 — Setting Up XTTS v2 (Text-to-Speech Engine)

You already installed Coqui TTS earlier, so let's configure it properly[179].

Test the voice output:

In VS Code, open a new file scripts/tts_test.py:

Python
```
from TTS.api import TTS

tts = TTS(model_name="tts_models/multilingual/multi-dataset/xtts_v2")
tts.tts_to_file(
    text="Hello there, I'm your AI VTuber speaking for the first time!",
    file_path="voice/test_voice.wav"
)
print("✅ Voice generated! Check voice/test_voice.wav")
```

180

Run it:

```
python scripts/tts_test.py
```

You should hear a natural-sounding voice181.

If you do — you've officially given your AI the power of speech182.

🧬 Step 3 — Creating Your Custom Voice (RVC)

Now let's make that voice yours.

RVC lets you transform the generated voice into a unique sound — male, female, robotic, anime-style, anything183.

Step 3.1 — Launch RVC

Go to your RVC folder:

AI_VTuber_Project/voice/RVC

Double-click go-web.bat[184].

When the console opens, it'll show:

Running on http://127.0.0.1:7865/

[185]Open that in your browser[186].

Step 3.2 — Train Your Voice

You'll need a few minutes of clear, clean speech[187]. You can use your own voice or a sample you recorded[188].

1. Upload your sample WAV under "Dataset"[189].
2. Click "Train model" → name it something like nova_voice[190].
3. Wait for training to complete (it may take 10–20 minutes)[191].
4. You'll now have a custom .pth model file saved under voice/RVC/models/[192].

Step 3.3 — Convert AI Audio to Your Voice

When your AI generates voice/ai_line.wav from XTTS, you can run it through RVC automatically[193].

Create a script:

scripts/voice_conversion.py

[194]

Paste:

Python
```
import os

source = "voice/ai_line.wav"
output = "voice/ai_line_converted.wav"
model = "voice/RVC/models/nova_voice.pth"

os.system(f'python voice/RVC/infer-web.py --model {model} --input {source} --output {output}')

print("✅ Voice converted successfully!")
```

Now your AI speech will sound like your chosen custom voice[195].

👄 Step 4 — Lip Sync and Mouth Movement (OpenSeeFace + VSeeFace)

Next, we'll make your avatar's mouth move when the AI speaks196.

You already installed:

- **VSeeFace** (for rendering and control)
- **OpenSeeFace** (for tracking) [197]

Step 4.1 — Start OpenSeeFace

Open the folder:

AI_VTuber_Project/avatar/OpenSeeFace

Run run.bat198.

It will open a webcam window and a console that shows tracking data — head position, eye blink, mouth open/close values199.

This data is what VSeeFace reads200.

Step 4.2 — Set Up Lip Sync Input

1. Open VSeeFace[201].
2. Load your .vrm model (exported from VRoid Studio)[202].
3. Go to **Settings → General → Lip Sync Input**[203].
4. Choose **Audio-based lip sync**[204].
5. Set input to your AI voice output (the WAV your TTS creates)205.
   Tip: You can route your audio using VB-Audio Virtual Cable (free)206.
   👉 https://vb-audio.com/Cable/ 207

Set your AI audio output to "CABLE Input," and VSeeFace to "CABLE Output"[208].

Step 4.3 — Test Your Lip Sync

Play your AI voice test clip (voice/test_voice.wav)209.

If your avatar's mouth moves — perfect! 210

If not:

- Double-check the audio routing in Windows Sound Settings[211].
- Make sure the correct device is chosen in VSeeFace212.
  Now your avatar reacts to audio — it speaks when the AI talks213.

🧩 Step 5 — Sync Everything Together

Here's the full mini-script that connects the brain, voice, and animation:

File: scripts/talk_pipeline.py

Python

```
from TTS.api import TTS
import subprocess, time, os

tts = TTS(model_name="tts_models/multilingual/multi-dataset/xtts_v2")
voice_file = "voice/ai_line.wav"

while True:
    user_input = input("You: ")
    if user_input.lower() == "exit":
        break

    # AI thinks
    ai_output = subprocess.run(
        ["ollama", "run", "nova-mistral"],
        input=user_input.encode(),
        capture_output=True,

     text=True
    ).stdout.strip()

    print(f"AI: {ai_output}")

    # Generate voice
    tts.tts_to_file(text=ai_output, file_path=voice_file)

    # Convert voice (optional)
    os.system(f'python scripts/voice_conversion.py')

    # Play final voice file
    os.system(f'start voice/ai_line_converted.wav')

    time.sleep(1)
```

Run it:

python scripts/talk_pipeline.py

Now your AI:

1. Reads your text input[215].
2. Responds in character[216].
3. Speaks with a custom voice[217].
4. Moves the avatar's mouth as the audio plays218.
   You've just created an AI-powered speaking VTuber — fully offline219.

🎚️ Step 6 — Add Emotions (Optional Power-Up)

You can extend lip sync into emotional expression220.

For example:

- Add *happy/sad/angry* emotion keywords to the AI's output[221].
- When the AI says "(happy)" → trigger a smiling animation in VSeeFace222.
  You can automate this later using OBS WebSocket events or a simple Python command that tells VSeeFace to change expressions when a certain emotion tag appears in text223.

✅ Chapter Summary

You now have:

- Offline text-to-speech (XTTS v2)
- Custom voice (RVC)
- Real-time lip sync (OpenSeeFace + VSeeFace)
- A complete text → voice → animation pipeline

This is the heart of your VTuber[224]. From here, you can stream, record videos, or add personality upgrades[225].

🧭 Coming Up Next:

Chapter 5 — Building Your Avatar (Rigging, Expressions, and Physics)

We'll make your model look natural, move smoothly, and show personality through body language and expressions226.

🎭 Chapter 5 — Building Your Avatar: Rigging, Expressions, and Physics

Quick map of this chapter:

1. Prepare the art/model
2. Basic rigging (3D and 2D)
3. Blendshapes / morph targets (face expressions)
4. Physics (hair, clothing)
5. Idle and action animations
6. Emotes, gestures, and triggers
7. Export, test, and optimize on your RTX 5070
8. Troubleshooting and tuning [227]

1) Prep: clean files and organization (do this first)

Before you start rigging, keep files tidy.

Inside your project folder:

```
AI_VTuber_Project/
  avatar/
    source_art/     # PSDs, PNGs, Blender files
    vrm_models/     # exported .vrm models
    live2d_models/  # .moc3, art, physics files
    exports/        # final builds for VSeeFace/VTube
```

228Why: rigging tools export and reference filenames — if you move stuff later, links break229.

Save early and save often.

2) 3D Avatar Path — VRoid → Unity / VSeeFace (recommended for full-body and gestures)

a) Create in VRoid Studio

- Open VRoid Studio → **New Character**[230].
- Use the GUI to set face, body, clothes. If you're not an artist, use default template and tweak colors[231].

- Hair: use VRoid's hair editor to create hair groups (this helps later for physics)[232].
- Export: File → Export → Export as VRM → save to avatar/vrm_models/your_model.vrm[233].

b) Add extra blendshapes (in Blender) — optional but powerful

If you want custom expressions beyond VRoid defaults:

1. Open Blender (free) and import your .vrm (use VRM_IMPORTER add-on)[234].
2. Switch to sculpt/edit mode for facial shapes (smile, frown, squint)[235].
3. In Blender shape keys, create keyframes for each expression (Smile, Angry, Surprise)[236].
4. Re-export .vrm (or export to Unity later)[237].

Note: Blender is optional. Many creators stick with VRoid's built-in shapes[238].

c) Use VSeeFace for live puppeting

- Open VSeeFace → Load .vrm[239].
- In VSeeFace settings:
  - **Tracking**: enable Face, Eye, and Lip tracking (if using webcam)[240].
  - **Lip Sync**: set to "Audio" for audio-driven lip movement (our pipeline will provide AI audio)[241].
  - Expressions: map model blendshapes to VSeeFace expression slots (Smile → expression_001 etc.)242.
    VSeeFace reads either webcam or external tracking data (OpenSeeFace/Mediapipe) and maps it to your avatar in real time243.

---

3) 2D Avatar Path — Live2D Cubism → VTube Studio (recommended for anime style faces)

a) Prepare artwork (Photoshop or Krita)

- Canvas: 3000×3000 or similar high resolution[244].
- Layer naming is vital: name mouth, eye_left, eye_right, hair_front, hair_back, etc.
- Save a layered PSD in avatar/source_art/[245].

b) Import into Live2D Cubism

- Open Cubism → **New Project** → import PSD[246].
- The tool auto-generates parts based on layer names[247].
- Arrange the art mesh for each part (deformers + parameters)248.
  Key Live2D concepts:
- **Parameters** — numeric controls like MOUTH_OPEN, EYE_BLINK, HEAD_X, MOUTH_FORM[249].
- **Deformers** — used to move parts smoothly (bends, twists)[250].
- **Physics** — Live2D physics engine for hair and clothes[251].

c) Create lip-sync and expressions

- Lip-sync: define mouth sprites or morphs for phonemes (A, I, U, E, O) and map audio visemes to MOUTH_OPEN + MOUTH_FORM[252].
- Blinks: create blink animation parameter mapped to EYE_BLINK[253].
- Expressions: create parameter presets for Smile, Angry, Sad;
  save as motions254.

d) Export to .moc3 and assets

- Export .moc3, texture PNGs, physics settings (physics3.json), and motion files[255].
- Put them into avatar/live2d_models/[256].

e) Use VTube Studio to animate live

- Load your .moc3 in VTube Studio[257].
- Set **Audio > Auto lip-sync** with your AI audio input (via virtual cable)[258].
- VTube Studio also supports face tracking for head tilt via webcam[259].

---

4) Blendshapes / Morph Targets (making expressions feel human)

Blendshapes control face expressions using small geometry changes[260].

For 3D (VRM/Unity):

- Blendshape list commonly includes: Smile, Angry, Surprise, Blink_L, Blink_R, Mouth_A, Mouth_I, Mouth_U, Mouth_E, Mouth_O[261].

- In Unity or Blender, map each viseme/phoneme to corresponding blendshape (this improves lip sync fidelity)[262].

For 2D (Live2D):

- Map viseme sprites or mouth morphs to Live2D parameters[263].
- Use rhythmic offsets so lip movement looks natural (not robotic)[264].

---

5) Physics — hair, skirts, cloth, and interactive props

Good physics adds subtle life[265].

Live2D physics:

- Open Physics settings in Cubism → Create physics objects for hair and accessories[266].
- Configure stiffness, drag, and gravity. Test with head movement to tune bounciness[267].

VRoid/Unity physics:

- In Unity, add simple SpringBone or DynamicBone components to hair bones[268].
- Tweak damping and stiffness:
  - **Stiffness** high → hair moves less (good for long recording sessions)[269].
  - **Damping** controls how fast it stops moving[270].
- Test with avatar idle movement and recorded voice to see how wind/gestures affect hair271.
  Performance tip: avoid dozens of heavy physics bones. Use 3–8 per hair cluster for balanced realism and performance on your RTX 5070272.

---

6) Idle & Action Animations (make your avatar feel alive when not speaking)

Idle animations are short loops: breathing, hair sway, subtle head tilt[273].

- Create 3–6 second idle loops in Live2D/Unity.
- Randomize idle triggers so the avatar doesn't feel robotic (e.g., every 10–30 seconds pick a random idle)[274].

- Action animations: wave, clap, dance — useful for chat triggers or subs/donations275.
  In Unity, use an Animator Controller with Idle, Talk, Gesture states. Transition using parameters (e.g., isSpeaking, gestureID)276.

---

7) Emotes, Gestures, and Chat Triggers

Add personality micro-interactions that respond to viewers277.

Examples:

- Chat command !dance → trigger dance animation[278].
- New follower → small jump animation + confetti overlay[279].
- !angry → Angry blendshape + cartoon steam overlay[280].

Implementation (no coding required):

- VSeeFace + OBS: map simple hotkeys to trigger animations in VSeeFace; use OBS scenes/filters for overlays281.
- For automatic triggers: later we'll wire the coordinator server (FastAPI) to send keyboard/macro events to VSeeFace or OBS when the LLM outputs specific [EMOTION] tags[282].

---

8) Exporting and Testing (the "does it look alive?" checklist)

Before streaming, test the full stack:

1. Load model in VSeeFace or VTube Studio[283].
2. Play a sample audio clip and watch lip-sync and expressions[284].
3. Trigger an emotion tag manually (e.g., send the text: "I'm so happy! [EMOTION:happy]")[285].
4. Verify physics: head turns, hair follows naturally; no clipping through body[286].
5. Check CPU/GPU usage in Task Manager (should be moderate; GPU VRAM within 12 GB for your 5070)287.
   If all good, export the final model to avatar/exports/your_model_ready.vrm or .moc3288.

---

9) Optimization Tips for RTX 5070 (keep it smooth)

Your RTX 5070 is great, but real-time animation + LLM inference + TTS is heavy289.

Use these tips:

- Use **lower-res textures** (2048x2048 instead of 4096x4096) for the avatar if GPU memory gets high[290].
- In Unity, set **Quality** → **Shadows** to Low or Off during streaming[291].
- Limit physics bones;
  fewer bones = less CPU/GPU overhead292.
- In VSeeFace, disable extra camera filters if you're seeing frame drops[293].
- Use model quantization/4-bit for LLM inference (Ollama handles quantization) to save VRAM[294].
- Close other GPU-heavy apps (browsers, games) while streaming[295].

---

10) Troubleshooting (common problems & quick fixes)

Problem: Mouth doesn't sync to audio

- Check that audio is routed through virtual cable into VSeeFace/VTube Studio[296].
- Make sure audio is PCM 16-bit WAV. Convert with FFmpeg if needed:

ffmpeg -i input.mp3 -ar 16000 -ac 1 -sample_fmt s16 output.wav

-

297

Problem: Avatar stutters while AI is generating speech

- LLM or TTS might be using the GPU concurrently;
  reduce TTS model size or use audio caching (generate and then play)298.
- If using real-time TTS, prebuffer short replies (max 6–8 seconds) and limit simultaneous animations299.
  Problem: Physics clips through model
- Reduce bone length or add collider shapes to prevent intersection.
- Increase stiffness slightly300.
  Problem: OBS frame drops
- Lower output resolution (1280×720) and bit rate[301].
- Use OBS hardware encoder (**NVENC**) to offload encoding to GPU[302].

---

11) Practical Mini-Workflow to Test Everything

1. Open VSeeFace and load your .vrm303.
   Set lip-sync to audio input = CABLE Input304.
2. Start OBS and add Window Capture for VSeeFace. Enable Virtual Camera[305].
3. Run scripts/talk_pipeline.py (from Chapter 4). Type a line and watch the avatar speak[306].
4. Trigger an idle animation or smile expression with a hotkey[307].
5. Watch resource usage; tweak quality settings if needed[308].

---

12) Where to go next from here

- Add a set of 8–12 emote animations mapped to chat events[309].
- Build an expression library (happy, sad, embarrassed, flustered) and map them to [EMOTION] tags your LLM can output[310].
- Create a simple UI (in OBS browser source) that shows the current expression and active triggers for debugging[311].

✅ Chapter Summary

You've learned how to:

- Prepare art and keep it organized[312].
- Rig 3D and 2D avatars for live tracking[313].
- Create blendshapes/morphs and map visemes for natural lip-sync[314].
- Add physics to hair and clothing while keeping performance reasonable[315].
- Implement idle animations, gestures, and chat-triggered emotes[316].
- Export, test, and optimize the avatar for real-time streaming on your RTX 5070[317].

⚡ Chapter 6 — Connecting Everything (The Full Bridge System)

You've already built all the parts.

Now we're going to link them together so your VTuber runs like this:

You type or chat → AI (Ollama) responds → Voice (XTTS + RVC) → Avatar moves → OBS streams it live

All **offline**, powered by your GPU, controlled through simple scripts and OBS automation[318].

🧠 Step 1 — Understand the Core Pipeline

Your system has 4 pillars that will talk to each other:

Role	Tool	What it does
AI Brain	**Ollama (Mistral / Llama 3)**[322]	**Thinks, generates responses**[323]
Voice	**XTTS + RVC**[324]	**Converts text → speech → your custom voice**[325]
Avatar	**VSeeFace / VTube Studio**[326]	**Animates and lip-syncs with voice**[327]
Stream	**OBS Studio + WebSocket plugin**[328]	**Captures video/audio, overlays, automates transitions**[329]
We'll use **FastAPI** (a tiny local web server) as the "bridge" that lets them talk[330].		

🧩 Step 2 — Install FastAPI & WebSocket Dependencies

Activate your virtual environment if not already:

.venv\Scripts\activate

Then install:

pip install fastapi uvicorn websockets requests

This allows your AI brain and OBS to communicate through a local web port — no internet required[331].

🧠 Step 3 — Build the Bridge Server

We'll create one central Python file that manages communication between all parts[332].

Create:

AI_VTuber_Project/scripts/bridge_server.py

333

Paste:

Python

```
from fastapi import FastAPI, WebSocket
from TTS.api import TTS
import subprocess, os, asyncio

app = FastAPI()
tts = TTS(model_name="tts_models/multilingual/multi-dataset/xtts_v2")

@app.get("/")
def read_root():
    return {"status": "Online", "message": "AI VTuber Bridge Running"}

@app.websocket("/chat")
async def chat_endpoint(websocket: WebSocket):
    await websocket.accept()
    while True:
        data = await websocket.receive_text()
        print(f"User: {data}")

        # Step 1: AI Brain (Ollama)
        ai_response = subprocess.run(
            ["ollama", "run", "nova-mistral"],
```

```
            input=data.encode(),
             capture_output=True,
             text=True
        ).stdout.strip()

        print(f"AI: {ai_response}")

        # Step 2: Text-to-Speech
        tts.tts_to_file(text=ai_response, file_path="voice/ai_line.wav")

        # Step 3: Voice Conversion (optional)
        os.system(f'python scripts/voice_conversion.py')


         # Step 4: Play final voice file
        os.system("start voice/ai_line_converted.wav")

        await websocket.send_text(ai_response)

if __name__ == "__main__":
    import uvicorn
    uvicorn.run(app, host="127.0.0.1", port=8000)
```

334

Run this:

```
python scripts/bridge_server.py
```

You'll see:

```
INFO:     Uvicorn running on http://127.0.0.1:8000
```

Your local AI control hub is live[335].

🌐 Step 4 — Test the Bridge

You can talk to the bridge directly in your browser[336].

1. Go to: http://127.0.0.1:8000
   You should see:

```
{"status": "Online", "message": "AI VTuber Bridge Running"}
```

2.

337

2. To chat through the WebSocket, open a Python shell and test:

```
Python
import asyncio, websockets

async def test_chat():
    async with websockets.connect("ws://127.0.0.1:8000/chat") as ws:
        await ws.send("Say hello to the viewers!")
        reply = await ws.recv()
        print("AI:", reply)

asyncio.run(test_chat())
```

The bridge should reply — and your VTuber's voice should play[338].

🎙️ Step 5 — Connect the Bridge to OBS

You'll now make OBS listen to your AI's events — so it can switch scenes, trigger effects, or change camera angles when your VTuber speaks339.

Ensure OBS WebSocket plugin is enabled (Tools → WebSocket Server Settings)340.

Install the OBS control library:

```
pip install obs-websocket-py
```

341

Create:

AI_VTuber_Project/scripts/obs_connector.py

342

Paste:

```
Python
import asyncio
from obswebsocket import obsws, requests
import websockets

# Connect to OBS
obs = obsws("localhost", 4455, "yourpassword")
obs.connect()

async def connect_ai():
    async with websockets.connect("ws://127.0.0.1:8000/chat") as ws:
```

```
    while True:
        msg = input("You: ")
        await ws.send(msg)
        response = await ws.recv()
        print("AI:", response)


        # OBS trigger example
        obs.call(requests.SetCurrentProgramScene("Talking"))
        await asyncio.sleep(2)
        obs.call(requests.SetCurrentProgramScene("Idle"))

asyncio.run(connect_ai())
```

343

Test it:

1. Run your bridge server (bridge_server.py)[344].
2. Run the OBS connector (obs_connector.py)[345].
3. Type a line → your AI responds → OBS automatically switches to the “Talking” scene and back[346].
   That’s a simple example of automation, but you can expand it — for example, trigger emotes or overlays for emotional moments[347].

🎨 Step 6 — Scene Setup in OBS

You can make OBS visually reflect what your AI says[348].

Example setup:

Scene	**Use**
Idle [351]	**Default, waiting mode** [352]
Talking [353]	**When AI speaks** [354]

Happy [355]	**Trigger when [EMOTION:happy] appears** [356]
Angry	Trigger when [EMOTION:angry] appears
Your script can read these emotion tags from the LLM output and switch scenes automatically[357].	
Later, you can even trigger OBS filters (e.g., confetti, lighting effects) for donations or certain emotions[358].	

🧠 Step 7 — Optional: Add Emotion Parsing

Make the AI insert emotion cues into its text for richer animation359.

Add this to your Ollama prompt (in personality.txt):

When expressing emotions, add tags like [EMOTION:happy], [EMOTION:sad], [EMOTION:angry].

360

Then modify your bridge_server.py before TTS:

Python
```
emotion = "neutral"
if "[EMOTION:happy]" in ai_response: emotion = "happy"
elif "[EMOTION:sad]" in ai_response: emotion = "sad"
elif "[EMOTION:angry]" in ai_response: emotion = "angry"

# Remove tags from spoken text
clean_text = ai_response.replace("[EMOTION:happy]", "").replace("[EMOTION:sad]", "").replace("[EMOTION:angry]", "")

tts.tts_to_file(text=clean_text, file_path="voice/ai_line.wav")

# Optionally tell OBS to change scene
```

```
if emotion != "neutral":
    os.system(f'python scripts/obs_change_scene.py {emotion}')
```

This way, your avatar's *mood* matches what it's saying[361].

💻 Step 8 — Launch Everything Automatically

Let's make life easy362.

Create a .bat file to start the full system:

AI_VTuber_Project/start_vtuber.bat

363

Paste this:

```
DOS
@echo off
cd /d "%~dp0"
call .venv\Scripts\activate
start python scripts/bridge_server.py
timeout /t 5
start python scripts/obs_connector.py
start "" "C:\Program Files\VSeeFace\VSeeFace.exe"
exit
```

Now double-click start_vtuber.bat, and your whole AI VTuber stack will boot:

- Bridge server
- OBS automation
- Avatar animation

When you type in the console or connect via a simple chat UI later, your VTuber responds, speaks, and performs live[364].

🧰 Step 9 — Optimize and Test

Performance checklist for your RTX 5070:

Component	What to Check	Fix if Needed

GPU usage > 95%	**Lower LLM model size (mistral instead of llama3)** [368]	**Use smaller quantized model** [369]
Audio delay [370]	**XTTS too heavy** [371]	**Cache frequent words; pre-generate phrases** [372]
Stutters	**OBS rendering too heavy** [373]	**Lower resolution (720p) or disable effects** [374]
Sync drift [375]	**TTS audio plays too early** [376]	**Add time.sleep(0.5) between AI and playback** [377]

You'll quickly find the sweet spot for balance between quality and smooth performance[378].

✅ Chapter Summary

You now have:

- A full working pipeline connected by a FastAPI "bridge."
- OBS automation through WebSockets[379].
- Real-time AI voice + avatar animation[380].
- Optional emotion detection and scene changes[381].
- A single command (start_vtuber.bat) to launch your complete system382.
  Your VTuber now thinks, speaks, moves, and performs — all locally, all under your control383.

🧭 Coming Up Next:

Chapter 7 — Fine-Tuning Personality, Emotions, and Natural Flow

We'll refine your AI's speech tone, add natural timing, and teach it to feel "alive" — subtle pauses, emotion triggers, and humor that sounds human384.

💬 Chapter 7 — Fine-Tuning Personality, Emotion, and Natural Flow

---

You already have the full pipeline built — the brain, the voice, the body, and the stage[385].

Now, we make it believable.

This chapter is all about the “feel” of your VTuber: how they speak, pause, joke, sigh, laugh, and connect emotionally to viewers[386].

We’ll shape tone, emotion, timing, and memory so your AI doesn’t just talk — it performs. [387]

🧠 Step 1 — What “Realistic” Means for an AI Performer

A believable AI character isn’t about being perfect — it’s about rhythm, tone, and intent[388].

When you hear a real person speak, you subconsciously notice:

- **Timing:** short pauses, laughter, filler words[389].
- **Tone:** warmth or sharpness in delivery[390].
- **Emotion consistency:** excitement or calm matching the topic[391].
- **Personality:** quirks, word choices, rhythm of thought[392].

We’ll program these qualities into your VTuber’s LLM and TTS pipeline — no special AI magic needed, just clever scripting and training data[393].

🧩 Step 2 — Add Natural Pacing (Pauses, Breathing, and Timing)

a) Insert pauses directly into the speech text

XTTS (and other TTS engines) can read punctuation like commas and ellipses (...) as pauses[394].

You can make your AI sound alive by preprocessing the text before it’s spoken[395].

Add this helper function to your TTS pipeline (inside talk_pipeline.py):

Python

```
def naturalize_text(text):
    # Add small pauses after exclamation marks, question marks, and long sentences
    text = text.replace("!", "! ...")
    text = text.replace("?", "? ...")
    text = text.replace(",", ", ...")
    # Simulate breathing pauses after long phrases
    sentences = text.split(".")
    new_text = []
    for s in sentences:
        if len(s.split()) > 10:

     new_text.append(s + ". ...")
        else:
            new_text.append(s + ".")
    return " ".join(new_text)
```

396

Then call:

```
Python
ai_output = naturalize_text(ai_output)
tts.tts_to_file(text=ai_output, file_path="voice/ai_line.wav")
```

Your AI will now pause naturally — it's a tiny trick that adds huge realism[397].

---

b) Randomize timing slightly

Human speech isn't perfectly timed. You can add random 0.2–0.6 second delays between phrases:

```
Python
import time, random

time.sleep(random.uniform(0.2, 0.6))
```

Insert it before each audio playback or scene change398.

These tiny imperfections make your AI feel alive instead of robotic399.

🧬 Step 3 — Add Emotional Delivery (Text-Based Emotion Engine)

We'll make your AI "read the room" by interpreting its own emotional tags and slightly altering voice parameters400.

In bridge_server.py, before TTS, add:

Python
```
emotion = "neutral"
if "[EMOTION:happy]" in ai_response: emotion = "happy"
elif "[EMOTION:sad]" in ai_response: emotion = "sad"
elif "[EMOTION:angry]" in ai_response: emotion = "angry"

# Remove tags for cleaner speech
clean_text = ai_response.replace("[EMOTION:happy]", "").replace("[EMOTION:sad]",
"").replace("[EMOTION:angry]", "")

# Change tone slightly
if emotion == "happy":
    clean_text = clean_text.replace(".", "!").replace("...", "!")
elif emotion == "sad":
    clean_text = "... " + clean_text
elif emotion == "angry":
    clean_text = clean_text.upper()  # subtle exaggeration
```

This makes your TTS output *sound different* depending on emotion, even before you fine-tune the voice model[401].

Optional: Use Multiple Voice Profiles for Emotions

XTTS supports switching between trained voice embeddings[402].

You can train multiple voices with RVC and use them as “moods.” [403]

Example setup:

```
voice/happy_voice.pth
voice/sad_voice.pth
voice/angry_voice.pth
```

[404]

Modify your voice_conversion.py to pick a model based on emotion:

Python
```
if emotion == "happy": model = "voice/RVC/models/happy_voice.pth"
elif emotion == "sad": model = "voice/RVC/models/sad_voice.pth"
elif emotion == "angry": model = "voice/RVC/models/angry_voice.pth"
```

Now your AI can literally *sound different* when it feels different[405].

🎭 Step 4 — Teach Your AI to Express Emotion Through Text

You can help your AI produce natural emotion by expanding your fine-tuning data[406].

Inside your dataset folder (brain/training_data), create files like:

emotion_happy.txt
emotion_sad.txt
emotion_angry.txt

[407]

Each file should contain conversational examples showing that emotion:

emotion_happy.txt
User: Hey Nova!
You hit 10,000 subscribers! [cite: 329]
AI: WHAAAT?! That's insane! Thank you so much, chat! You guys are the best!
emotion_sad.txt
User: The stream crashed halfway.
AI: Aww... that sucks. I hate when that happens[cite: 330].
We'll make it better next time, promise.

emotion_angry.txt
User: Someone just called you boring.
AI: EXCUSE ME?! [cite: 331]
I'm literally powered by RTX graphics and caffeine — boring isn't even an option! [cite: 332]

When you fine-tune with Unsloth, the AI learns tone and phrasing patterns automatically[408].

🗣️ Step 5 — Add "Stage Personality" Layers (Quirks, Humor, and Vocabulary)

Every good VTuber has quirks — words they overuse, running jokes, habits[409].

You can teach your AI those quirks using few-shot examples (tiny prompt samples)[410].

Edit your personality.txt like this:

You are Nova, a chaotic but charming VTuber who loves gaming.
You frequently say phrases like "Let's gooo!", "No way, bro!", or "I swear on my GPU." [cite: 337]
You use humor to defuse tension and tease your viewers playfully. [cite: 338]
You sometimes pretend to be overly dramatic for comedic effect. [cite: 339]
If a viewer compliments you, reply with exaggerated excitement.

411411411411This defines behavioral texture — your AI now acts like a streamer, not a narrator[412].

🧩 Step 6 — Add a "Memory Core" (Offline Persistent Personality)

Let's give your AI some continuity — so it remembers who you are or what you said last stream[413].

You already made a short-term memory system in Chapter 3[414].

Now we'll make it save between sessions[415].

Edit memory_chat.py to include:

```
Python
import json, os

def save_memory():
    with open("brain/memory.json", "w") as f:
        json.dump(memory, f)

def load_memory():
    global memory
    if os.path.exists("brain/memory.json"):
        with open("brain/memory.json") as f:
            memory = json.load(f)
    else:
        memory = []

load_memory()

# After each response:
save_memory()
```

[416]Now your AI's memory persists between sessions — like an ongoing relationship with your viewers[417].

🕹️ Step 7 — Add Humor and Flow Filters

Humor can be generated algorithmically using simple pattern matching[418].

Example:

```
Python
def humor_injector(text):
    jokes = [
        "I feel like my code’s about to crash from excitement!",
        "My GPU is blushing right now!",
        "Wait... was that lag or my emotions?"
]
    if "thank" in text.lower():
        text += " " + jokes[0]
    elif "love" in text.lower():
        text += " " + jokes[1]
    elif "game" in text.lower():
        text += " " + jokes[2]
    return text
```

419Run your AI’s response through humor_injector(ai_output) before voice synthesis420.

This keeps tone light and streamer-like without rewriting the model421.

🎵 Step 8 — Add Sound Cues and Background Reactions

You can enhance immersion by layering sound effects dynamically422.

Use FFmpeg or Python’s playsound module to mix background sounds with voice423.

Example:

```
Python
import threading
from playsound import playsound

def play_sfx(file):
    threading.Thread(target=playsound, args=(file,)).start()
```

424

Then:

```
Python
if "[EMOTION:happy]" in ai_response:
    play_sfx("assets/sounds/cheer.wav")
elif "[EMOTION:sad]" in ai_response:
    play_sfx("assets/sounds/aww.wav")
```

That gives your AI sound-based reactions — like laughter or chat cheers[425].

🎯 Step 9 — Timing and Response Flow (Humanizing Interaction)

Humans don't reply instantly426.

A small, variable delay between question and response feels natural427.

Add:

Python
import random, time
delay = random.uniform(0.8, 1.6)
time.sleep(delay)

428Place it right before your AI speaks429.

It makes it seem like the AI "thinks" before answering430.

Combine this with naturalize_text() for realistic pacing[431].

🔧 Step 10 — Fine-Tuning & Testing Strategy

To perfect your AI's vibe:

Step	Action	Goal
1	**Add more sample dialogues** [434]	**Improve emotional consistency** [435]
2	**Fine-tune with Unsloth** [436]	**Capture phrasing quirks** [437]
3	**Test each emotion tag** [438]	**Confirm audio + animation sync** [439]

4[440]	Watch for tone drift	**Adjust prompt or training data**[441]
5	Create a "persona changelog"	**Track evolution of your VTuber**[442]

Fine-tuning is iterative — think of it like rehearsals for your digital actor[443].

✅ Chapter Summary

You've now learned to:

- Add emotion, timing, and pauses to speech[444].
- Give your AI natural humor and tone[445].
- Create emotion-driven audio and animation[446].
- Maintain personality memory across sessions[447].
- Teach your VTuber quirks, humor, and humanity[448].

Your VTuber no longer just *talks* — it *performs, reacts, and connects.*[449]

🧭 Coming Up Next:

Chapter 8 — Streaming, Scenes, and Interactivity

You'll set up OBS for live streaming, add scene triggers, overlays, and chat interaction — turning your creation into a fully interactive performer450.

📺 Chapter 8 — Streaming, Scenes, and Audience Interaction

This chapter turns your project into a full broadcast system: your AI performs on screen, changes scenes when it emotes, reacts to viewers, and even reads chat — all while running offline or semi-local451.

We'll do it safely, efficiently, and in a way that keeps you in control452.

🧠 Step 1 — How Everything Connects

Here's the full flow for streaming:

AI (Ollama) → Voice (XTTS/RVC) → Avatar (VSeeFace/VTube Studio)
↓
OBS captures → Stream output → Chat events → Back to AI brain

Your AI is the *core*, OBS is the *stage*, and chat is the *audience.* [453]

🧩 Step 2 — Set Up OBS Scenes and Sources

Launch **OBS Studio**, then follow this setup:

Create these scenes:

Scene Name	Purpose
Idle [456]	**Default waiting screen** [457]
Talking [458]	**Active AI responses** [459]
Happy [460]	**Triggered by [EMOTION:happy]** [461]
Angry	Triggered by [EMOTION:angry]
Sad [462]	**Triggered by [EMOTION:sad]** [463]

For each scene:

1. Add a **Window Capture** → select your VSeeFace or VTube Studio window[464].
2. Add a **Background** → a looping video, static image, or room scene[465].
3. Add a **Browser Source** for chat overlay (optional).

- For Twitch: https://www.streamlabs.com/widgets/chat-box
- For YouTube: open your live chat popout → copy link[466].

4. Add **Audio Input Capture** → select your virtual cable output[467].

Tip: Lock everything once arranged (to prevent accidental movement)[468].

⚙️ Step 3 — Automate Scene Switching via OBS WebSocket

You already installed the OBS WebSocket plugin earlier469.

Let's make your AI control scene transitions automatically.

In scripts/bridge_server.py, add emotion-based scene triggers like this:

Python
```
from obswebsocket import obsws, requests

obs = obsws("localhost", 4455, "yourpassword")
obs.connect()

def set_scene(emotion):
    scene_map = {
        "happy": "Happy",
        "sad": "Sad",
        "angry": "Angry",
        "neutral": "Talking"
    }
    scene = scene_map.get(emotion, "Idle")
    obs.call(requests.SetCurrentProgramScene(scene))
```

470

Call set_scene(emotion) right after detecting the AI's emotion tags.

Now when your AI says something like:

> "That was awesome! [EMOTION:happy]"

OBS instantly cuts to the "Happy" scene — complete with background and animation[471].

💬 Step 4 — Add Chat Input from Viewers

To make your AI respond to live chat messages, you can pull data from Twitch or YouTube[472].

a) Install pytchat for YouTube or twitchio for Twitch:

pip install pytchat
pip install twitchio

473

b) Example — YouTube Chat Integration

Create scripts/youtube_chat_listener.py:

Python
```
import pytchat, asyncio, websockets

async def connect_bridge():
    async with websockets.connect("ws://127.0.0.1:8000/chat") as ws:
        chat = pytchat.create(video_id="YOUR_LIVE_STREAM_ID")
        while chat.is_alive():
            for c in chat.get().sync_items():
                username = c.author.name

     message = c.message
                print(f"{username}: {message}")
                await ws.send(f"{username} says: {message}")
                reply = await ws.recv()
                print("AI:", reply)

asyncio.run(connect_bridge())
```

474Replace YOUR_LIVE_STREAM_ID with your YouTube live ID475.

Now your AI can read live chat messages and respond on stream — in real time476.

---

c) Example — Twitch Chat Integration

Create scripts/twitch_chat_listener.py:

Python
```
from twitchio.ext import commands
import asyncio, websockets

bot = commands.Bot(token="YOUR_TWITCH_OAUTH_TOKEN", prefix="!",
initial_channels=["YOUR_CHANNEL"])

async def connect_bridge():
```

```
    return await websockets.connect("ws://127.0.0.1:8000/chat")

@bot.event
async def event_message(ctx):
    if ctx.author.name.lower() == bot.nick.lower():
        return
    ws = await connect_bridge()
    await ws.send(f"{ctx.author.name} says: {ctx.content}")
    response = await ws.recv()
    print(f"AI: {response}")

bot.run()
```

[477] This version uses Twitch IRC to feed your chat to the AI and print responses back — all while your VTuber talks out loud[478].

🎮 Step 5 — Add Chat Commands and Reactions

You can define chat-based triggers that cause your VTuber to act out animations or speak special lines[479].

Example for Twitch (inside the same listener):

```
Python
@bot.command(name="dance")
async def dance_command(ctx):
    ws = await connect_bridge()
    await ws.send("!dance")
    print("Dance command triggered!")
```

[480]

Then in your AI prompt, define:

If a user says '!dance', reply with a funny or excited line and the tag [ACTION:dance].

[481] You can use this tag to trigger OBS scene switches or animation files (like your avatar dancing or playing sound effects)[482].

🎨 Step 6 — Create Chat Overlays and Widgets

Make your stream visually reactive with overlays[483].

Free tools to integrate with OBS:

- **StreamElements / Streamlabs:** chat box, follower alerts, and goal meters[484].
- **Kapwing or Canva Video:** animated backgrounds or transitions[485].
- OBS Browser Source: paste the overlay URL486.
  You can also create custom local overlays using HTML/JS — for example, a “mood indicator” bar that lights up depending on [EMOTION]487.

🧠 Step 7 — Real-Time Emote and Expression Control

You can combine emotional tags with physical reactions488.

Add this small listener inside your obs_connector.py:

Python
```
if "[ACTION:dance]" in response:
    obs.call(requests.SetCurrentProgramScene("Dance"))
elif "[EMOTION:happy]" in response:
    obs.call(requests.SetCurrentProgramScene("Happy"))
elif "[EMOTION:sad]" in response:
    obs.call(requests.SetCurrentProgramScene("Sad"))
```

Now, if your AI says something like:

> “Let’s celebrate! [ACTION:dance]”
>
> your OBS switches to a “Dance” scene automatically489.

🎧 Step 8 — Add Background Music and Sound Layers

A lively stream always has ambient sound490.

Add background music (BGM) and effects that react dynamically491.

- Place your music files in assets/music/[492].
- Add them to OBS as **Media Sources** → loop enabled[493].
- Control them via Python:

Python
```
os.system("nircmd.exe mutesysvolume 0")
os.system("nircmd.exe setsysvolume 50000")  # volume 50%
```

[494]You can crossfade between BGM tracks using simple fade scripts or OBS audio filters[495].

🔄 Step 9 — Build a Stream Startup Script

Simplify your life — one click to launch everything[496].

Create start_stream.bat:

```
DOS
@echo off
cd /d "%~dp0"
call .venv\Scripts\activate
start python scripts/bridge_server.py
timeout /t 3
start python scripts/youtube_chat_listener.py
timeout /t 2
start "" "C:\Program Files\OBS Studio\bin\64bit\obs64.exe"
start "" "C:\Program Files\VSeeFace\VSeeFace.exe"
exit
```

[497] Now you just double-click one file — your entire ecosystem wakes up[498].

⚙️ Step 10 — Performance & Stability Tips

You're running a *lot* of systems at once. Keep it stable:

Tip	Description
Use smaller LLMs (e.g. Mistral)	**Keeps latency low** [501]
Preload XTTS [502]	**Generates voice faster after first use** [503]
Reduce OBS sources	**Avoid 4K backgrounds or filters** [504]

Limit memory growth [505]	**Clear chat log after 200 messages** [506]
Restart after long sessions [507]	Python memory can slowly climb [508]
You can monitor your system in **Task Manager → Performance → GPU**[509].	
Aim for <80% sustained GPU use for smooth real-time streaming[510].	

🧠 Step 11 — Optional: Offline Chat Logs & Replay

To simulate chat when testing offline, create a text file like mock_chat.txt:

User1: Hey Nova!
User2: Sing for us!
User3: You're so funny today.

[511]

Run this Python snippet to feed it into your AI bridge:

Python
```
import websockets, asyncio

async def mock_chat():
    async with websockets.connect("ws://127.0.0.1:8000/chat") as ws:
        for line in open("mock_chat.txt", "r").readlines():
            await ws.send(line.strip())
            reply = await ws.recv()
            print("AI:", reply)

asyncio.run(mock_chat())
```

Great for rehearsing your AI's flow and testing timing before going live[512].

🧩 Step 12 — Stream Checklist

Before each stream, check these boxes:

- [ ] Start virtual cable audio[513].
- [ ] Launch start_stream.bat[514].
- [ ] Confirm bridge server says “Online.”[515]
- [ ] Test lip sync and voice playback[516].
- [ ] Check OBS scenes switching with emotion tags[517].
- [ ] Load background music[518].
- [ ] Start stream (Twitch / YouTube)[519].
- [ ] Monitor performance (CPU/GPU <80%)[520].
- [ ] Smile — you’re live[521].

✅ Chapter Summary

You now have a **fully functional AI VTuber streamer**:

- OBS scenes and overlays
- Emotion-driven automation
- Live chat integration (Twitch/YouTube)
- One-click startup workflow
- Smooth performance on your RTX 5070

Your AI doesn’t just exist — it performs, reacts, and entertains[522].

🧭 Coming Up Next:

Chapter 9 — Optimization, Safety, and Upgrades

We’ll make your setup bulletproof: safe chat filtering, faster AI responses, smoother audio, and ways to expand with more characters523.

⚙️ Chapter 9 — Optimization, Safety, and Upgrades

🧱 1. Performance: squeeze every frame

a) Use the right model size

- Phi-3 Mini (3.8 B) → fastest;
  still witty enough for chat[524].
- **Mistral 7 B** → sweet spot between brains & speed[525].
- Llama 3 8 B Instruct → heavy; only for story segments[526].
  Swap models on the fly in Ollama:

Bash
ollama pull mistral
ollama run mistral

[527]

…and update your bridge script's ["ollama","run","mistral"] line.

b) Quantization = free speed

Every Ollama model can run in 4-bit:

Bash
ollama run mistral:Q4_K_M

VRAM use drops by ~40 %, output stays human-level[528].

c) Batch the TTS calls

When your AI spits multiple sentences, generate one audio clip instead of five:

Python
tts.tts_to_file(text=" ".join(sentences), file_path="voice/full_line.wav")

Less I/O = less delay + fewer mouth-sync hiccups[529].

d) Cache frequent replies

Add this near your TTS call:

Python
```
from hashlib import md5
cache = "voice/cache/"
os.makedirs(cache, exist_ok=True)
key = md5(ai_output.encode()).hexdigest()[:10]
path = cache + key + ".wav"
if not os.path.exists(path):
    tts.tts_to_file(text=ai_output, file_path=path)
```

```
else:
    print("🎧 Cached voice used")
os.system(f"start {path}")
```

Common greetings (“Hey chat!” etc.) play instantly[530].

🔊 2. Audio stability & lip-sync accuracy

a) Keep sample rate consistent

All audio → 16 kHz mono 16-bit PCM:

Bash
```
ffmpeg -i input.wav -ar 16000 -ac 1 -sample_fmt s16 output.wav
```

531

b) Synchronize voice ↔ avatar

Give VSeeFace a 50 ms buffer:

> **VSeeFace → Settings → Audio Delay = 50 ms**

That matches average XTTS latency[532].

c) GPU vs CPU usage

If XTTS hogs your GPU:

Bash
```
setx TTS_USE_CUDA 0
```

533

XTTS will use CPU; Mistral keeps GPU534.

Handy when voice and brain fight for VRAM535.

🔐 3. Safety and Content Filtering

a) Input moderation

Install detoxify:

Bash
```
pip install detoxify
```

Add before sending user text to Ollama:

Python
```
from detoxify import Detoxify
if Detoxify('original').predict(user_input)['toxicity'] > 0.6:
    print("🚫 Message blocked")
    continue
```

536

b) Output moderation

Quick keyword filter:

Python
```
banned = ["politics","nsfw","violence"]
if any(word in ai_output.lower() for word in banned):
    ai_output = "Let's talk about something nicer!"
```

537

c) Ethical mode

In your prompt:

```
You are an entertainer who keeps content safe for all ages.
If a viewer asks for inappropriate topics, redirect politely.
```

538

d) Log everything privately

Python
```
with open("logs/session.txt","a",encoding="utf8") as f:
    f.write(f"User:{user_input}\nAI:{ai_output}\n---\n")
```

You'll catch bugs and bad responses before they reach chat[539].

⚙️ 4. System Reliability (so it never crashes mid-stream)

a) Restart loops gracefully

Wrap main loop in try/except:

Python
```
try:
    # main chat loop
```

```
except Exception as e:
    print("⚠️ Error:",e)
    time.sleep(3)
```

540

b) Auto-recover OBS connection

If WebSocket drops:

```
Python
while True:
    try:
        obs.call(requests.GetVersion())
        break
    except:
        time.sleep(2)
        obs.connect()
```

541

c) Watchdog monitor

Use psutil to auto-relaunch crashed processes:

```
Bash
pip install psutil
```

Then check every 10 seconds whether bridge_server.py

and VSeeFace.exe are alive542.

⚡ 5. Latency Optimization (instant chat feel)

Bottleneck	**Fix**
LLM delay	**Use Mistral 7B 4-bit or Phi-3 Mini** [545]
TTS delay [546]	Keep XTTS model in RAM;

don’t reload each time [547]	
Disk I/O	**Store temp audio in RAM disk or tempfile** [548]
OBS lag [549]	**Lower preview resolution or disable stats overlay** [550]
Multi-thread sync	**Use asyncio for bridge server calls** [551]

🧩 6. Scalability — Multi-Character or Co-Host Setup

You can spawn multiple AIs locally — each with its own model, voice, and personality[552].

Example: two VTubers talking

Run two Ollama instances on different ports:

Bash
```
ollama serve --port 11434  # Nova
ollama serve --port 11435  # Airi
```

553

In Python:

Python
```
def ask(model_port, text):
    return subprocess.run(
        ["ollama","run",f"--port={model_port}","mistral"],
        input=text.encode(),capture_output=True,text=True
    ).stdout.strip()

nova = ask(11434,"Hey Airi!")
airi = ask(11435,f"Nova said: {nova}")
```

They talk to each other like a podcast duo[554].

🔄 7. Maintenance & Updates

Task	Frequency	Command
Update Ollama models	**Weekly** [558]	**ollama pull mistral** [559]
Clear cache/audio [560]	Monthly [561]	**Delete voice/cache/** [562]
Archive logs [563]	**Monthly** [564]	**Move logs/*.txt to logs/archive/** [565]
Test TTS voice [566]	When tone drifts	**Regenerate RVC model** [567]
Backup fine-tunes [568]	After training	Copy brain/fine_tuned/ to USB or cloud storage [569]

🧠 8. Diagnostics Dashboard (optional but cool)

Install rich for color output:

Bash
pip install rich

570

Add live stats:

Python
from rich.console import Console
from rich.table import Table
import torch, psutil, time

console = Console()
while True:

```
    gpu = torch.cuda.memory_allocated()/1e9
    cpu = psutil.cpu_percent()
    table = Table("Metric","Value")
    table.add_row("GPU Memory (GB)",f"{gpu:.2f}")
    table.add_row("CPU Usage (%)",str(cpu))
    console.clear();
    console.print(table)
    time.sleep(3)
```

[571] Keep it running in a side terminal to spot lag before viewers do[572].

🔐 9. Privacy & Offline Backups

Even though you're offline:

- Store all chat logs locally, not in clouds[573].
- Use Windows BitLocker or VeraCrypt to encrypt your AI_VTuber_Project folder[574].
- Backup fine-tuned weights and configs to an external drive[575].

🧰 10. Future-Proof Upgrades

Upgrade	**What it adds**	**Free tool**
Face Tracking 2.0	**Better eye movement** [579]	**MediaPipe FaceMesh** [580]
Gesture Control [581]	**Hand tracking for VRoid** [582]	**LeapMotion Controller SDK** [583]
Multi-language support	**Auto-translate chat** [584]	**Argos Translate** offline [585]
Singing Mode [586]	**AI vocals for songs** [587]	**DiffSinger or RVC song mode** [588]

Camera moves 589	OBS scene transitions on emotion	**Built-in OBS Move Transition plugin** 590

✅ Chapter Summary

You now know how to:

- Reduce lag and GPU usage with quantization & caching[591].
- Keep audio and lip sync tight and clean[592].
- Add moderation and safety layers for family-friendly streams[593].
- Auto-recover crashes and monitor performance[594].
- Expand your setup to multiple AI characters[595].
- Maintain and upgrade your project like a real studio596.
  Your AI VTuber is now stable, safe, and scalable — ready for hours of continuous performance597.

🧭 Coming Up Next

Chapter 10 — Final Touch: Quick Reference Sheet & Daily Workflow

You'll get a compact cheat-sheet of all commands, folders, and launch steps to keep beside you when streaming598.

🗒️ Chapter 10 — Quick Reference Sheet & Daily Workflow

🗂️ 1 – Folder Layout

```
AI_VTuber_Project/
 ├─ brain/          # prompts, memory.json, fine-tunes
 ├─ voice/          # TTS, RVC models, cache, audio clips
 ├─ avatar/         # VRM / Live2D models, physics, exports
 ├─ scripts/        # bridge_server.py, talk_pipeline.py, etc.
 ├─ assets/         #
music, sfx, overlays
 ├─ logs/           # session logs
 └─ .venv/          # Python environment
```

599

⚙️ 2 – Core Launch Commands

Bash
Activate environment
.venv\Scripts\activate

Start bridge server (AI ↔ TTS ↔ OBS)
python scripts/bridge_server.py

Start OBS connector
python scripts/obs_connector.py

Run full chat pipeline manually
python scripts/talk_pipeline.py

Fine-tune a model
unsloth train --config brain/unsloth_config.yaml

Test voice
python scripts/tts_test.py

600

🧠 3 – Model Management

Task	Command	Note
List models	**ollama list** 604	**Shows all local brains** 605
Pull new one	**ollama pull mistral** 606	**Downloads 7 B model** 607

Run model [608]	ollama run mistral	**Chat test** [609]
Delete model [610]	ollama rm mistral	**Frees space** [611]
Fine-tune load [612]	ollama create nova-mistral -f ./brain/fine_tuned/adapter.safetensors	Adds your LoRA personality [613]

🔊 4 – Audio & Voice

Action	Tool	Shortcut
Convert text → voice	**XTTS v2** [617]	**tts.tts_to_file(text, "file.wav")** [618]
Custom voice	**RVC** [619]	**python scripts/voice_conversion.py** [620]
Check sync	**VSeeFace** [621]	**50 ms audio delay** [622]
Mute AI quickly [623]	Win + F1 (OBS Hotkey)	**assign manually** [624]

🎨 5 – Avatar & Scenes

Step	Where	Notes

Load model	VSeeFace / VTube Studio	**.vrm or .moc3** [628]
Idle scene [629]	OBS → Scene: Idle	**default** [630]
Emotion scenes [631]	OBS → Happy/Sad/Angry	**linked via tags** [632]
Dance scene [633]	OBS → Dance	**triggered by [ACTION:dance]** [634]

🧩 6 – Emotion Tags & Reactions

Tag	Result
[EMOTION:happy]	Scene → Happy, cheer sfx
[EMOTION:sad] [637]	Scene → Sad, “aww” sfx [638]
[EMOTION:angry]	**Scene → Angry, red overlay** [639]
[ACTION:dance]	**Scene → Dance** [640]
[ACTION:idle] [641]	Scene → Idle [642]

🧰 7 – Maintenance Checklist

Task	Frequency	Tool
Update drivers & CUDA	**Monthly** [646]	NVIDIA site [647]
Update Ollama models [648]	Weekly	**ollama pull** [649]
Clear cache/audio [650]	**Monthly** [651]	**delete voice/cache** [652]
Backup fine-tunes	After training	**copy brain/fine_tuned/** [653]
Test voice latency	Before streams	**tts_test.py** [654]

🧠 8 – Performance Targets (for RTX 5070)

Component	Ideal Usage	Notes
GPU VRAM	**≤ 10 GB of 12 GB** [657]	use Q4 models [658]
CPU Usage [659]	≤ 60 %	**offload TTS to GPU if free** [660]
Latency [661]	< 2 s reply time	**combine TTS cache + smaller LLM** [662]
FPS (VSeeFace)	**45–60 fps** [663]	**limit physics bones** [664]

🧠 9 – Startup Routine (5 minutes before stream)

1. Plug in camera + mic.
2. Start Virtual Audio Cable.
3. Double-click start_stream.bat[665].
4. Verify bridge says “Online.” [666]
5. Check OBS audio levels + scene switches[667].
6. Test one AI line and watch lip-sync[668].
7. Hit “Start Streaming.”
8. Keep Task Manager open (minimized)[669].

🧩 10 – Emergency Shortcuts

Key	Action
Ctrl +C	**Stop Python loop safely** [672]
Alt + Tab → OBS → Stop Virtual Cam [673]	**recover if VSeeFace freezes** [674]
Win + M [675]	**mute system audio** [676]
Alt + F4 [677]	**force close bridge window (crash recovery)** [678]
Ctrl + Shift + Esc	**open Task Manager to kill stuck process** [679]

🗒️ 11 – Troubleshooting Cheat Lines

Issue	Fix
“CUDA out of memory”	**switch to Phi-3 or Q4 model** [682]
“ffmpeg not found” [683]	add C:\ffmpeg\bin to PATH [684]
“OBS not responding” [685]	disable preview / restart WebSocket [686]
“Voice delay” [687]	**lower sample rate to 16 kHz** [688]
“No mouth movement” [689]	check audio input device = CABLE Input [690]

🎉 12 – Shutdown Routine

1. Say goodbye to chat[691].
2. In OBS → **Stop Streaming**[692].
3. Close VSeeFace first → OBS → bridge server[693].
4. Deactivate .venv[694].
5. Copy logs/session.txt → archive[695].
6. Power nap (optional but recommended)[696].

✅ End of Handbook

You now own a complete, self-sufficient AI VTuber studio:

- **Offline brain, voice, body, and show.**

- One PC, one click, one performer697.
  Keep the reference sheet printed or pinned beside OBS, and treat your AI like any other on-air talent—she just happens to live inside your GPU698.
