The Technology Behind ‘Mirror, Mirror’

Introduction

This document provides a detailed overview of the technology powering *Mirror, Mirror*. It explains the tools chosen for each part of the system, the reasons behind those decisions, and how the components connect to form a seamless, end-to-end pipeline—from a visitor speaking, to a virtual human responding in real time.

Choosing the right technology

"Mirror, Mirror" is an intimate, philosophical installation that confronts visitors with a digital copy of themselves that speaks with their own voice and appearance.

The cloning needs to happen in real-time. We cannot pre-train any models as we do not know beforehand who will use ‘Mirror, Mirror’. We also cannot pre-render the cloning, as the concept is designed to be a 5 minute experience. A longer experience would be impractical in a crowded place like Dutch Design Week. It is also crucial that the cloning is possible with a very small input sample from the user. It is not practical to use a voice cloning technique that requires a 10 minute voice sample, for example.

Controlling the latency is crucial. We need to simulate a conversation with a human, so the latency has to be as low as possible. The experience can’t be good if the latency between input and response is 20 seconds, for example. I will need to research what users find to be the acceptable latency threshold and try to get the total end-to-end latency below that.

To break it down, the technical requirements to realize this concept are:

- A realistic virtual human with realistic mouth movements based on its speech
- A way to clone a face in real-time, with little training data needed
- A way to clone a voice in real-time, with little training data needed
- An AI model that can respond with as little latency as possible

Virtual Human with real-time lip-sync

This was not much of a challenge for me to figure out. I have past experience with creating AI virtual humans that can talk, which means that I can re-use the same method here.

I am using **Unreal Engine 5** and their MetaHuman system for creating the virtual character. For lip-syncing, I use a separate tool by Nvidia called **Audio2Face**, which is part of their Omniverse ecosystem. You can use this tool to pre-render lip-sync animations, which is how gaming studios use it, but you can also enable LiveLink. Enabling LiveLink allows you to connect the A2F tool directly to Unreal Engine and stream the blendshapes in real-time.

This works really well and although I need to measure the exact latency during the benchmarks, the first impressions seem very good.

If I want to integrate the A2F tool into my pipeline, I need to build a Python server that will automatically send all audio output from the Virtual Human in ‘.wav’ format to the A2F Streaming Audio Player.

Face Cloning

I spend a lot of my time on Twitter because I find it to be a massive news source for any machine learning related news. When I knew that my concept included face cloning, I immediately thought of a demo that I saw 2 months ago, where someone deepfaked their webcam feed in real time. This was done with an open source tool called **Deep-Live-Cam.**

First I tried running the Deep-Live-Cam tool on my Macbook Pro (M1 Pro), but that was not a success. The tool was running at a 0.5fps framerate, so a new image every two seconds. Not very usable. I have a PC at home with an RTX 4090 card, so I tried running it on that PC instead. This got me about 20 fps. Still not perfect, but definitely usable.

I tested Deep-Live-Cam with a real target first. Because I do not have a webcam connected to my PC, I used OBS Studio instead. OBS is open source screen capturing software and it has a neat feature where it can simulate a webcam feed. So I screenshared my browser with a video of Keanu Reeves running and deepfaked my face over it. This worked pretty well!

After that, I tested it with a MetaHuman, which worked even better! Because a MetaHuman is more static than a real person, it results in a more stable deepfaked image. A concern was that Unreal Engine took away a lot of GPU resources, which dropped the deepfaked framerate significantly. More specifically, the frame rate went from *~20 fps to ~10 fps*. I fixed this by capping the framerate in Unreal at 24 fps, which restored the deepfake fps to *~18-19 fps*. You can find a video of the test in the PortFlow evidence.

The Deep-Live-Cam offers a way to control the tool through the CLI interface, which makes this convenient to fit in my pipeline.

Voice cloning

For voice cloning, my thoughts immediately went to 'F5-TTS'. **F5-TTS** is a text-to-speech model that can synthesize a voice based on a 5-9 seconds voice sample. The model is open-source and you can run the model locally.

I tested the model with Umberto, a fellow student. I wanted to see whether it worked well and what his reaction would be. I was testing it on my MacBook Pro, which does not have the proper hardware to run a TTS model locally, so I used a cloud GPU (HuggingFace Space). You can view the video of this test in the PortFlow evidence!

During the test using the HF Space, the latency was around 10 seconds. Running it locally on my RTX 4090 with CUDA 12.4 brings that down to <2 seconds.

F5-TTS is a text-to-speech model, as mentioned earlier. Not a speech-to-speech model. So when the user speaks, I need to find a way to transcribe it almost instantly and send that to the LLM model. **Whisper by OpenAI** is one of the best models and one of the few models that they've actually made openly available. I tried running it locally in the browser using GPU acceleration with WebGPU and it transcribes almost instantly. This is the method I'm using for now, but I might use a more native solution if this becomes necessary later in the development process.

The F5-TTS inference tool from Github has a CLI interface, which makes this convenient to fit in my pipeline. If needed, I could also adjust the code so that it fully integrates with the rest of my pipeline.

AI LLM model

I am using Gemini as my LLM and the reason is simple: it's free. **Gemini 2.0 Flash** is incredibly fast. Faster than any other AI I have used, although I will do benchmarks to back this up. But the biggest advantage is that the API is completely free for experimental models. This makes it easier for me to experiment with it.

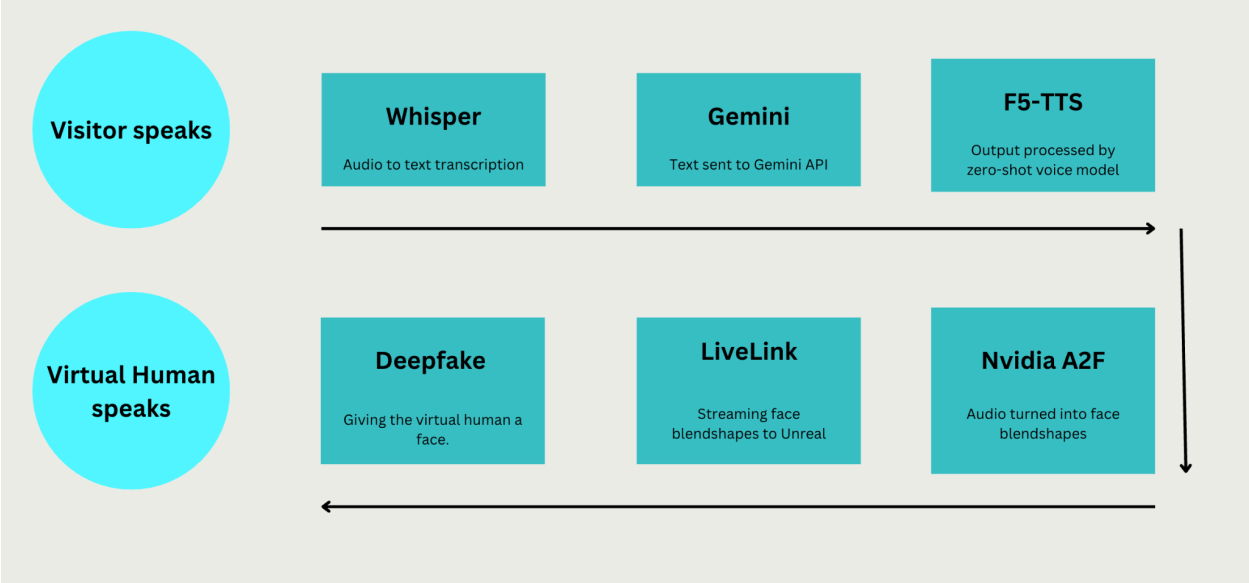
Of course, another way to avoid API costs is to run the model locally. However, I choose not to due to two reasons:

- Local LLMs running on consumer GPU's just aren't very fast. I will later test this out in the 'Benchmarks' section of this document, but my experience so far has been anything but good. The only reason to get a fast experience with local LLM models is by running a very small model. But I'd rather not sacrifice quality. The conversation has to feel natural.
- I am currently running multiple tools on a single PC and these tools are all quite demanding. Wherever I can take some load off the GPU, I will opt to do so.

The Pipeline

So based on the technologies that I talked about in detail above, I created this pipeline:

1. Visitor speaks.
2. Whisper instantly transcribes the visitor's speech.
3. Gemini receives the transcription. Gemini has been finetuned to give the right answers that follow the script of the experience.
4. Gemini's output is sent to F5-TSS, which turns it into speech audio, based on the visitor's voice.
5. The audio is sent as a '.wav' file to a headless A2F server
6. The audio is processed into blendshapes, which are sent to Unreal Engine using LiveLink.
7. The talking MetaHuman is deepfaked, using an image of the visitor.
8. Virtual Human speaks.



Challenges and solutions

This is where I will document the challenges I faced during the process and how I fixed them. For example: “The end-to-end latency is too high” or “How do I get the voice + face sample from the visitor?”

Benchmarks

This is where I will publish the results of the benchmarks.