Scthe / **ai-iris-avatar** Public

master · Go to file · Code

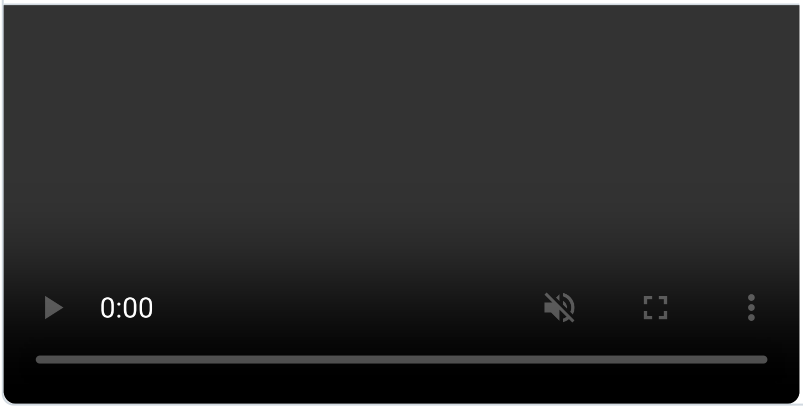
Scthe Allow streaming without deepspeed · 2ee9f88 · 10 months ago

server	Allow streaming without deepspeed	10 months ago
unity-project	Small fixes next day after GitHub	last year
.gitignore	Add xtts scripts to select best speaker	last year
INSTALL_AND_USAGE.md	Update INSTALL_AND_USAGE.md	last year
LICENSE.md	Add readme and license	last year
README.md	Add videos README.md	last year
config.example.yaml	Allow streaming without deepspeed	10 months ago
config_xtts.yaml	Small fixes next day after GitHub	last year
main.py	Allow streaming without deepspeed	10 months ago
makefile	Server: Add tts-first-chunk timings	last year

AI-Iris-Avatar

Talk with AI-powered detailed 3D avatar. Use large language model (LLM), text-to-speech (TTS), Unity game engine, and lip sync to bring the character to life.

prompt-answer-out.mp4



In the video we are asking the character "Who is Michael Jordan?". The avatar 'answers' the question after a short delay. Using the previous messages as a context, we can have entire conversatons. Notice the hair physics and blinking!

vfx-out.mp4

Showcase of remote events triggered from web browser. After selecting each VFX, the respective particle system is played. Popular usage is a firework particle effect when someone donates $5 on Twitch etc. During rain VFX you might even notice splash and bounce when droplet interacts with the character (top of the hair).

Feature set

The core functionality is a custom 3D model that 'speaks'. It emits voice and uses Oculus' lip sync library to give a (hopefully convincing) impression. Here is a feature set:

- Runs **100% locally** on your hardware. No internet connection is required.
- Everything is **configurable**. Swap large language models or voices. Replace the entire 3D model or just the hair color. It's your character.
- Choose **any large language model(LLM)** you want. Most LLMs express biases, are focused on American culture, or are forbidden to talk about topics like politics. In my app, you can use uncensored models.
- **Add custom knowledge base** to LLM. A few weeks ago I wrote about how to use retrieval-augmented generation to provide custom data for LLMs. Imagine you can ask a character from a movie/video game to tell you more about its world.
- **Chat context.** Previous messages affect subsequent questions.
- **Many voices** (both male and female) are available. And if you don't like any, swap the text-to-speech (TTS) model. Voice cloning is also an option (see usage FAQ).
- Fast LLM and TTS (with streaming and deepspeed) for **<4s response time** (see videos). It actually feels like a real-time conversation.
- Power of **Unity game engine**. If it's good enough for a $1 billion per year gross video game, it is enough for everyone.

    - You can also use specialized plugins like Magicacloth2 for clothes. Or keijiro's KvantWig (DO IT!).
- **Lip sync** for automated mouth movement. Uses shape keys and works with any voice type. The delay between audio and lip sync is customizable. It's more natural to hear the sound after the lips move.
- **3D skeleton animation.** E.g. different animations when the character is idle and when it is speaking. Each such state machine randomly selects clips to play. All integrated into Unity's mecanim system.
    - Animations I've used come from the Adobe Mixamo library.
- **Trigger events remotely.** E.g. play particle effect based on a button in the web browser. Popular usage is a firework particle effect when someone donates $5 on Twitch etc.
- Small features that **make the character come to life**:
    - **Eyes controller.** You can track any object regardless of the whole body motion. The pupils are constrained to not 'leave' the eyes. By default, eye movement has a small stutter known as saccade. It's based on both time interval and angle delta. Can be switched off if you want.
    - **Hair physics.** Implemented as bones that follow spring physics (and collision spheres). Provides secondary animation through hair or jewelry. E.g. fluttering earrings after head movement. With weight painting, it can also simulate more organic/softer objects. Think ribbons, hair bows, cloth, etc.
    - **Blinking.** Configurable interval and duration. Uses shape keys internally.
    - A few other small interactions. I don't want to spoil too much, but I hope you will notice it!
- **Reconnect** if the connection to the server is lost.
- Can be configured to **run the Unity client on mobile devices**. Requires converting the project from HDRP to URP. After that, just point WebSocket to the PC server.
- **Open source.**

How does this work?

1. **The user** runs the Unity client that connects to the Python server.
2. **The user** inputs the query using text. It is sent to the **server** through the WebSocket.
3. **The large language model (LLM)** takes the query and previous messages (chat context) to generate a text response.
4. **Text-to-speech (TTS)** generates voice.
5. **The server** sends bytes of the speech as WAV to the **Unity client**.
6. **The Unity client** uses the audio to apply lip sync using the Oculus Lipsync library.
7. **The Unity client** plays the audio. The delay between lip sync and audio is configurable. It's usually better to change mouth shape before the sound is emitted.

The flow does not rely on any particular implementation. Feel free to mix and match LLMs, TTSs, or any suitable 3D models (requires specific shape keys). As you might notice, this architecture gives us ultimate flexibility. As you might imagine, the previous sentence is an understatement.

> There is no speech recognition, the prompt is text-only. It would be trivial to add this feature using Whisper Fast. See below for instructions. TL;DR send GET or POST to `/prompt` endpoint.

How fast does this work?

Using TTS with streaming and DeepSpeed, I usually get a <4s response (from sending the prompt to the first sound). It's small enough, that conversation feels real-time. At this point, the bottleneck is the LLM. On a single GPU you can't run LLM and TTS at the same time (I have tried, check the FAQ about `tts.chunk_size` config option). We have to first generate all text tokens and only then generate sound. I've tried offloading TTS to the CPU, but this also struggles.

Streaming means that we split the generated text into smaller chunks. There is a small crossfade to mask chunk transitions. A small first chunk means fast time-to-first-sound. **DeepSpeed** is a Microsoft library to speedup GPT inference. Both streaming and DeepSpeed are optional but recommended.

The first question after the server starts always takes the longest (~10s) as the server has to load the AI models. When used in the Unity editor, you will rarely have a garbage collection pause (kinda noticeable with audio). But I would be surprised if you actually got a GC issue in the production build.

I've got to say, I'm amused. I expected some problems when using the same GPU for both Unity rendering and the AI. I knew that an Android/iOS app was an easy fallback to offload the Unity cost to a separate device. But it's not necessary on my hardware. It's kind of unexpected that it works smoothly. Ain't gonna complain. I also limited Unity to a 30FPS (just in case).

Specs

- GPU: RTX 3060 with 12GB of VRAM. It's a $250 GPU, comfortably in the consumer range.
- CPU: AMD Ryzen 5 5600.
- RAM: 16GB.
- OS: Windows 11.
- Hard drive: a lot.
- Unity: 2022.3.21f1. The latest LTS at the moment of writing.

- LLM: gemma:2b-instruct (3B parameters). Docs on ollama.
- TTS: XTTS v2.0.
- Default resolution: 1280x720. You can go higher or DLSS/FSR upscale it.

Further optimizations

- LLM quantization to process more elements at a time. As a by-product, maybe even fit a 7B model?
- Faster text-to-speech model. Just select one from the TTS library. Or use Piper which runs even on Raspberry Pi.
- Use libraries like DeepSpeed, flash-attention, and others.
  - DeepSpeed is already integrated into the app. It will be automatically detected if installed. See INSTALL_AND_USAGE.md for more details.
- FSR/DLSS for higher video resolution. Unity has FSR 1 build-in.
- Some Unity performance magic. I'm not an expert. Or don't use HDRP.
- Optimize the 3D model, use less expensive shaders, smaller textures, etc.

If you go to the control panel you will see the timings for each response stage. For Unity, use the built-in profiler.

Usage

See INSTALL_AND_USAGE.md. It also includes instructions on how to use/expand current features.

FAQ

The questions below are about general the philosophy of this app. For a more usage-oriented FAQ, see INSTALL_AND_USAGE.md.

Q: What is the value added?

This app shows we already have the technology to render a detailed 3D Avatar and run a few neutral nets on a single consumer-grade GPU in real-time. It is customizable and does not need an internet connection. It can also work in a client-server architecture, to facilitate e.g. rendering on mobile devices.

Q: Why create a custom 3D model?

I could have used the standard Sintel model. I've created my own character because, well, I can. From dragging the vertices, painting the textures, animating the mouth, and adjusting hair physics to a 'talking' 3D avatar. Quite an enjoyable pastime if I do say so myself.

I've also wanted to test texture reprojection from a stable diffusion-generated image. E.g. you can add 'bald' to the positive prompt and 'hair' to the negative. It does speed up workflow a lot. Alas, reprojection will have specular highlights, etc. to remove manually.

I've used Sintel as a base mesh as it already has basic shape keys. Especially to control each part of the mouth - just add Blender 4.0-compatible drivers. This made it trivial to create viseme shape keys. I've already used Sintel's model many times in the past, so it was a no-brainer for this project.

PS. I hate rigging.

Q: How does this differ from stable diffusion-generated avatars?

You've probably seen 'talking' real-time stable diffusion-generated virtual characters. It is a static image with the mouth area regenerated on every frame based on sound. You will notice that it's unstable. If you diffuse teeth every frame, they will shift around constantly. I've used stable diffusion a lot. I've seen my share of mangled body parts (hands!). It's... noticeable with teeth. A popular implementation is SadTalker. It even has Stable Diffusion web UI extension.

Instead, my app uses boring old technology that has been in video games for years. If you have hundreds of hours of dialogue (Baldur's Gate 3, Cyberpunk 2077, etc.), you can't animate everything by hand. Systems like JALI are used in every major title.

If you want real-time animated characters why use solely AI? Why not look for solutions used by the largest entertainment sector in the world? At the very least you could use it as a base for img2img. In recent years we also had VTubers, which push the envelope each day. A lot of this stuff is based on tech developed by Hatsune Miku fans.

Q: How does this differ from Neuro-sama?

Neuro-sama is a popular virtual streamer. It's an AI-driven character that plays video games and talks with its creator, Vedal. Here is how my app stacks against it:

- 100% local. AFAIK it was not published how exactly Neuro-sama works. It's speculated that text-to-speech is MS Azure's Ashley with a higher pitch. With my app, you can inspect the code. Or unplug the RJ45 cable.
- 3D. Nearly all VTubers use VTubeStudio. It's great if you are going for the anime look. However, some might want to experiment with the 3D models. Be that for realistic lighting or interactive physics. As for me, I grew up watching Toy Story 1 in primary school.

- Everything is configurable. Swap large language models or voices. Replace the entire 3D model or just the hair color.
- Uses the Unity game engine. You can do literally everything.
- Open source.

Q: What is the license?

> This app includes source code/assets created by other people. Each such instance has a dedicated README.md in its subfolder that explains the licensing. E.g. I've committed to this repo source code for the "Oculus Lipsync" library, which has its own license (accept it before use!). XTTS v2.0 is also only for non-commercial use. The paragraphs below only affect things created by me.

It's GPLv3. It's one of copyleft licenses. GPL/copyleft licenses should be familiar to most programmers from Blender or Linux kernel. It's quite extreme, but it's dictated by the nature of the app. And, particularly, one of the possible uses.

Recently I've watched "Apple's $3500 Nightmare" by Eddy Burback. It's a review of the $3500 (!) Apple Vision Pro. One of the presented apps allows the user to date an AI "girlfriend". The interface has a stable diffusion-generated image on the left (I smell PastelDiffusedMix with Seraphine LoRA?). Text chat on the right. Is that the state of the art for this kind of software? It's lazy.

Ofc. the mobile dating apps were filled with controversies from the get-go. Tinder and Co. do not want to lose repeat customers. Scams galore before we even get to machine learning. There are millions of AI profiles on Tinder. And with straight-up AI dating it's a whole other issue.

Q: Can I use this for realistic models?

You can use any model you like. Lip sync uses shape keys that correspond to ovrlipsync's visemes. With the "Enemies" tech demo, Unity has proven that it can render realistic humans.

Personally, I would use Unreal Engine's metahuman. You would have to rewrite my Unity code. For this effort, you get a state-of-the-art rig and a free high-fidelity asset. You could also try to import metahuman into Unity.

For some reason, Unity does not have a built-in pipeline for human characters. Even when creating the "Enemies" cinematic linked above, they did not bother to make it community-viable. It's a custom set of tools tailored to Autodesk Maya. And I've never heard about the 4D clip file format. Congratulations to the project lead! It's a baffling decision. E.g. they have their HairFX for hair rendering and simulation. It's based on TressFX. I've ported TressFX to OpenGL, WebGL, and Vulkan. I understand it quite well. And yet this app uses hair cards!

Original Sintel has splines-based hair, this should have been a simple export operation. These systems need proper documentation.

At the end of the day, the tool is just a tool. I wish Unity got their priorities in order. I'd say rendering people is quite important in today's market.

Q: Can I use this for 2D/anime models?

Yes, but make sure you understand why you want to use a 3D engine for a 2D rendering technique. For Guilty Gear Xrd, the authors had to tweak normals on a per-frame basis. Even today, 3D is frowned upon by anime fans. The only exception (as far as I know) is Land of the Lustrous. And this is helped by its amazing shot composition.

Looking at Western real-time animation we have e.g. Borderlands. It replicates the comic book style using flat lighting, muted colors, and thick ink lines. There are tons of tutorials on YouTube for flat shading, but you won't get a close result without being good at painting textures.

While this might sound discouraging, I want you to consider your goal. There is a reason why everyone else is using VTubeStudio and Live2D. Creating models for 2D and 3D has no comparison in complexity, it's not even the same art form.

Disregard everything I said above if you work for Riot Games, Fortiche, Disney/Pixar DreamWorks, or Sony Pictures Animation.

Q: Why Unity over Unreal Engine 5?

Unity installation size is smaller. It is aimed at hobbyists. You can just write a C# script and drop it onto an object to add new behavior. While the UX can be all over the place, it's frictionless in core aspects.

Unity beats UE5 on ease of use and iteration time. The main reason to switch to UE5 would be a metahuman (!), virtual production, or industry-standard mocap.

Q: How smart is it?

Depends on the LLM model. The default `gemma:2b-instruct` is tiny (3 billion parameters). It can create coherent sentences, but that's how far it can mostly go. If you can use a state-of-the-art 7B model (even with quantization), or something bigger, go for it. You can always swap it for ChatGPT too. Or use a multi-GPU setup. Or, run Unity on a mobile phone, TTS on a Raspberry PI, and have full VRAM for LLM.

Q: Does it support special animations like blushing, smiling, etc.?

I've not added this. It would require special cases added to the 3D model. E.g. it might be hard to animate the mouth during the lipsync. Blushing with 3D avatars is usually done by blending special texture in a shader graph.

Yet the basic tech is already there. If you want to detect emotions in text, you can use LLM for sentiment analysis. I've also added the tech to trigger the events using WebSocket. ATM it's starting a particle effect. Half of the C# code deals with triggering shape keys. Blinking is a function called every few seconds. Once you create an interaction on the 3D model, you can start it at any time. It's just time-consuming to create.

Q: Have you tried to apply AI modification to the rendered frame? Like img2img transformation before showing it to the user?

Yes, I tried (not added to this repo). The original plan was to style transfer the rendered frame to a stable diffusion-generated image. From my quick experiments, besides performance problems, the simplest solutions do not have the necessary quality or temporal stability.

We do not have a performance budget to run VGG16/19. This excludes the 'original' techniques like "A Neural Algorithm of Artistic Style" [Gatys2015] or "Perceptual Losses for Real-Time Style Transfer and Super-Resolution" [Johnson2016]. None of them also looked at flickering. They were designed only for static images and not videos. There were further works that looked into that problem: [Jamriska2019], [Texler2020].

I know Unity also tried real-time style transfer in 2020: "Real-time style transfer in Unity using deep neural networks".

Afterward, we are in transformers territory (surprise!). Last year, "Data AugmenTation with diffUsion Models (DATUM)" [CVPR-W 2023] used diffusion (again, surprise!). There is a paperswithcode category called Synthetic-to-Real Translation if you want to track state of the art.

At this point, I've decided that it was a feature creep to try to fit this into the app.

> There was a Two Minute Papers episode that looked into similar techniques: "Intel's Video Game Looks Like Reality!". Based on Intel's "Enhancing Photorealism Enhancement" [Richter2021].

Q: Is the 3D model included?

Yes, check .fbx inside unity-project/Assets/Sintel.

Q: What's with the name?

All my projects have utilitarian names. This time, I wanted something more distinct. Iris is a purple-blue flower. Iris is a part of the eye. Seemed fitting? Especially since eyes and hair are **the** problems in CG characters.

Honourable mentions

3D

- Blender, Blender Institute.
- GIMP and Inkscape. I'm so used to them I almost forgot to mention.
- Sintel Lite 2.57b by BenDansie. Used as base mesh.
- 21 Realtime man Hairstyles collection by Vincent Page.
- mixamo for animations.
- Tons of animation guides. I especially liked videos by Sir Wade Neistadt on YouTube:
  - Animating Eyes: Character Blinks,
  - The Secret Workflow for Animating Dialogue.

LLM

- Google's Gemma.
- Ollama.

TTS

- coqui-ai's TTS.
- erew123, S95Sedan, and Danil Boldyrev who have pre-compiled DeepSpeed for Windows.

Unity

- Oculus Lipsync for Unity.
- NativeWebSocket package.
- Springs Bones script by Unity Technologies Japan.
- A few Unity samples e.g. for hair shader.