AI & LLMS

Local Video Avatar Generator

Build a local AI video avatar with Ollama and Wav2Lip — a complete tutorial for offline talking-head generation with no cloud dependency.

Dr. Farshid Pirahansiah · pirahansiah.com

Building a Local Video Avatar Generator Using Ollama and Open-Source Tools

Creating a video avatar generator completely locally without cloud services or API keys is challenging but possible. Here's a step-by-step guide to build a system that generates talking video avatars using locally-run models.

Prerequisites

- A computer with decent GPU (at least 8GB VRAM recommended)
- 16GB+ RAM
- 50GB+ free storage space
- Linux or macOS (Windows with WSL also works)
- Basic familiarity with command line

Step 1: Set Up Ollama for Local LLM

Ollama allows you to run large language models locally for text generation.

1. Install Ollama:

```
# For macOS/Linux
curl -fsSL https://ollama.com/install.sh | sh

# For Windows (via WSL)
# First install WSL, then run the Linux command above
```

2. Pull a suitable model (Llama3 recommended for better performance):

```
ollama pull llama3
```

3. Test your Ollama installation:

```
ollama run llama3 "Write a short 30-second script about climate change"
```

Step 2: Install Local Text-to-Speech Engine

We'll use Piper, a fast local TTS system:

1. Install dependencies:

```
sudo apt-get update
sudo apt-get install -y build-essential python3-pip python3-venv
```

2. Set up a Python virtual environment:

```
python3 -m venv ~/venv-tts
source ~/venv-tts/bin/activate
```

3. Install Piper:

```
pip install piper-tts
```

4. Download a voice model:

```
mkdir -p ~/.local/share/piper-tts/voices
cd ~/.local/share/piper-tts/voices
wget https://huggingface.co/rhasspy/piper-
voices/resolve/main/en/en_US/lessac/medium/en_US-lessac-medium.onnx
wget https://huggingface.co/rhasspy/piper-
voices/resolve/main/en/en_US/lessac/medium/en_US-lessac-medium.onnx.json
```

5. Test Piper:

```
echo "This is a test of the text to speech system." | piper \
  --model ~/.local/share/piper-tts/voices/en_US-lessac-medium.onnx \
  --output-raw | aplay -r 22050 -f S16_LE -c 1
```

Step 3: Install Wav2Lip for Avatar Animation

1. Clone the repository:

```
git clone https://github.com/Rudrabha/Wav2Lip.git
cd Wav2Lip
```

2. Set up environment:

```
python3 -m venv ~/venv-wav2lip
source ~/venv-wav2lip/bin/activate
pip install -r requirements.txt
```

3. Download the pre-trained model:

```
mkdir -p checkpoints
wget -O checkpoints/wav2lip.pth
https://github.com/Rudrabha/Wav2Lip/releases/download/v1.0/wav2lip.pth
```

4. Prepare a reference face image or video clip to be animated (save as “face.jpg” or “face.mp4”)

Step 4: Install FFmpeg for Video Processing

```
sudo apt-get install -y ffmpeg
```

Step 5: Create the Pipeline Script

Create a file called `avatar_generator.py`:

```python
#!/usr/bin/env python3
import os
import subprocess
import argparse
import tempfile
import json

def generate_script(prompt):
    """Generate script text using Ollama"""
    print("Generating script with Ollama...")
    cmd = f'ollama run llama3 "{prompt}"'
    result = subprocess.run(cmd, shell=True, capture_output=True, text=True)
    return result.stdout.strip()

def text_to_speech(text, output_wav):
    """Convert text to speech using Piper"""
    print("Converting text to speech...")
    voice_model = os.path.expanduser("~/.local/share/piper-tts/voices/en_US-lessac-medium.onnx")
    with tempfile.NamedTemporaryFile(mode='w', suffix='.txt') as f:
        f.write(text)
        f.flush()
        cmd = f'cat {f.name} | piper --model {voice_model} --output_file {output_wav}'
        subprocess.run(cmd, shell=True)

def animate_avatar(face_path, audio_path, output_video):
    """Animate the avatar using Wav2Lip"""
    print("Animating avatar...")
    wav2lip_path = os.path.expanduser("~/Wav2Lip")
    os.chdir(wav2lip_path)
    cmd = f'python inference.py --checkpoint_path checkpoints/wav2lip.pth --face {face_path} --audio {audio_path} --outfile {output_video} --nosmooth'
    subprocess.run(cmd, shell=True)

def main():
    parser = argparse.ArgumentParser(description='Generate a talking avatar video locally')
    parser.add_argument('--prompt', required=True, help='Prompt for script generation')
    parser.add_argument('--face', required=True, help='Path to face image or video')
    parser.add_argument('--output', default='output.mp4', help='Output video path')
    args = parser.parse_args()

    # Create temporary directory for intermediate files
```

```
    with tempfile.TemporaryDirectory() as tmpdir:
        script_file = os.path.join(tmpdir, 'script.txt')
        audio_file = os.path.join(tmpdir, 'speech.wav')

        # Generate script
        script = generate_script(args.prompt)
        with open(script_file, 'w') as f:
            f.write(script)
        print(f"Generated script:\n{script}\n")

        # Convert script to speech
        text_to_speech(script, audio_file)

        # Animate avatar
        animate_avatar(args.face, audio_file, args.output)

        print(f"Video generated and saved to {args.output}")

if __name__ == "__main__":
    main()
```

Make the script executable:

```
chmod +x avatar_generator.py
```

Step 6: Run the Avatar Generator

1. Prepare a face image or short video clip of the avatar you want to animate
2. Run the script:

```
./avatar_generator.py --prompt "Write a short introduction about renewable energy" --face face.jpg --output avatar_video.mp4
```

Step 7: Add Optional Captions (Using Local Whisper)

1. Install the local version of Whisper:

```
python3 -m venv ~/venv-whisper
source ~/venv-whisper/bin/activate
pip install openai-whisper
```

2. Create a caption script:

```python
#!/usr/bin/env python3
import whisper
import subprocess
import os
import argparse

def generate_captions(audio_file):
    """Generate captions using Whisper"""
    print("Generating captions...")
    model = whisper.load_model("base")
    result = model.transcribe(audio_file)
    return result["text"]

def add_captions_to_video(video_file, caption_text, output_file):
    """Add captions to video using FFmpeg"""
    print("Adding captions to video...")
    with open("captions.srt", "w") as f:
        f.write("1\n00:00:00,000 --> 00:05:00,000\n" + caption_text)

    cmd = f'ffmpeg -i {video_file} -vf subtitles=captions.srt {output_file}'
    subprocess.run(cmd, shell=True)
    os.remove("captions.srt")

def main():
    parser = argparse.ArgumentParser(description='Add captions to a video')
    parser.add_argument('--video', required=True, help='Input video file')
    parser.add_argument('--audio', required=True, help='Input audio file for
transcription')
    parser.add_argument('--output', default='captioned_video.mp4', help='Output video
path')
    args = parser.parse_args()

    captions = generate_captions(args.audio)
    add_captions_to_video(args.video, captions, args.output)
    print(f"Captioned video saved to {args.output}")

if __name__ == "__main__":
    main()
```

Troubleshooting Tips

1. **Memory Issues**: If you encounter memory errors with Ollama or Wav2Lip, try using a smaller model or reducing batch sizes.
2. **GPU Problems**: Some models require CUDA. Make sure your GPU drivers are properly installed:

```
# Check CUDA installation
nvidia-smi
```

3. **Video Quality**: For better results:
   - Use high-quality reference faces
   - Ensure good lighting in the reference image
   - Try different models or parameters in Wav2Lip
4. **Integration Issues**: If components don't work together, ensure all paths are correctly specified in the scripts.

Conclusion

You now have a completely local video avatar generator pipeline that: - Generates script text using a local LLM (Ollama) - Converts text to speech using Piper - Animates a face to match the speech using Wav2Lip - Optionally adds captions using Whisper

All components run locally without any cloud services, online dependencies, or API keys. This approach gives you full control over the process and protects your privacy, though it requires more computational resources than cloud-based alternatives.