Voice Cloning with F5-TTS: Fine-Tuning and Production Deployment

Author: Yirmeyahu Mwangelwa
Date: January 2026
Technologies: F5-TTS, PyTorch, RunPod Serverless, Docker, CUDA

Executive Summary

This project involved fine-tuning the F5-TTS (Flow-matching Text-to-Speech) model on custom voice data and deploying it as a production-ready serverless API. The goal was to create a cost-effective alternative to commercial TTS services (such as ElevenLabs) while maintaining high-quality voice cloning capabilities.

Key Achievements: - Successfully fine-tuned F5-TTS on ~2 hours of custom voice recordings - Deployed a serverless inference endpoint on RunPod with GPU acceleration - Achieved natural-sounding voice synthesis with optimized inference parameters - Built a complete production API with multiple output formats and configurable settings

---

Project Overview

Objective

Replace commercial TTS services with a self-hosted voice cloning solution that: 1. Produces natural-sounding narration matching a specific voice 2. Operates cost-effectively on a pay-per-use serverless model 3. Provides low-latency inference for integration with other applications

Technology Selection

F5-TTS was selected for its: - State-of-the-art voice cloning capabilities - Flow-matching architecture for high-quality audio generation - Support for fine-tuning on custom voice data - Open-source availability

---

Technical Implementation

Phase 1: Data Preparation

Dataset Creation: - Collected approximately 2 hours of high-quality voice recordings - Performed audio preprocessing: - Noise reduction and normalization - Segmentation into training clips (5-15 seconds each) - Transcription alignment with audio segments - Created metadata files mapping audio clips to transcriptions

Data Format:

```
dataset_v3/
├── clips/
│   ├── training_sample_1_clean_clip_0000.wav
│   ├── training_sample_1_clean_clip_0001.wav
│   └── ...
└── metadata.csv
```

Phase 2: Model Fine-Tuning

Training Configuration: - Base model: F5-TTS pretrained checkpoint - Training hardware: RunPod GPU instance (RTX 4090) - Training iterations: Multiple epochs until convergence - Model versioning: Iterative improvements (v1 → v2 → v3)

Key Training Insights: 1. Quality of training data directly impacts output quality 2. Consistent recording conditions improve model convergence 3. Proper audio preprocessing eliminates artifacts in generated speech

Phase 3: Inference Optimization

Extensive experimentation was conducted to optimize inference parameters:

Parameter	Tested Range	Optimal Value	Impact
`nfe_step`	32-50	46	Quality vs. speed tradeoff; lower values produce “robotic” artifacts
`speed`	0.5-1.0	0.80	Controls speech pace; affects naturalness
`cross_fade_duration`	0.1-0.2	0.1	Blending between audio

Parameter	Tested Range	Optimal Value	Impact
			chunks

Critical Discovery: Reference Audio Requirements

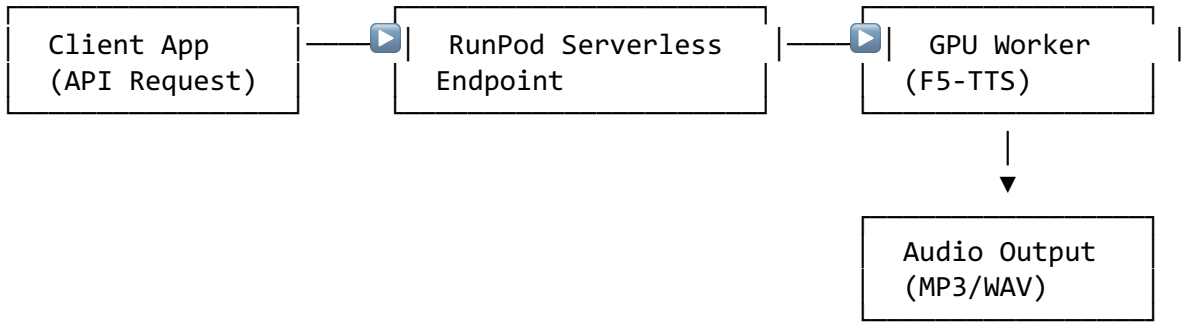
A significant finding during optimization was the importance of reference audio structure for F5-TTS inference:

1. **Problem:** Generated audio contained artifacts where the reference audio "bled" into the output
2. **Root Cause:** F5-TTS is designed to continue from the reference audio context
3. **Solution:** Reference audio must be:
    - A complete sentence/thought (not cut mid-paragraph)
    - Include ~1 second of silence at the end
    - Begin naturally (not mid-sentence)

This discovery eliminated audio artifacts and significantly improved output quality.

Phase 4: Production Deployment

Architecture:



Docker Image Configuration: - Base: `runpod/base:0.6.2-cuda12.1.0` - Model weights baked into image for fast cold starts - Reference audio and metadata included - FFmpeg for audio format conversion

Key Implementation Challenges Solved:

1. **Cross-Platform Build Issues**
    - Problem: Docker images built on ARM (Mac M4) failed on RunPod (AMD64)
    - Solution: Used `docker buildx --platform linux/amd64` for cross-platform builds
2. **Python Environment Conflicts**
    - Problem: `pip install` targeted wrong Python version in container
    - Solution: Used `python3 -m pip install` to ensure correct environment
3. **Module Import Failures**

    - Problem: RunPod module not found despite installation
    - Solution: Added verification step in Dockerfile to catch issues early

API Features: - Multiple output formats: WAV, MP3, M4A, AAC, OGG - Configurable inference parameters - Optional S3-compatible storage integration - Base64 or URL response modes - Audio post-processing (lowpass filter option)

Results

Quality Metrics

- **Voice Similarity:** High fidelity to original voice characteristics
- **Naturalness:** Achieved natural prosody with optimized parameters
- **Artifact Reduction:** Eliminated reference audio bleed through proper clip preparation

Performance Metrics

- **Cold Start:** ~30-60 seconds (model loading)
- **Warm Inference:** ~2-5 seconds for typical narration lengths
- **Cost Efficiency:** Pay-per-use model significantly reduces costs compared to commercial alternatives

Comparison to Commercial Services

Aspect	F5-TTS (Self-Hosted)	Commercial TTS
Voice Customization	Full control via fine-tuning	Limited to provided voices or expensive cloning
Cost	Pay for GPU time only	Per-character pricing
Quality	Comparable after optimization	Consistent but less customizable
Latency	Slightly higher cold start	Generally faster
Data Privacy	Complete control	Data sent to third party

Lessons Learned

Technical Insights

1. **Diffusion Model Behavior:** NFE steps have a “sweet spot” - more steps don’t always mean better quality
2. **Reference Audio Design:** The structure of reference audio is as important as the model itself

3. **Audio Format Impact:** Lossy compression (MP3) can mask minor artifacts present in lossless (WAV) output
4. **Containerization:** Cross-platform Docker builds require explicit platform specification

Best Practices Established

1. Always verify module installation in Docker builds
2. Use complete sentences with natural endings for reference audio
3. Test inference parameters systematically to find optimal values
4. Implement configurable parameters for production flexibility

---

Future Improvements

1. **Model Optimization:** Explore quantization for faster inference
2. **Streaming Support:** Implement streaming audio output for real-time applications
3. **Multi-Voice Support:** Enable switching between multiple fine-tuned voices
4. **Automatic Chunking:** Add server-side text chunking for long narrations

---

Conclusion

This project successfully demonstrated the end-to-end process of fine-tuning a state-of-the-art TTS model and deploying it as a production-ready service. The resulting system provides high-quality voice cloning capabilities at a fraction of the cost of commercial alternatives, with full control over the voice characteristics and inference parameters.

The key technical contribution was the discovery and documentation of reference audio requirements for F5-TTS, which significantly impacts output quality and is not well-documented in existing resources.

Technical Stack

- **ML Framework:** PyTorch, F5-TTS
- **Deployment:** RunPod Serverless, Docker
- **Audio Processing:** FFmpeg, librosa
- **Cloud Storage:** S3-compatible (optional)
- **Languages:** Python

Resume Summary

Voice Cloning System with F5-TTS — Fine-tuned the F5-TTS flow-matching text-to-speech model on custom voice data and deployed as a production serverless API on RunPod. Optimized inference parameters through systematic experimentation, achieving natural-sounding voice synthesis. Solved critical audio artifact issues by discovering undocumented reference audio requirements. Built complete production infrastructure including Docker containerization, multiple audio format support, and Python client library. Technologies: PyTorch, F5-TTS, Docker, RunPod Serverless, CUDA, FFmpeg.