A Practical End-to-End Text-to-Speech (TTS) Pipeline for Single-Speaker Voice Synthesis

System Overview, Usage Guide, and Smoke-Run Verification

Project Owner: Emmanuel Quaye • Implementation Partner: ChatGPT (GPT■5 Thinking)

September 5, 2025

Abstract

This paper documents a compact, end■to■end text■to■speech (TTS) pipeline designed to train a single■speaker neural synthesizer from user-provided audio (≈3 hours of Bible readings) and transcripts. The system includes a PyQt-based recorder/transcriber, a Tacotron■style acoustic model with teacher■forcing, and dual vocoder paths (WaveGlow or Griffin–Lim fallback). We describe the system purpose, dependencies, configuration schema, and provide step■by■step instructions for the main application and a minimal smoke■run that verifies the entire pipeline in minutes.

1. Purpose & Scope

The project's purpose is to enable a technically curious user to build a personal voice model using their own audio. Pragmatically, the aim is not to advance state■of■the■art synthesis quality, but to ensure a transparent pipeline that records, segments, transcribes, aligns with metadata, trains a Tacotron■style model to predict mel spectrograms, and vocalizes text via a robust vocoder path. With limited data (~3 hours), we target intelligibility and speaker likeness rather than perfect prosody. The smoke■run ensures that all integration points (I/O, config, training, checkpointing, and synthesis) function as intended.

2. System Overview

• Front■End (Synthesizer.py): PyQt5 app to record at 44.1 kHz, resample to 22.05 kHz, split into ~4 s segments, and transcribe via Whisper. It maintains `datasets/my_dataset/metadata.csv` and ensures each transcript aligns with a corresponding WAV file.
• Model (tts_model.py): A Tacotron■style model trained with teacher_forcing_ratio=1.0 initially. Loss is MSE over mel frames; gradient clipping and a ReduceLROnPlateau scheduler improve stability.
• Vocoder: Preferred WaveGlow via TorchHub (requires first■time internet); otherwise Griffin–Lim to guarantee audible output offline.
• Debugging (Debugging.py): Mel visualization and sanity checks.
• Configs: `config.json` and `temp_config.json` specify audio params, training hyper■parameters, and dataset paths.

3. Environment & Dependencies

• Python 3.9+; PyTorch (CUDA optional but recommended).
• PyQt5 (GUI) and system multimedia backends.
• PyAudio + PortAudio for recording (Windows: prebuilt wheel; macOS: `brew install portaudio`).
• Whisper model weights (downloaded on first use). WaveGlow weights via TorchHub (first run only).

4. Dataset Preparation

1) Place source audio under `datasets/my_dataset/wavs/` and ensure transcripts in `datasets/my_dataset/metadata.csv`.
2) The GUI recorder automates segmentation and transcription; manual data can be added if filenames

match the metadata IDs.
3) Keep segments semantically coherent (phrases/sentences), trim long silences, and normalize peak levels.

5. Main Program Usage (Synthesizer.py)

1) Launch `Synthesizer.py`.
2) Record a session (44.1 kHz) → the app writes a full WAV, resamples to 22.05 kHz, auto■segments (~4 s), and transcribes each.
3) Review or edit transcripts in the UI. The app updates `metadata.csv` and the segment WAVs.
4) Configure training in the Options tab (epochs, batch size, learning rate). Teacher■forcing is enabled via `teacher_forcing_ratio` in the config (default 1.0).
5) Click **Train**. The system logs loss per batch/epoch, applies gradient clipping, and saves `datasets/output/tacotron2_model.pth`.
6) Use **Synthesize** to generate audio from text. If WaveGlow is unavailable, the system falls back to Griffin–Lim.

6. Smoke■Run Verification (smokerun_tts.py)

The smoke run is a minimal, automated verification that the entire pipeline is wired correctly. Use the provided script `smokerun_tts.py` to:
• Build a tiny subset dataset with 2 valid (WAV+metadata) pairs in `datasets/smoke_subset/`.
• Write `datasets/smoke_subset_config.json` with `epochs=1`, `batch_size=2`, and `teacher_forcing_ratio=1.0`.
• Train for 1 epoch into `datasets/output_smoke/tacotron2_model.pth`.
• Synthesize a short sentence into `datasets/output_smoke/smoke_sample.wav`.

Run examples:
• CPU: `python smokerun_tts.py`
• With CUDA: `python smokerun_tts.py --use_cuda`
• Custom text: `python smokerun_tts.py --text "Let there be light."`
• Larger subset: `python smokerun_tts.py --subset_items 3`

7. Training Details & Configuration

• Teacher Forcing: Critical for convergence. Start at 1.0; optionally schedule down once alignments stabilize.
• Loss: MSE on predicted vs. target mel frames. Ensure trimming to predicted time steps to avoid length mismatch.
• Stability: Gradient clipping (e.g., max_norm=1.0) and LR scheduling (ReduceLROnPlateau) reduce collapse risk.
• Audio Params: 22,050 Hz; n_mels=80; FFT=1024; hop=256 (must be consistent across train/inference).

8. Troubleshooting & Notes

• No audio playback in GUI: QMedia backends may be missing; save WAV and play externally to isolate backend issues.
• WaveGlow download fails: Synthesis still works via Griffin–Lim fallback; quality will be lower but audible.
• Import errors with librosa/NumPy: Use small compatibility shim (`np.complex = complex`, `np.float = float`).
• PyAudio errors at 22.05 kHz: Recording at 44.1 kHz then resampling fixes most device capability issues.

9. Ethics & Responsible Use

Train only on speech you own or have rights to use. Respect voice likeness and consent; clearly label synthetic audio in demonstrations.

Appendix A: Configuration Snapshot

Config: temp_config.json
epochs: 150
batch_size: 50
learning_rate: 0.0015
teacher_forcing_ratio: 1.0
dataset_path: datasets/my_dataset/
audio.sample_rate: 22050
audio.num_mels: 80
audio.fft_size: 1024
audio.hop_length: 256
Config: config.json
epochs: 150
batch_size: 50
learning_rate: 0.0024999999999999999
teacher_forcing_ratio: 1.0
dataset_path: datasets/my_dataset/
audio.sample_rate: 22050
audio.num_mels: 80
audio.fft_size: 1024
audio.hop_length: 256

Appendix B: Public API Signatures

API Signatures (from tts_model.py)
- train_model(config_path, output_path, progress_callback=None)
- synthesize(text, output_file, model_path, config_path=None, use_cuda=True)