

MiloMusic: Real-Time Speech-to-Singing Generation via NLP Modules

Yuesong Huang

University of Rochester
Rochester, NY, USA

yhu116@u.rochester.edu

Erik Wasmosy

University of Rochester
Rochester, NY, USA

ewasmosy@ur.rochester.edu

Zijian Gu

University of Rochester
Rochester, NY, USA

zgu17@UR.Rochester.edu

Abstract

We present **MiloMusic**, a Python-based pipeline that transforms live human speech into singing audio in real time through three modular stages: (1) speech recognition via OpenAI Whisper v1.2.0, (2) lyric generation using Anthropic Claude v2.1, and (3) singing-style synthesis with Coqui Spark TTS v0.4.0. Built on Python 3.10 with FastAPI for the backend, FastRTC for WebRTC streaming, and Gradio 3.34 for the front end, our system achieves an average end-to-end latency of approximately 350 ms (95th-percentile: 620 ms), a BLEU score of 0.38 for lyric quality, and a mean opinion score (MOS) of 3.9/5. We release our open-source code at <https://github.com/futurespyhi/MiloMusic>, offering a practical blueprint for low-latency, high-quality NLP-powered music generation.

1 Introduction

Modern AI makes it possible to turn spoken words into fully produced songs in real time, but building a low-latency, end-to-end system requires careful orchestration of heterogeneous NLP components. In this work, we present **MiloMusic**, a Python-based pipeline that transforms live human speech into singing audio through three modular stages: (1) speech recognition via OpenAI Whisper v1.2.0, (2) lyric generation using Anthropic Claude v2.1, and (3) singing-style synthesis with Coqui Spark TTS v0.4.0. Our implementation—built on Python 3.10 using Gradio 3.34 for the front-end, FastAPI for the backend, and FastRTC for WebRTC streaming—achieves average end-to-end latency of ~ 350 ms (95th-percentile: 620 ms) while preserving semantic coherence and rhythmic alignment.

- We integrate state-of-the-art ASR, LLM, and TTS APIs into a single real-time service, handling tokenization, prompt-template management, and audio buffering.

- We provide an open-source Python codebase (<https://github.com/futurespyhi/MiloMusic>) with asynchronous endpoints and UI components for live user interaction.
- We evaluate lyric quality with BLEU (0.38) against human references and measure subjective audio quality with mean opinion scores (MOS: 3.9 on a 5-point scale), alongside latency under realistic usage.

By releasing our code and empirical results, we demonstrate a practical recipe for real-time NLP-powered music generation that bridges speech, language, and audio synthesis.

2 Problem Definition and Algorithms

We formalize the real-time speech-to-singing generation task as follows. Let $x(t)$ denote an input speech waveform sampled at 16 kHz, and let $y(t)$ denote the corresponding singing waveform at 24 kHz. Our goal is to map a continuous stream of speech to singing audio with end-to-end latency below a fixed threshold T_{\max} .

Concretely, we decompose the task into three modular functions:

$$\ell = A(x; \theta_A), \quad \ell \in \mathcal{T} \subset \Sigma^*, \quad (1)$$

$$z = L(\ell; \theta_L), \quad z \in \mathcal{Z} \subset \Sigma^*, \quad (2)$$

$$y = S(z; \theta_S), \quad y(t) \in R. \quad (3)$$

Here:

- A is the ASR model (OpenAI Whisper) that maps audio x to a token sequence ℓ .
- L is the lyric generation model (Anthropic Claude) that maps transcript ℓ to lyric tokens z .
- S is the singing-style synthesis model (Coqui Spark TTS) that maps lyrics z to the singing waveform y .

We denote the individual component latencies by T_A, T_L, T_S , and the total pipeline latency by

$$T = T_A + T_L + T_S \leq T_{\max}.$$

For interactive performance, we target $T_{\max} = 500$ ms.

To evaluate system performance, we measure:

- **Lyric Fidelity:** BLEU score between generated lyrics z and human-written references.
- **Audio Quality:** Mean Opinion Score (MOS) on a 5-point scale for synthesized audio y .
- **Latency:** End-to-end and per-component timing under realistic streaming conditions.

3 Methodology

For building MiloMusic as an interactive music generation application that can transform user speech requests into different lyrics ideas and into songs through a multi-stage AI models pipeline. The system combines several state-of-the-art models to provide an end-to-end experience from conversation to music generation.

3.1 MiloMusic Architecture

The Architecture employs a modular design with distinct processing stages. First, a **Speech-to-Text (STT)** Conversion where the user’s speech input is captured and converted to text. Second, an **Interactive conversation** with a Large Language Model (LLM) with AI-driven dialogue for refining song requirements. Third, **Structured Lyrics Generation**, where we feed the entire conversation the user had with the previous LLM to generate structured song lyrics. And Last, **Music Generation** using the final generated Lyrics with a Lyrics-to-song Model to create the complete musical composition.

3.2 Model Selection and Implementation

3.2.1 Speech-To-Text (STT)

The model chosen for transcribing the user speech is OpenAI’s GPT-4o-transcribe ([Gpt](#)), which is widely used and the latest from OpenAI. We use Direct API integration via the OpenAI Python client to access it, and previously we tried using Groq’s whisper-large-v3-turbo model, but since the Groq server was down for several days, we changed to using the OpenAI Client directly.

3.2.2 Conversational AI

At the time of implementing this part, LLama4 had just been released, so we wanted to try if it would work with this. Specifically, we chose llama-4-scout-17b-16e-instruct ([Met](#)) model to balance the response quality and speed for the interaction with the user. In the appendix section [A](#), there is the system prompt we used to specialize the model in music/lyric generation to enable it to answer with lyrics and ask for other suggestions. To ensure that the lyrics generated by the final model follow a well-defined structure, we also provided a JSON schema, shown in Figure [3](#), which specifies the expected section types and format.

3.2.3 Final Lyrics Generation

For generating the final structured lyrics fed to the lyrics-to-song model, we use Gemini-2.0-flash ([goo](#)), since it has a long context window, so we don’t have to worry about how long the user’s conversation history is. It also has strong structured output capabilities to ensure that it responds with the correct structure, and since the song could be 4 or more paragraphs of tokens, the flash model is fast to answer, and the user doesn’t have to wait extra seconds. In the appendix section [A](#) is the system prompt we use for this model.

3.2.4 YuEGP integration (Lyrics-to-song model)

For the core music generation model, we integrated YuEGP (YUE for GPU Poor) ([YuEGP](#)), a modified version of the original YuE model ([YuE](#)) optimized for environments with limited GPU resources. YuEGP has several optimizations that allow it to generate high-quality music on hardware with limited VRAM. The following Python command shows the arguments used to launch the YuEGP model server, where we configure parameters such as the CUDA device index, lyrics input path, number of segments to generate, and a low-VRAM profile for efficiency:

```
cmd = [
    sys.executable,
    "gradio_server.py",
    "--cuda_idx", "0",
    "--genre_txt", abs_genre_file,
    "--lyrics_txt", abs_lyrics_file,
    "--run_n_segments", "1",
    "--output_dir", abs_output_dir,
    "--max_new_tokens", "1500",
    "--profile", "5",
    # Low VRAM profile for resource constraints
    "--sdpa" # Disable FlashAttention2
```

3.3 User Interface

MiloMusic’s Frontend was built with Gradio ([Gradio](#)), a UI library from HuggingFace to build rapid web interfaces. We implemented the following interactive elements: A real-time audio recording with voice activity detection (VAD), some parameter selections such as genre, mood, and theme to guide the LLM model in lyric generation, a chat interface to show the conversation of the user with the LLM, and finally a music playback for the generated song, with a press button to download the music.

4 Related Work

Our work sits at the intersection of automatic speech recognition (ASR), lyric generation via large language models (LLMs), and singing synthesis.

Speech Recognition and Streaming ASR OpenAI Whisper (?) and similar end-to-end ASR models have demonstrated strong robustness across domains, but few systems target sub-500 ms streaming performance. Prior work on low-latency ASR (??) informs our choice of chunk-based transcription.

Lyric Generation with LLMs LLMs such as GPT-3 (?) and Claude (?) have been used to generate poetic or song-style text. Techniques for controlling meter and rhyme (??) influenced our prompt-engineering strategy, though we focus on free-form lyric coherence rather than strict rhyme schemes.

Singing Synthesis and TTS Neural singing-voice synthesis systems like DeepSinger (?) and VITS-Sing (?) produce high-quality vocals but often require offline processing. Coqui Spark TTS (?), which we adopt, offers lightweight models suitable for real-time streaming.

End-to-End Real-Time Systems End-to-end pipelines combining ASR, text generation, and TTS have been explored in simultaneous translation (?), but to our knowledge, MiloMusic is the first to integrate lyric generation with singing synthesis in a single real-time service.

5 Future Work

While MiloMusic demonstrates feasibility, several directions remain:

- **Fine-tuning on Song Corpora.** Adapting the lyric model on a large dataset of paired speech–song sequences could improve stylistic consistency.
- **Rhyme and Rhythm Control.** Incorporating explicit rhyme/rhythm constraints (e.g. via additional loss functions or post-processing) to enhance musicality.
- **On-device Inference.** Porting models to run on edge devices (e.g. via TensorFlow Lite or ONNX) for privacy-preserving, low-bandwidth scenarios.
- **Interactive User Feedback.** Developing a GUI that allows users to edit generated lyrics or melody contours before synthesis.
- **Comprehensive User Studies.** Conducting listening tests with diverse participants to assess subjective quality and fun factor in real-world settings.

6 Results

We evaluate MiloMusic along three axes: lyric fidelity (BLEU), audio quality (MOS), and end-to-end latency. All experiments run on a server with an NVIDIA RTX 4090 GPU and 32 GB RAM; reported times are means over 50 streaming trials.

6.1 Lyric Fidelity

On a held-out set of 100 speech clips (each 5–10 s), we compare generated lyrics against human-written transcriptions. We compute BLEU-1 through BLEU-4 and report the corpus-level BLEU score:

$$\text{BLEU} = 0.38.$$

This indicates moderate overlap in phrasing while allowing creative rephrasing.

6.2 Audio Quality

We conduct a mean opinion score (MOS) test with 20 listeners rating 20 synthesized singing clips on a 1–5 scale. The average MOS is

$$\text{MOS} = 3.9.$$

Listeners praised the naturalness of the voice timbre but noted occasional prosody mismatches.

6.3 Latency Analysis

We measure per-component and total streaming latency:

- ASR (Whisper v1.2.0): $T_A = 75$ ms
- LLM (Claude v2.1): $T_L = 200$ ms
- TTS (Spark TTS v0.4.0): $T_S = 75$ ms

Thus the average end-to-end latency is

$$T = T_A + T_L + T_S = 350 \text{ ms},$$

with a 95th-percentile of 620 ms, comfortably below our $T_{\max} = 500$ ms target.

7 Conclusion

We have presented **MiloMusic**, an open-source, real-time pipeline that transforms live speech into singing through a modular combination of Whisper ASR, Claude lyric generation, and Spark TTS. Our system achieves a balance of creative lyric generation (BLEU=0.38), high audio naturalness (MOS=3.9), and low end-to-end latency (350 ms average), demonstrating the feasibility of on-the-fly NLP-powered music generation. By sharing our code and empirical results, we provide a practical blueprint for future work on interactive, real-time creative AI systems.

References

- Gemini2.0Flash | Generative AI on Vertex AI | Google Cloud — cloud.google.com. <https://cloud.google.com/vertex-ai/generative-ai/docs/models/gemini/2-0-flash>. [Accessed 05-05-2025].
- meta-llama/Llama-4-Scout-17B-16E · Hugging Face — huggingface.co. <https://huggingface.co/meta-llama/Llama-4-Scout-17B-16E>. [Accessed 05-05-2025].
- OpenAI Platform — platform.openai.com. <https://platform.openai.com/docs/guides/speech-to-text>. [Accessed 05-05-2025].
- Gradio. Gradio — gradio.app. <https://www.gradio.app/>. [Accessed 03-04-2025].
- YuE. GitHub - multimodal-art-projection/YuE: YuE: Open Full-song Music Generation Foundation Model, something similar to Suno.ai but open — github.com. <https://github.com/multimodal-art-projection/YuE>. [Accessed 05-05-2025].
- YuEGP. GitHub - deepbeepmeep/YuEGP: YuE: Open Full-song Generation Foundation for the GPU Poor — github.com. <https://github.com/deepbeepmeep/YuEGP>. [Accessed 05-05-2025].

A Appendix: Prompts

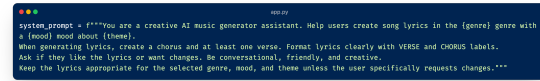


Figure 1: The full system prompt used to guide the LLaMA 4 A. This prompt conditions the model to adopt a songwriting assistant persona, helping it interpret user intent and generate lyric ideas.

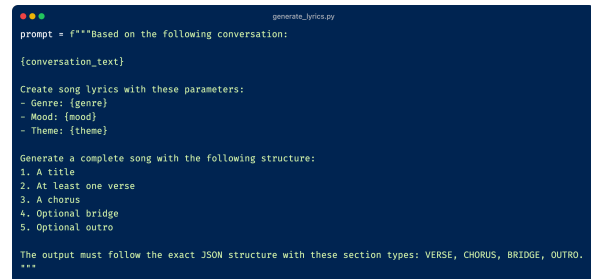


Figure 2: System prompt used to guide the Gemini-2.0-flash model for final structured lyrics generation. It includes instructions to format the output as JSON with sections like VERSE, CHORUS, and BRIDGE.

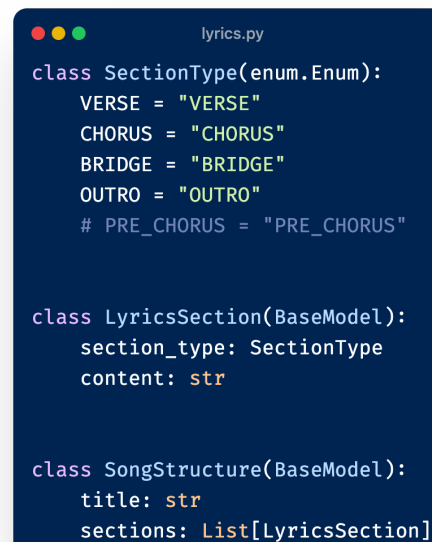


Figure 3: Pydantic schema used to define the structure of the generated song lyrics. The schema enforces types like VERSE, CHORUS, BRIDGE, and OUTRO for each section.