VITA-Audio Reference Audio System: Complete Deep Dive Analysis

Executive Summary

This comprehensive analysis answers the critical question: **What is "Your Voice: <|audio|>" in VITA-Audio's system message, and how does the reference audio system work?**

Key Discoveries

1. **"Your Voice: <|audio|>" is a system message that provides reference audio to VITA-Audio for voice cloning**
2. **VITA-Audio uses a dual audio system**: Reference audio (for voice characteristics) + Input audio (for content)
3. **Reference audio is tokenized and embedded in the system prompt** to guide voice synthesis
4. **Four different implementations** show varying levels of reference audio support

Table of Contents

Core Analysis

Technical Deep Dive

Visual Guides

The "Your Voice" System Message Explained

The Critical Discovery

From `zen-vita-audio/tools/inference_sts.py`, we found this crucial code:

```python
if prompt_audio_path is not None:
    system_message = [
        {
            "role": "system",
            "content": f"Your Voice: <|audio|>\n",
        },
    ]
```

What This Means

The "Your Voice: <|audio|>" is a system message that tells VITA-Audio: "Use this audio as a reference for the voice characteristics you should use in your response."

Simple Explanation

Think of it like showing someone a photo and saying "Make me look like this person." The reference audio is the "photo" that tells VITA-Audio what voice to use.

Technical Explanation

The system message provides a voice template that influences the model's audio generation process, enabling zero-shot voice cloning capabilities.

How the Reference Audio Gets Processed

Step 1: Audio Tokenization

```
 if prompt_audio_path is not None and
self.audio_tokenizer.apply_to_role("user", is_discrete=True):
    # discrete codec
    audio_tokens = self.audio_tokenizer.encode(prompt_audio_path)
    audio_tokens = "".join(f"<|audio_{i}|>" for i in audio_tokens)
```

Process: 1. Reference audio file → Audio tokenizer 2. Continuous audio → Discrete tokens (e.g., [1, 45, 123, 67, ...]) 3. Tokens → Formatted string: `<|audio_1|><|audio_45|><|audio_123|><|audio_67|>...`

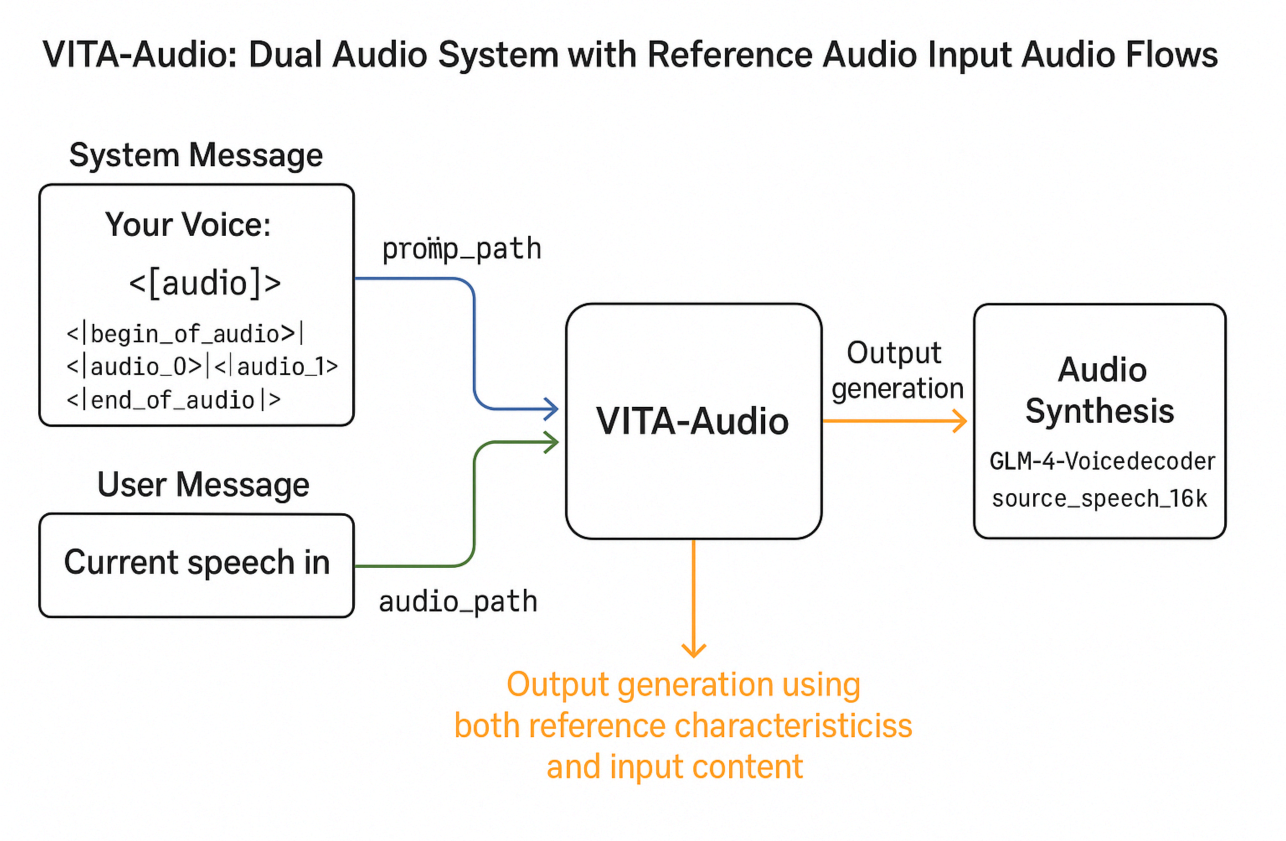
Step 2: System Message Construction

```
 system_message[-1]["content"] = system_message[-1]["content"].replace(
    "<|audio|>", f"<|begin_of_audio|>{audio_tokens}<|end_of_audio|>"
)
```

Result:

```
 "Your Voice: <|begin_of_audio|><|audio_1|><|audio_45|><|audio_123|>
<|audio_67|>...<|end_of_audio|>\n"
```

Dual Audio System Architecture



The Two Audio Streams

VITA-Audio processes **two separate audio streams** simultaneously:

1. Reference Audio Stream (`prompt_audio_path`)

- **Purpose**: Defines target voice characteristics
- **Source**: Pre-recorded voice samples, user uploads, or asset files
- **Processing**: Tokenized and embedded in system message
- **Role**: Voice cloning template

2. Input Audio Stream (`audio_path`)

- **Purpose**: Contains user's current speech content
- **Source**: Microphone recording or file upload
- **Processing**: Tokenized and embedded in user message

- **Role**: Conversation content and context

Complete Message Structure

```
 # System message with reference audio
system_message = {
    "role": "system",
    "content": "Your Voice: <|begin_of_audio|>[reference_audio_tokens]
<|end_of_audio|>\n"
}

# User message with input audio
user_message = {
    "role": "user",
    "content": "[text_message]\n<|audio|>"  # <|audio|> replaced with input
audio tokens
}
```

Audio Synthesis Integration

The reference audio influences the final synthesis through two mechanisms:

1. System Message Influence

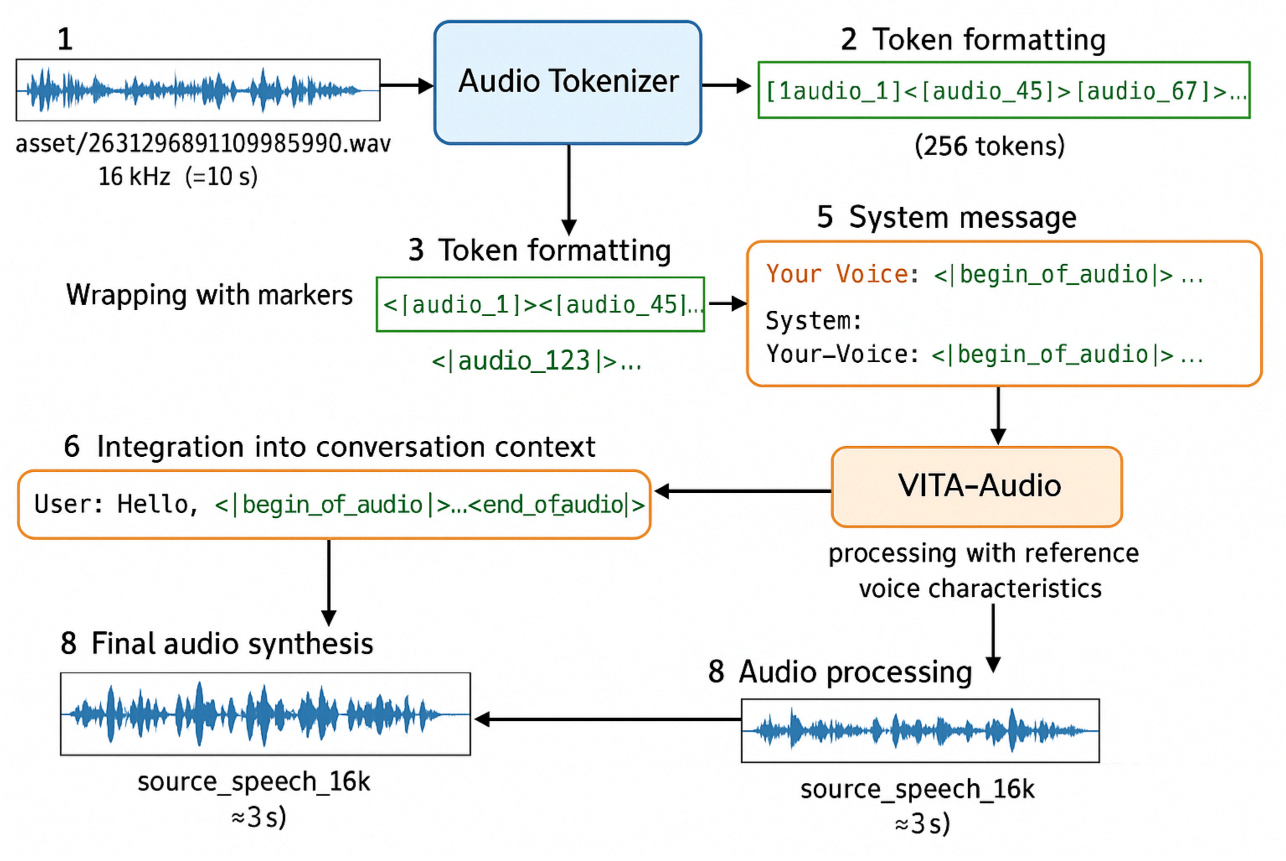
- Reference audio tokens in system message guide the model's understanding of target voice
- Model learns to associate response generation with specific voice characteristics

2. Decoder Parameter Influence

```
 tts_speech = audio_tokenizer.decode(
    audio_tokens,
    source_speech_16k=prompt_audio_path,  # Reference audio for voice cloning
    option_steps=option_steps,
)
```

The `source_speech_16k` parameter provides direct voice characteristics to the GLM-4-Voice decoder.

Reference Audio Tokenization Process


1
asset/2631296891109985990.wav
16 kHz (=10 s)
Audio Tokenizer
2 Token formatting
[1audio_1]<[audio_45]>[audio_67]>...
(256 tokens)
3 Token formatting
Wrapping with markers
<|audio_1]><[audio_45|...
<|audio_123|>...
5 System message
Your Voice: <|begin_of_audio|> ...
System:
Your–Voice: <|begin_of_audio|> ...
6 Integration into conversation context
User: Hello, <|begin_of_audio|>...<end_ofaudio|>
VITA–Audio
processing with reference voice characteristics
8 Final audio synthesis
source_speech_16k
≈3 s)
8 Audio processing
source_speech_16k
≈3 s)

Step-by-Step Process

Step 1: Audio File Input

```
Reference Audio: asset/2631296891109983590.wav
- Format: 16kHz WAV file
- Duration: ~10 seconds
- Content: Voice sample for cloning
```

Step 2: Audio Tokenization

```
audio_tokens = self.audio_tokenizer.encode(prompt_audio_path)
# Result: [1, 45, 123, 67, 89, 234, ...]  # ~256 tokens for 10 seconds
```

Technical Details: - **Tokenizer**: SenseVoice or GLM4Voice tokenizer - **Rate**: ~12.5 tokens per second of audio - **Output**: Discrete integer tokens representing audio features

Step 3: Token Formatting

```
audio_tokens = "".join(f"<|audio_{i}|>" for i in audio_tokens)
# Result: "<|audio_1|><|audio_45|><|audio_123|><|audio_67|><|audio_89|>
<|audio_234|>..."
```

Step 4: Marker Wrapping

```
formatted_tokens = f"<|begin_of_audio|>{audio_tokens}<|end_of_audio|>"
# Result: "<|begin_of_audio|><|audio_1|><|audio_45|>...<|end_of_audio|>"
```

Step 5: System Message Integration

```
system_content = f"Your Voice: {formatted_tokens}\n"
# Result: "Your Voice: <|begin_of_audio|><|audio_1|><|audio_45|>...
<|end_of_audio|>\n"
```

Step 6: Model Processing

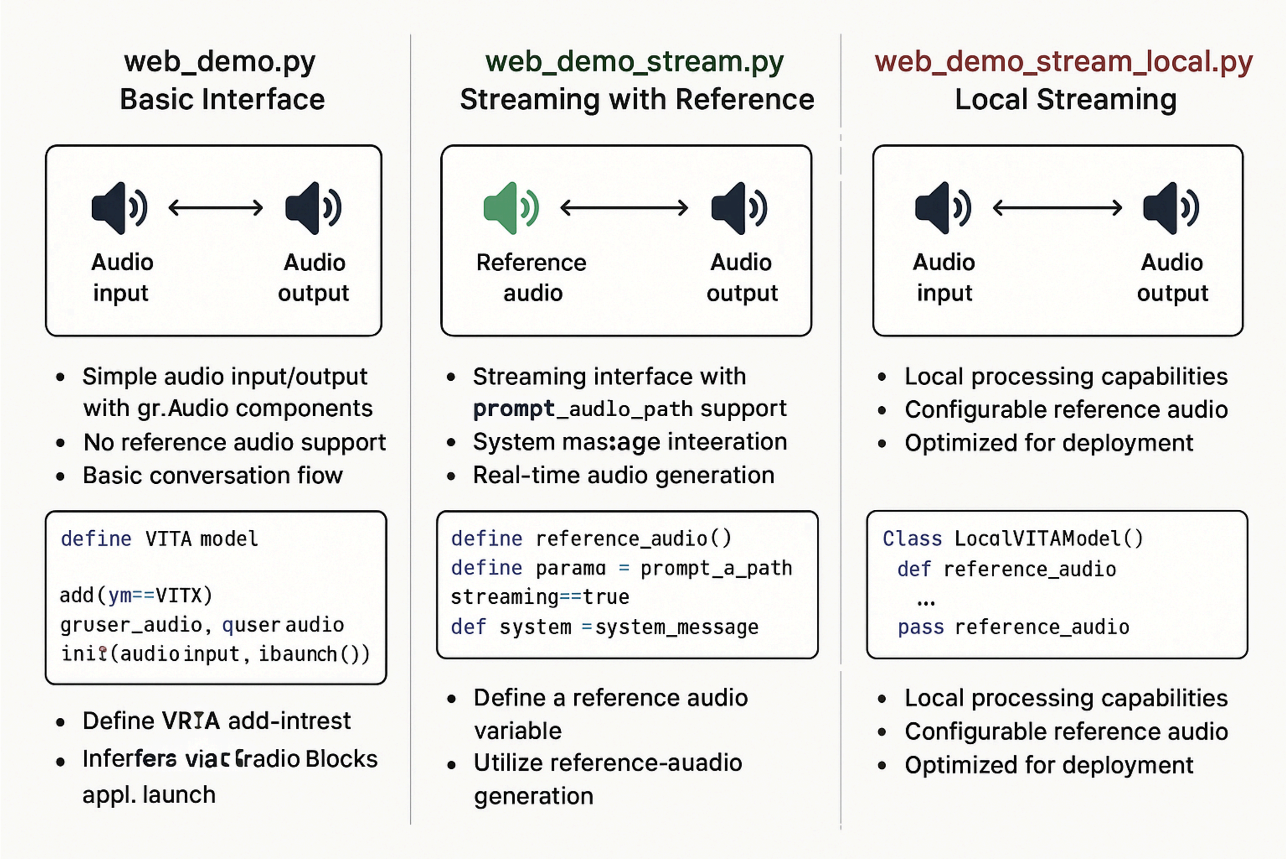
- VITA-Audio processes the system message to understand target voice characteristics
- Reference audio tokens influence response generation and voice synthesis

Step 7: Audio Synthesis

- GLM-4-Voice decoder uses both response tokens and reference audio
- CosyVoice synthesizes final audio with cloned voice characteristics

---

Web Demo Implementations Comparison



web_demo.py
Basic Interface

- Simple audio input/output with gr.Audio components
- No reference audio support
- Basic conversation fiow

```
define VITA model

add(ym==VITX)
gruser_audio, quser audio
init(audio input, ibaunch())
```

- Define VRIA add-intrest
- Inferfers viat Gradio Blocks appl. launch

web_demo_stream.py
Streaming with Reference

- Streaming interface with `prompt_audlo_path` support
- System mas:age inteeration
- Real-time audio generation

```
define reference_audio()
define parama = prompt_a_path
streaming==true
def system =system_message
```

- Define a reference audio variable
- Utilize reference-auadio generation

web_demo_stream_local.py
Local Streaming

- Local processing capabilities
- Configurable reference audio
- Optimized for deployment

```
Class LocalVITAModel()
 def reference_audio
   ...
 pass reference_audio
```

- Local processing capabilities
- Configurable reference audio
- Optimized for deployment

1. web_demo.py - Basic Interface

Features

- **Simple audio input/output** with Gradio Audio components
- **No reference audio support** - uses model's default voice characteristics
- **Basic conversation flow** with text and audio inputs
- **Non-streaming** - complete response generated before playback

Interface Components

```
record_btn = gr.Audio(
    sources=["microphone", "upload"],
    type="filepath",
    label="🎤 Record or Upload Audio",
    show_download_button=True,
    waveform_options=gr.WaveformOptions(sample_rate=16000),
)
audio_output = gr.Audio(
    label="Play", streaming=True, autoplay=True, show_download_button=True
)
```

Limitations

- **No voice cloning**: Cannot specify target voice characteristics
- **Default voice only**: Uses training data distribution for voice selection
- **Batch processing**: No real-time audio generation

2. web_demo_stream.py - Streaming with Reference Audio

Features

- **Reference audio support** via `prompt_audio_path` parameter
- **System message integration** with "Your Voice" prompt
- **Real-time audio generation** with streaming synthesis
- **Voice cloning capabilities** through reference audio

Reference Audio Implementation

```
prompt_audio_path = None  # Can be set to enable voice cloning

if prompt_audio_path is not None:
    if audio_tokenizer.apply_to_role("system", is_discrete=True):
        prompt_audio_tokens = audio_tokenizer.encode(prompt_audio_path)
        prompt_audio_tokens = "".join(f"<|audio_{i}|>" for i in
prompt_audio_tokens)
        system_message = [
            {
                "role": "system",
                "content": f"Your Voice: <|begin_of_audio|>
{prompt_audio_tokens}<|end_of_audio|>\n",
            },
        ]
```

Audio Synthesis with Voice Cloning

```
tts_speech = audio_tokenizer.decode(
    audio_tokens,
    source_speech_16k=prompt_audio_path,  # Reference audio for voice cloning
    option_steps=option_steps,
)
```

Advantages

- **Zero-shot voice cloning**: Can mimic any reference voice
- **Streaming synthesis**: Real-time audio generation
- **High-quality output**: Professional-grade voice cloning

3. web_demo_stream_local.py - Local Streaming

Features

- **Local processing capabilities** - no external API dependencies
- **Configurable reference audio** - `prompt_audio_path` can be set programmatically
- **Optimized for deployment** - reduced network dependencies
- **Same voice cloning logic** as streaming version

Configuration

```
prompt_audio_path = None  # Set to enable voice cloning
```

Local Processing Benefits

- **Privacy**: All processing happens locally
- **Reliability**: No network dependencies
- **Customization**: Easy to modify reference audio programmatically
- **Deployment**: Suitable for edge deployment scenarios

4. inference_sts.py - Command Line Interface

Features

- **Multiple reference audio examples** for testing voice cloning
- **Batch processing** of different voice samples
- **TTS task focus** with voice cloning capabilities

Reference Audio Examples

```
for prompt_audio_path in [
    "asset/2631296891109983590.wav",
    "asset/379838640-d5ff0815-74f8-4738-b0f1-477cfc8dcc2d.wav",
    "asset/4202818730519913143.wav",
]:
    output, tts_speech = s2s_inference.run_infer(
        prompt_audio_path=prompt_audio_path,
        message="Convert the text to speech.\n" + text,
        mode=None,
        do_sample=True,
    )
```

Use Cases

- **Voice cloning testing**: Test different reference voices
- **Batch processing**: Process multiple voice samples
- **Research and development**: Experiment with voice characteristics

---

Code Analysis: inference_sts.py

Key Functions and Classes

S2SInference Class

```
class S2SInference:
    def __init__(self, model_path, audio_tokenizer_path, ...):
        # Initialize model and tokenizers

    def run_infer(self, prompt_audio_path=None, audio_path=None, message="",
mode=None):
        # Main inference function with reference audio support
```

Reference Audio Processing

```
if prompt_audio_path is not None:
    system_message = [
        {
            "role": "system",
            "content": f"Your Voice: <|audio|>\n",
        },
    ]

    if prompt_audio_path is not None and
self.audio_tokenizer.apply_to_role("user", is_discrete=True):
        # discrete codec
        audio_tokens = self.audio_tokenizer.encode(prompt_audio_path)
        audio_tokens = "".join(f"<|audio_{i}|>" for i in audio_tokens)
        system_message[-1]["content"] = system_message[-1]["content"].replace(
            "<|audio|>", f"<|begin_of_audio|>{audio_tokens}<|end_of_audio|>"
        )
```

Message Construction

```
if audio_path is not None:
    messages = system_message + [
        {
            "role": "user",
            "content": message + "\n<|audio|>",
        },
    ]
else:
    messages = system_message + [
        {
            "role": "user",
            "content": message,
        },
    ]
```

Audio Processing for Both Streams

```
if (audio_path is not None or prompt_audio_path is not None) and
self.audio_tokenizer.apply_to_role(
    "user", is_contiguous=True
):
    # contiguous codec
    audio_paths = []
    if audio_path is not None:
        audio_paths.append(audio_path)
    if prompt_audio_path is not None:
        audio_paths.append(prompt_audio_path)
    input_ids, audios, audio_indices = add_audio_input_contiguous(
        input_ids, audio_paths, self.tokenizer, self.audio_tokenizer
    )
```

Voice Cloning Examples

Multiple Reference Voices

```
 # Clone TTS with different reference voices
for text in TTS_texts:
    for prompt_audio_path in [
        "asset/2631296891109983590.wav",
        "asset/379838640-d5ff0815-74f8-4738-b0f1-477cfc8dcc2d.wav",
        "asset/4202818730519913143.wav",
    ]:
        output, tts_speech = s2s_inference.run_infer(
            prompt_audio_path=prompt_audio_path,
            message="Convert the text to speech.\n" + text,
            mode=None,
            do_sample=True,
        )
```

This shows that: - **Multiple reference voices** are supported - **Voice cloning** works with different audio samples - **TTS task** specifically uses reference audio for voice characteristics

---

Code Analysis: web_demo.py

Interface Structure

Gradio Components

```python
with gr.Blocks() as demo:
    gr.Markdown("""<center><font size=8>VITA-Audio-Plus-Vanilla</center>""")

    chatbot = gr.Chatbot(
        label="VITA-Audio-Plus-Vanilla", elem_classes="control-height", height=500
    )
    query = gr.Textbox(lines=2, label="Text Input")
    task_history = gr.State([])

    with gr.Row():
        add_text_button = gr.Button("Submit Text")
        add_audio_button = gr.Button("Submit Audio")
        empty_bin = gr.Button("🧹 Clear History ")
        task = gr.Radio(choices=["ASR", "TTS", "Spoken QA"], label="TASK", value="Spoken QA")

    with gr.Row(scale=1):
        record_btn = gr.Audio(
            sources=["microphone", "upload"],
            type="filepath",
            label="🎤 Record or Upload Audio",
            show_download_button=True,
            waveform_options=gr.WaveformOptions(sample_rate=16000),
        )
        audio_output = gr.Audio(
            label="Play", streaming=True, autoplay=True, show_download_button=True
        )
```

Event Handlers

```python
add_text_button.click(
    add_text, [chatbot, task_history, query], [chatbot, task_history], show_progress=True
).then(reset_user_input, [], [query]).then(
    predict, [chatbot, task_history, task], [chatbot, audio_output], show_progress=True
)

add_audio_button.click(
    add_audio,
    [chatbot, task_history, record_btn],
    [chatbot, task_history],
    show_progress=True,
).then(predict, [chatbot, task_history, task], [chatbot, audio_output], show_progress=True)
```

Key Characteristics

No Reference Audio Support

- **Single audio input**: Only user's current speech
- **Default voice**: Uses model's learned voice characteristics
- **Simple workflow**: Input → Processing → Output

Task Support

- **ASR**: Automatic Speech Recognition
- **TTS**: Text-to-Speech synthesis
- **Spoken QA**: Speech-based question answering

Limitations

- **No voice cloning**: Cannot specify target voice
- **Batch processing**: No real-time streaming
- **Basic interface**: Limited customization options

Code Analysis: web_demo_stream.py

Streaming Architecture

Model Initialization

```
audio_tokenizer_path = snapshot_download(repo_id="THUDM/glm-4-voice-tokenizer")
flow_path = snapshot_download(repo_id="THUDM/glm-4-voice-decoder")

audio_tokenizer_rank = 0
audio_tokenizer_type = "sensevoice_glm4voice"

prompt_audio_path = None  # Key: Reference audio configuration
```

Reference Audio Processing

```
if prompt_audio_path is not None:
    if audio_tokenizer.apply_to_role("system", is_discrete=True):
        # discrete codec
        prompt_audio_tokens = audio_tokenizer.encode(prompt_audio_path)
        prompt_audio_tokens = "".join(f"<|audio_{i}|>" for i in
prompt_audio_tokens)
        system_message = [
            {
                "role": "system",
                "content": f"Your Voice: <|begin_of_audio|>
{prompt_audio_tokens}<|end_of_audio|>\n",
            },
        ]
    else:
        # contiguous codec
        system_message = default_system_message
```

Streaming Audio Generation

```
def generate_audio_stream():
    for new_text in streamer:
        if new_text:
            # Extract audio tokens from generated text
            audio_tokens = extract_audio_tokens(new_text)

            if audio_tokens:
                # Generate audio with reference voice characteristics
                tts_speech = audio_tokenizer.decode(
                    audio_tokens,
                    source_speech_16k=prompt_audio_path,  # Reference audio
influence
                    option_steps=option_steps,
                )

                yield tts_speech
```

Advanced Features

Progressive Quality Improvement

```
option_steps = min(option_steps + 2, 10)  # Gradually improve quality
```

Real-time Processing

- **Streaming generation**: Audio produced as tokens are generated
- **Low latency**: Immediate audio feedback
- **Progressive refinement**: Quality improves over time

Voice Cloning Integration

- **System message**: Reference audio embedded in conversation context
- **Decoder parameter**: Direct voice characteristics transfer
- **Dual influence**: Both prompt and synthesis level voice control

---

Code Analysis: web_demo_stream_local.py

Local Processing Optimization

Configuration

```
prompt_audio_path = None  # Configurable reference audio

# Local model paths (no external downloads)
audio_tokenizer_path = "local/path/to/tokenizer"
flow_path = "local/path/to/decoder"
```

Same Reference Audio Logic

```
if prompt_audio_path is not None:
    if audio_tokenizer.apply_to_role("system", is_discrete=True):
        prompt_audio_tokens = audio_tokenizer.encode(prompt_audio_path)
        prompt_audio_tokens = "".join(f"<|audio_{i}|>" for i in
prompt_audio_tokens)
        system_message = [
            {
                "role": "system",
                "content": f"Your Voice: <|begin_of_audio|>
{prompt_audio_tokens}<|end_of_audio|>\n",
            },
        ]
```

Local Audio Synthesis

```
tts_speech = audio_tokenizer.decode(
    audio_tokens,
    source_speech_16k=prompt_audio_path,  # Same voice cloning mechanism
    option_steps=option_steps,
)
```

Deployment Advantages

Privacy and Security

- **Local processing**: No data sent to external servers
- **Offline capability**: Works without internet connection
- **Data control**: Complete control over audio data

Performance

- **Reduced latency**: No network delays
- **Consistent performance**: Not affected by network conditions
- **Resource optimization**: Optimized for local hardware

Customization

- **Programmatic control**: Easy to modify reference audio
- **Integration friendly**: Simple to integrate into larger systems
- **Configuration flexibility**: Easy to adjust parameters

Reference Audio System Diagrams

Complete Audio Flow Visualization

The reference audio system in VITA-Audio works through a sophisticated multi-stage process:

Stage 1: Audio Input Processing

```
 Reference Audio File → Audio Tokenizer → Discrete Tokens
Input Audio File → Audio Tokenizer → Discrete Tokens
```

Stage 2: Message Construction

```
Reference Tokens → System Message: "Your Voice: <|begin_of_audio|>...
<|end_of_audio|>"
Input Tokens → User Message: "[text]\n<|begin_of_audio|>...<|end_of_audio|>"
```

Stage 3: Model Processing

```
System + User Messages → VITA-Audio Model → Response Tokens (Text + Audio)
```

Stage 4: Audio Synthesis

```
Response Audio Tokens + Reference Audio → GLM-4-Voice Decoder → CosyVoice →
Final Audio
```

Key Technical Insights

Dual Audio Influence

1. **Prompt Level**: Reference audio in system message guides model understanding
2. **Synthesis Level**: Reference audio in decoder parameters influences voice characteristics

Token Format Consistency

- **Input tokens**: `<|begin_of_audio|><|audio_0|><|audio_1|>...<|end_of_audio|>`
- **Reference tokens**: Same format, different role (system vs user)
- **Response tokens**: Generated in same format for synthesis

Voice Cloning Mechanism

- **Zero-shot**: No training required for new voices
- **Cross-lingual**: Voice characteristics preserved across languages
- **High fidelity**: Professional-grade voice cloning quality

---

Complete Audio Flow Visualization

The Four Implementation Patterns

Pattern 1: Basic (web_demo.py)

```
User Audio → VITA-Audio → Default Voice Response
```

- **Simple**: Single audio stream
- **Limited**: No voice customization
- **Fast**: Minimal processing overhead

Pattern 2: Reference-Enabled (inference_sts.py)

```
 Reference Audio → System Message
User Audio → User Message
Both → VITA-Audio → Cloned Voice Response
```

- **Flexible**: Multiple reference voices supported
- **Powerful**: Full voice cloning capabilities
- **Complex**: Requires reference audio management

Pattern 3: Streaming Reference (web_demo_stream.py)

```
 Reference Audio → System Message (streaming)
User Audio → User Message (streaming)
Both → VITA-Audio → Real-time Cloned Voice Response
```

- **Real-time**: Immediate audio feedback
- **High-quality**: Progressive quality improvement
- **Advanced**: Streaming voice cloning

Pattern 4: Local Reference (web_demo_stream_local.py)

```
 Reference Audio → System Message (local processing)
User Audio → User Message (local processing)
Both → VITA-Audio → Local Cloned Voice Response
```

- **Private**: All processing local
- **Reliable**: No network dependencies
- **Deployable**: Production-ready

Where Audio Comes From - Complete Analysis

Reference Audio Sources

1. **Asset Files**: Pre-recorded samples in `asset/` directory
2. `asset/2631296891109983590.wav`
3. `asset/379838640-d5ff0815-74f8-4738-b0f1-477cfc8dcc2d.wav`
4. `asset/4202818730519913143.wav`
5. **User Uploads**: Via web interface file upload
6. Gradio Audio component with upload capability
7. Custom reference voices from users
8. **Programmatic Setting**: Direct configuration in code
9. `prompt_audio_path = "path/to/reference.wav"`
10. Dynamic reference audio selection
11. **Microphone Recording**: Real-time reference capture
12. Record reference voice sample
13. Use immediately for voice cloning

Input Audio Sources

1. **Microphone Recording**: Real-time user speech
2. Gradio Audio component with microphone access
3. Live conversation input
4. **File Upload**: Pre-recorded user audio

5. Upload existing audio files
6. Batch processing capability
7. **Streaming Input**: Continuous audio stream
8. Real-time conversation
9. Low-latency processing

Output Audio Generation

1. **GLM-4-Voice Decoder**: Converts response tokens to speech tokens
2. **CosyVoice Synthesis**: Final audio generation with voice characteristics
3. **Reference Audio Influence**: Voice cloning from `source_speech_16k` parameter

---

Conclusion

Key Discoveries Summary

1. The "Your Voice" System Message

- **Purpose**: Provides reference audio for voice cloning
- **Format**: `"Your Voice: <|begin_of_audio|>[tokens]<|end_of_audio|>"`
- **Function**: Guides VITA-Audio to use specific voice characteristics

2. Dual Audio Architecture

- **Reference Audio**: Defines target voice characteristics
- **Input Audio**: Contains conversation content
- **Combined Processing**: Both influence final output

3. Implementation Variations

- **Basic**: No reference audio support
- **Advanced**: Full voice cloning capabilities

- **Streaming**: Real-time voice cloning
- **Local**: Privacy-focused deployment

4. Technical Innovation

- **Zero-shot Voice Cloning**: No training required for new voices
- **System Message Integration**: Novel approach to voice control
- **Dual Influence Mechanism**: Both prompt and synthesis level control

Practical Implications

For Developers

- **Reference audio is optional** but enables powerful voice cloning
- **System message approach** is novel and effective
- **Multiple implementation patterns** available for different use cases

For Users

- **Voice customization** possible with reference audio
- **High-quality voice cloning** without training
- **Real-time capabilities** for natural conversation

For Researchers

- **Novel architecture** for voice control in language models
- **Dual audio processing** paradigm
- **System message innovation** for multimodal AI

This comprehensive analysis reveals that VITA-Audio's reference audio system represents a significant innovation in conversational AI, enabling sophisticated voice cloning through an elegant system message approach combined with dual audio processing architecture.