GitaGPT API Documentation

Overview

GitaGPT provides a REST API for processing spiritual questions about the Bhagavad Gita using AI-powered semantic search, speech recognition, and text-to-speech synthesis.

Base URL

```
http://<server-ip>:5000
```

Authentication

Currently no authentication required for local network deployment.

Endpoints

Health Check

GET `/health`

Check if the server is running and models are loaded.

Response

```
{
  "status": "healthy",
  "models_loaded": true
}
```

Status Codes

- `200` - Server healthy and ready
- `503` - Models not loaded, server not ready

Example

```
curl http://192.168.1.100:5000/health
```

Process Audio Question

POST `/process_audio`

Submit audio question and receive transcription, spiritual guidance, and audio response.

Request

- **Content-Type**: `application/octet-stream`
- **Body**: Raw PCM audio bytes (int16, mono, 16kHz)

Audio Format Requirements

- **Sample Rate**: 16000 Hz
- **Channels**: 1 (mono)
- **Bit Depth**: 16-bit
- **Format**: PCM (signed integer)
- **Endianness**: Little-endian

Response


```
{
  "transcription": "What is the meaning of dharma?",
  "response": "According to Krishna in the Bhagavad Gita, dharma refers to
righteous living and one's moral duty...",
  "formatted_response": "🕉 Krishna's Wisdom:\n\nAccording to Krishna in the
Bhagavad Gita, dharma refers to righteous living...",
  "audio": "52494646b8af0100574156456d7420...",
  "response_raw": "According to Krishna in the Bhagavad Gita [1.1], dharma
refers to..."
}
```


Response Fields

- `transcription` - Speech-to-text result of the question
- `response` - Clean response without citation markers
- `formatted_response` - Response with emoji formatting for display
- `audio` - Hex-encoded WAV audio bytes (can be null if TTS fails)
- `response_raw` - Raw LLM output including citation tokens

Status Codes

- `200` - Success
- `400` - Bad request (invalid audio data)
- `503` - Models not loaded
- `500` - Internal server error

Example (Python)

```python
import requests
import numpy as np

# Record or load audio (16kHz, mono, int16)
audio_data = np.array([...], dtype=np.int16)
audio_bytes = audio_data.tobytes()

response = requests.post(
    'http://192.168.1.100:5000/process_audio',
    data=audio_bytes,
    headers={'Content-Type': 'application/octet-stream'}
)

if response.status_code == 200:
    result = response.json()
    print(f"Question: {result['transcription']}")
    print(f"Answer: {result['response']}")

    # Play audio response
    if result['audio']:
        audio_hex = result['audio']
        audio_wav = bytes.fromhex(audio_hex)
        with open('response.wav', 'wb') as f:
            f.write(audio_wav)
```

Audio Processing Pipeline

Input Processing

1. **Raw PCM bytes** received via HTTP POST
2. **Format validation** (int16, mono channel extraction if stereo)
3. **Normalization** to float32 range [-1.0, 1.0]

AI Processing

1. **Speech Recognition**: OpenAI Whisper transcription
2. **Semantic Search**: FAISS similarity search across Gita verses
3. **Context Building**: Retrieve top-K relevant verses
4. **LLM Generation**: Generate response using Ollama/Transformers
5. **Response Cleaning**: Remove citation tokens for natural speech

Output Generation

1. **Text-to-Speech**: Piper TTS → pyttsx3 → win32com SAPI fallback
2. **Audio Encoding**: WAV format, hex-encoded for JSON transport
3. **Response Formatting**: Add spiritual emojis and structure

---

Error Handling

Common Errors

400 Bad Request

```
{
  "error": "No audio data received"
}
```

Solution: Ensure audio data is included in request body

400 Bad Request

```
{
  "error": "Failed to parse int16 audio bytes: ..."
}
```

Solution: Check audio format (must be int16 PCM)

503 Service Unavailable

```
{
  "error": "Models not loaded on server."
}
```

Solution: Wait for server initialization, check server logs

500 Internal Server Error

```
{
  "error": "Whisper transcription failed: ..."
}
```

Solution: Check audio quality, duration, and format

Error Response Format

All errors return JSON with `error` field:

```
{
  "error": "Description of what went wrong"
}
```

Rate Limiting

Currently no rate limiting implemented. For production deployment, consider:

- Request throttling per IP
- Concurrent request limits
- Audio duration limits

Performance Considerations

Server Requirements

- **RAM**: 8GB+ for model loading
- **CPU**: Multi-core for parallel processing
- **Network**: Stable connection for audio streaming

Optimization Tips

- Use smaller Whisper models for faster transcription
- Pre-build FAISS index to reduce startup time
- Consider GPU acceleration for Whisper
- Implement connection pooling for multiple clients

Typical Response Times

- **Transcription**: 1-5 seconds (depends on Whisper model size)
- **Semantic Search**: 100-500ms
- **LLM Generation**: 2-10 seconds (varies by model)
- **TTS**: 1-3 seconds
- **Total**: 5-20 seconds per request

Integration Examples

JavaScript/Node.js

```
const fs = require('fs');
const axios = require('axios');

async function askGita(audioFilePath) {
    const audioData = fs.readFileSync(audioFilePath);
```

```
    try {
        const response = await axios.post(
            'http://192.168.1.100:5000/process_audio',
            audioData,
            {
                headers: { 'Content-Type': 'application/octet-stream' }
            }
        );

        return response.data;
    } catch (error) {
        console.error('API Error:', error.response.data);
        throw error;
    }
}
```

Arduino/ESP32

```
#include <WiFi.h>
#include <HTTPClient.h>

void sendAudioToGita(uint8_t* audioData, size_t length) {
    HTTPClient http;
    http.begin("http://192.168.1.100:5000/process_audio");
    http.addHeader("Content-Type", "application/octet-stream");

    int httpResponseCode = http.POST(audioData, length);

    if (httpResponseCode == 200) {
        String response = http.getString();
        // Parse JSON response
        Serial.println(response);
    }

    http.end();
}
```

Monitoring and Logging

Server Logs

Monitor console output for:

- Model loading status
- Request processing times
- Error messages and stack traces
- Audio processing statistics

Health Monitoring

Implement periodic health checks:

```
# Check every 30 seconds
while true; do
    curl -f http://192.168.1.100:5000/health || echo "Server down!"
    sleep 30
done
```

Security Considerations

⚠ **Warning**: This API is designed for local network use only.

Current Limitations

- No authentication or authorization
- No input validation beyond format checks
- No rate limiting or abuse protection
- Processes any audio input without content filtering

Production Recommendations

- Add API key authentication
- Implement request signing
- Add input sanitization
- Use HTTPS with proper certificates
- Add logging and monitoring
- Implement proper error handling without exposing internals