A-Z Guide: Building a Voice AI Agent with Twilio, OpenRouter, and Sarvam AI

This guide provides a comprehensive, step-by-step process for building a functional voice AI agent prototype using Twilio for telephony, OpenRouter for LLM intelligence, and Sarvam AI for authentic Indian language Text-to-Speech (TTS) and potentially Speech-to-Text (STT).

Goal: Create an AI agent that can handle phone calls, understand user speech, generate intelligent responses, and speak back using natural-sounding Indian voices.

Target Audience: Developers familiar with Python, Flask, and basic API concepts.

Table of Contents

6. **Testing & Validation**
    - Making Test Calls
    - Checking Logs
    - Verifying Sarvam TTS Output
7. **Troubleshooting Common Issues**
    - "An error has occurred" on Call Pickup
    - Sarvam TTS Audio Not Playing
    - Incorrect Language/Voice
    - API Errors (Twilio, OpenRouter, Sarvam)
    - Webhook Failures
8. **Next Steps & Improvements**

---

1. Prerequisites & Setup

Required Accounts & API Keys

- **Twilio:**
    - Account SID
    - Auth Token
    - A Twilio Phone Number (capable of voice calls)
- **OpenRouter:**
    - API Key (or another LLM provider like OpenAI)
- **Sarvam AI:**
    - API Key (for accessing Bulbul TTS and potentially STT)
- **Ngrok:**
    - Account (Free tier is sufficient for testing)

Software Installation

Ensure you have Python (3.8+ recommended) and pip installed.

Create a project directory and install necessary libraries:

```
 mkdir voice-ai-agent
cd voice-ai-agent
python -m venv venv
source venv/bin/activate  # On Windows use `venv\Scripts\activate`
pip install Flask twilio requests python-dotenv
```

(Using `python-dotenv` is recommended for managing API keys securely instead of hardcoding them.)

Project Structure

Organize your project like this:

```
voice-ai-agent/
├── venv/                # Virtual environment
├── audio_files/         # Directory to store generated TTS audio
├── .env                 # File to store API keys (add to .gitignore!)
├── app.py               # Main Flask application code
├── requirements.txt     # List of dependencies
└── README.md            # Project documentation
```

Create an empty `audio_files` directory.

Create a `.env` file (and add it to your `.gitignore`) with your API keys:

```
TWILIO_ACCOUNT_SID="ACxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
TWILIO_AUTH_TOKEN="your_twilio_auth_token"
TWILIO_PHONE_NUMBER="+1xxxxxxxxxx"
OPENROUTER_API_KEY="sk-or-v1-xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
SARVAM_API_KEY="your_sarvam_api_key"
SERVER_URL="" # Will be set later with ngrok URL
```

---

2. Core Components Implementation

(This section details the Python classes and functions needed. Refer to the previously provided *voice_agent_with_sarvam_tts.py* for a complete example, but understand the components here.)

Flask Application Setup (`app.py`)

```python
import os
import logging
from flask import Flask, request, jsonify, send_file
from dotenv import load_dotenv

load_dotenv() # Load environment variables from .env

app = Flask(__name__)

# Configure logging
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)
```

```
# Create audio directory if it doesn't exist
AUDIO_DIR = os.path.join(os.path.dirname(os.path.abspath(__file__)), 'audio_files')
os.makedirs(AUDIO_DIR, exist_ok=True)

# Load configuration from environment variables
TWILIO_ACCOUNT_SID = os.getenv("TWILIO_ACCOUNT_SID")
TWILIO_AUTH_TOKEN = os.getenv("TWILIO_AUTH_TOKEN")
TWILIO_PHONE_NUMBER = os.getenv("TWILIO_PHONE_NUMBER")
OPENROUTER_API_KEY = os.getenv("OPENROUTER_API_KEY")
SARVAM_API_KEY = os.getenv("SARVAM_API_KEY")
SERVER_URL = os.getenv("SERVER_URL") # This needs to be updated after starting ngrok

# ... (Rest of the component implementations will go here) ...

if __name__ == '__main__':
    if not all([TWILIO_ACCOUNT_SID, TWILIO_AUTH_TOKEN,
TWILIO_PHONE_NUMBER, OPENROUTER_API_KEY, SARVAM_API_KEY]):
        logger.error("ERROR: Missing one or more API keys in .env file!")
    else:
        logger.info("API Keys loaded successfully.")
    # Note: SERVER_URL check happens later
    app.run(debug=True, host='0.0.0.0', port=5000)
```

Twilio Integration

```
 from twilio.rest import Client
from twilio.twiml.voice_response import VoiceResponse, Gather, Play

twilio_client = None
try:
    if TWILIO_ACCOUNT_SID and TWILIO_AUTH_TOKEN:
        twilio_client = Client(TWILIO_ACCOUNT_SID, TWILIO_AUTH_TOKEN)
        logger.info("Twilio client initialized.")
    else:
        logger.warning("Twilio credentials not found in .env")
except Exception as e:
    logger.error(f"Twilio initialization failed: {e}")

# Functions for making calls and generating TwiML will use twilio_client
# and VoiceResponse, Gather, Play classes.
```

OpenRouter LLM Integration

```
 import requests
```

```
def get_llm_response(user_input, conversation_history, system_message):
    logger.info(f"Getting LLM response for: {user_input[:50]}...")
    if not OPENROUTER_API_KEY:
        logger.error("OpenRouter API Key not configured.")
        return "Error: LLM not configured."
    try:
        url = "https://openrouter.ai/api/v1/chat/completions"
        messages = [{"role": "system", "content": system_message}]
        messages.extend(conversation_history[-6:]) # Keep context limited
        messages.append({"role": "user", "content": user_input})

        headers = {
            "Authorization": f"Bearer {OPENROUTER_API_KEY}",
            "Content-Type": "application/json"
        }
        data = {
            "model": "openai/gpt-3.5-turbo", # Or choose another model
            "messages": messages,
            "max_tokens": 80,
            "temperature": 0.7
        }
        response = requests.post(url, headers=headers, json=data, timeout=15)

        if response.status_code == 200:
            result = response.json()
            ai_response = result['choices'][0]['message']['content'].strip()
            logger.info(f"LLM response: {ai_response[:50]}...")
            return ai_response
        else:
            logger.error(f"LLM API error: {response.status_code} - {response.text}")
            return "I encountered an issue generating a response."
    except Exception as e:
        logger.error(f"LLM request error: {e}")
        return "I had trouble connecting to the AI model."
```

Sarvam AI TTS Integration (Bulbul Model)

This is the critical part for getting the voice right.

```
 import uuid
import time
from urllib.parse import urljoin

def generate_sarvam_tts(text, language="hi", voice="female"):
    logger.info(f"Generating Sarvam TTS for: '{text[:50]}...' in {language}")
    if not SARVAM_API_KEY:
        logger.error("Sarvam AI API Key not configured.")
        return None
    if not SERVER_URL:
```

```
        logger.error("SERVER_URL not configured. Cannot create audio URL.")
        return None

    try:
        api_url = "https://api.sarvam.ai/v1/tts"
        filename = f"tts_{int(time.time())}_{uuid.uuid4().hex[:8]}.mp3"
        output_path = os.path.join(AUDIO_DIR, filename)

        payload = {
            "text": text,
            "language": language,
            "voice": voice,
            "model": "bulbul-v2"
        }
        headers = {
            "Authorization": f"Bearer {SARVAM_API_KEY}",
            "Content-Type": "application/json"
        }

        response = requests.post(api_url, headers=headers, json=payload, timeout=20) # Increased timeout

        if response.status_code == 200:
            with open(output_path, 'wb') as f:
                f.write(response.content)
            logger.info(f"Sarvam TTS audio saved to: {output_path}")
            # IMPORTANT: Return the PUBLICLY ACCESSIBLE URL
            audio_url = urljoin(SERVER_URL, f'/audio/{filename}')
            logger.info(f"Sarvam TTS audio URL: {audio_url}")
            return audio_url
        else:
            logger.error(f"Sarvam TTS API error: {response.status_code} - {response.text}")
            return None
    except requests.exceptions.Timeout:
        logger.error("Sarvam TTS API request timed out.")
        return None
    except Exception as e:
        logger.error(f"Sarvam TTS generation error: {e}")
        return None

# Basic language detection (can be improved)
def detect_language(text):
    if any('\u0900' <= c <= '\u097F' for c in text): return "hi"
    # Add more language checks here (Tamil, Telugu, etc.)
    return "en"
```

(Optional) Sarvam AI STT Integration

Sarvam AI also offers STT. If you want to use it instead of Twilio's default STT:

1. **Modify Twilio `<Gather>`:** You would need to use `<Record>` instead of `<Gather input='speech'>` to capture raw audio.
2. **Send Audio to Sarvam STT:** Get the recording URL from Twilio, download the audio, and send it to the Sarvam STT API endpoint.
3. **Process Transcript:** Use the transcript from Sarvam STT as input for the LLM.

Note: This adds complexity (audio handling, API calls). For the prototype, using Twilio's built-in STT via `<Gather input='speech'>` is simpler and often sufficient.

Sarvam AI STT API details would be needed here if implementing. The guide will assume Twilio STT for simplicity unless specified otherwise.

Audio File Management

We need an endpoint to serve the generated audio files.

```
@app.route('/audio/<filename>')
def serve_audio(filename):
    try:
        file_path = os.path.join(AUDIO_DIR, filename)
        if os.path.exists(file_path):
            logger.info(f"Serving audio file: {filename}")
            return send_file(file_path, mimetype='audio/mpeg')
        else:
            logger.error(f"Audio file not found: {filename}")
            return "File not found", 404
    except Exception as e:
        logger.error(f"Error serving audio {filename}: {e}")
        return "Error serving file", 500
```

---

3. Call Handling Logic

(Store conversation history, e.g., in a simple dictionary keyed by CallSid)

```
conversations = {}
```

Outbound Call Initiation

```
def make_outbound_call(phone_number, system_message, greeting):
    if not twilio_client:
        return {"success": False, "error": "Twilio client not initialized"}
    if not SERVER_URL:
        return {"success": False, "error": "SERVER_URL not configured"}

    try:
        webhook_url = urljoin(SERVER_URL, '/voice_webhook')
        call = twilio_client.calls.create(
            to=phone_number,
            from_=TWILIO_PHONE_NUMBER,
            url=webhook_url,
            method='POST'
        )
        conversations[call.sid] = {
            'history': [],
            'system_message': system_message,
            'greeting': greeting
        }
        logger.info(f"Initiated call {call.sid} to {phone_number}")
        return {"success": True, "call_sid": call.sid}
    except Exception as e:
        logger.error(f"Failed to initiate call to {phone_number}: {e}")
        return {"success": False, "error": str(e)}
```

Inbound Call Webhook (`/voice_webhook`)

This is where the initial greeting happens and where the "error occurred" likely stems from.

```
@app.route('/voice_webhook', methods=['POST'])
def voice_webhook():
    response = VoiceResponse()
    call_sid = request.form.get('CallSid')
    logger.info(f"Incoming call webhook for {call_sid}")

    # Retrieve or set up conversation state
    if call_sid not in conversations:
        # For inbound calls, define default greeting/system message
        conversations[call_sid] = {
            'history': [],
            'system_message': "You are a helpful AI assistant. Respond in the language
the user speaks (Hindi or English). Keep responses concise.",
            'greeting': "          !                                          ?"
        }
    call_data = conversations[call_sid]
```

```
    greeting = call_data['greeting']
    greeting_language = detect_language(greeting)

    # *** CRITICAL STEP: Generate and Play Greeting ***
    greeting_audio_url = generate_sarvam_tts(greeting, language=greeting_language)

    if greeting_audio_url:
        logger.info(f"Playing Sarvam TTS greeting: {greeting_audio_url}")
        response.play(greeting_audio_url)
    else:
        # ** FALLBACK / ERROR HANDLING **
        logger.error(f"Failed to generate Sarvam TTS for greeting. Using fallback.")
        # Option 1: Use Twilio TTS fallback
        # response.say(greeting, voice='Polly.Aditi', language='hi-IN') # Example Twilio
Hindi
        # Option 2: Play a generic pre-recorded error/greeting MP3
        # response.play(urljoin(SERVER_URL, '/audio/generic_greeting.mp3'))
        # Option 3: Say a simple error message (Leads to user hearing "error occurred")
        response.say("Sorry, I encountered an issue starting the call. Please try again
later.", voice='alice')
        response.hangup()
        return str(response)

    # Proceed to gather user input
    gather = Gather(
        input='speech',
        action=urljoin(SERVER_URL, '/process_speech'),
        method='POST',
        speech_timeout='auto',
        language='en-IN,hi-IN' # Listen for multiple languages
    )
    # Optional: Add a spoken prompt after the greeting
    # gather.say("Please tell me how I can help.", voice='alice')
    response.append(gather)

    # If gather times out (user says nothing)
    response.redirect(urljoin(SERVER_URL, '/handle_no_input'), method='POST')

    return str(response)

@app.route('/handle_no_input', methods=['POST'])
def handle_no_input():
    response = VoiceResponse()
    no_input_message = "I didn't hear anything. Goodbye."
    no_input_audio_url = generate_sarvam_tts(no_input_message, language='en')
    if no_input_audio_url:
        response.play(no_input_audio_url)
    else:
        response.say(no_input_message, voice='alice')
    response.hangup()
    return str(response)
```

Speech Processing Webhook (`/process_speech`)

```
@app.route('/process_speech', methods=['POST'])
def process_speech():
    response = VoiceResponse()
    call_sid = request.form.get('CallSid')
    speech_result = request.form.get('SpeechResult', '').strip()
    logger.info(f"Processing speech for {call_sid}: '{speech_result}'")

    if call_sid not in conversations:
        logger.error(f"Conversation state not found for {call_sid}")
        response.say("Sorry, there was a system error. Goodbye.")
        response.hangup()
        return str(response)

    call_data = conversations[call_sid]
    conversation_history = call_data['history']
    system_message = call_data['system_message']

    if not speech_result:
        # Handle empty speech result (e.g., silence)
        no_understand_msg = "Sorry, I didn't catch that. Could you please repeat?"
        no_understand_audio_url = generate_sarvam_tts(no_understand_msg, 'en')
        if no_understand_audio_url:
            response.play(no_understand_audio_url)
        else:
            response.say(no_understand_msg, voice='alice')
    else:
        # Add user input to history
        conversation_history.append({"role": "user", "content": speech_result})

        # Get LLM response
        ai_response_text = get_llm_response(speech_result, conversation_history,
system_message)

        # Add AI response to history
        conversation_history.append({"role": "assistant", "content": ai_response_text})

        # Generate Sarvam TTS for the response
        response_language = detect_language(ai_response_text)
        ai_audio_url = generate_sarvam_tts(ai_response_text,
language=response_language)

        if ai_audio_url:
            response.play(ai_audio_url)
        else:
            logger.error("Failed to generate Sarvam TTS for AI response. Using fallback.")
            # Fallback to Twilio TTS
            # response.say(ai_response_text, voice='Polly.Aditi', language='hi-IN') # Example
            response.say("I have a response, but I'm having trouble speaking it.",
```

```
voice='alice')

    # Check for end-of-call phrases
    end_phrases = ['goodbye', 'bye', '            ', '            ']
    if any(phrase in speech_result.lower() for phrase in end_phrases):
        goodbye_msg = "Thank you for calling. Goodbye!"
        goodbye_audio_url = generate_sarvam_tts(goodbye_msg, 'en')
        if goodbye_audio_url:
            response.play(goodbye_audio_url)
        else:
            response.say(goodbye_msg, voice='alice')
        response.hangup()
    else:
        # Continue gathering input
        gather = Gather(
            input='speech',
            action=urljoin(SERVER_URL, '/process_speech'),
            method='POST',
            speech_timeout='auto',
            language='en-IN,hi-IN'
        )
        response.append(gather)
        response.redirect(urljoin(SERVER_URL, '/handle_no_input'), method='POST')

    # Update conversation state
    conversations[call_sid]['history'] = conversation_history
    return str(response)
```

Conversation State Management

The `conversations` dictionary stores the history and configuration for each active call, keyed by `CallSid`.

---

4. Putting It Together: The Code

Combine all the Python snippets above into your `app.py` file. Ensure imports are at the top and the Flask app is run at the bottom. The structure mirrors the `voice_agent_with_sarvam_tts.py` file provided previously, but this guide breaks down why each part exists.

---

5. Configuration & Deployment

API Key Configuration

- Ensure your `.env` file is correctly populated with **all** required API keys (Twilio SID, Token, Phone; OpenRouter Key; Sarvam Key).

Webhook Setup (ngrok)

1. Start your Flask app: `python app.py`
2. Start ngrok: `ngrok http 5000`
3. Copy the `https://` URL provided by ngrok.
4. **CRITICAL:** Update the `SERVER_URL` in your `.env` file with this ngrok URL.
5. **Restart your Flask app** so it picks up the new `SERVER_URL` from the `.env` file.
6. Configure your Twilio phone number's voice webhook: Go to your Twilio console -> Phone Numbers -> Active Numbers -> Click your number -> Under "Voice & Fax", set "A CALL COMES IN" to Webhook, paste your ngrok URL followed by `/voice_webhook` (e.g., `https://your-ngrok-url.ngrok-free.app/voice_webhook`), and set the method to `HTTP POST`. Save.

Running the Application

```
source venv/bin/activate
python app.py
```

Keep both the Flask app and ngrok running.

---

6. Testing & Validation

1. **Outbound Test:** If you implement an endpoint to trigger `make_outbound_call`, use it to call your own phone.
2. **Inbound Test:** Call your Twilio phone number from another phone.
3. **Listen Carefully:** Does the initial greeting play correctly using the Sarvam AI voice? Or do you hear the "error occurred" message?
4. **Speak:** Interact with the agent in Hindi and English.
5. **Check Logs:** Monitor the Flask application logs (`python app.py` output) for errors related to API calls (Sarvam, OpenRouter), audio file generation/serving, or webhook processing.

6. **Check Twilio Logs:** Review the call logs in your Twilio console for detailed error information if calls fail.

---

7. Troubleshooting Common Issues

"An error has occurred" on Call Pickup

This is the **most likely issue** you described and almost always happens in the `/voice_webhook` handler before the first `<Gather>`.

- **Root Cause:** Failure to generate or play the initial greeting audio using `generate_sarvam_tts`.
- **Checklist:**
    1. **Sarvam API Key:** Is `SARVAM_API_KEY` correct in your `.env` file? Is the key active?
    2. **Sarvam API Status:** Is the Sarvam AI API operational? Check their status page if available.
    3. **Network Issues:** Can your server reach `api.sarvam.ai`? (Firewall, DNS issues?)
    4. **`SERVER_URL` :** Is the `SERVER_URL` in `.env` correctly set to your **HTTPS** ngrok URL? Did you **restart the Flask app** after setting it?
    5. **`/audio/` Endpoint:** Can Twilio reach the generated audio file URL (e.g., `https://your-ngrok-url.ngrok-free.app/audio/tts_....mp3`)? Try accessing this URL directly in your browser. Does it work? Check ngrok logs for requests to `/audio/`.
    6. **File Permissions:** Does your Flask application have permission to write files to the `audio_files` directory?
    7. **Sarvam API Timeout:** The `generate_sarvam_tts` function includes a timeout. Check logs for timeout errors. Maybe increase the timeout value slightly.
    8. **TwiML Generation:** Look at the Flask logs. Does it show an error before generating the TwiML for the `/voice_webhook`? If the `generate_sarvam_tts` call returns `None`, the fallback logic is triggered. If the fallback also fails or isn't robust, Twilio might default to its error message.
    9. **Twilio Debugger:** Check the Twilio Console -> Monitor -> Logs -> Errors. Look for errors related to your CallSid, especially TwiML fetch/parse errors or invalid URL errors for the `<Play>` verb.

- **Debugging Strategy:**
    - Add more detailed logging within `generate_sarvam_tts` and `/voice_webhook`.
    - Temporarily replace `response.play(greeting_audio_url)` with a simple `response.say("Testing greeting.", voice=\'alice\')` to isolate the TTS generation issue.