Goal: Understand LLM APIs, Speech AI (STT/TTS), and build production-ready voice AI agents using Pipecat and LiveKit, deployed publicly.

Prerequisites: Sign Up for All Services (Do This First)

Complete these signups before starting Week 2. All have free tiers sufficient for learning.

Service	Sign Up URL	What You Need
OpenAI	platform.openai.com	API key, add $5-10 credits
Deepgram	console.deepgram.com	API key (free $200 credits)
ElevenLabs	elevenlabs.io	API key (free tier: 10k chars/month)
LiveKit Cloud	cloud.livekit.io	Free tier account
Railway	railway.app	For deployment (free tier: $5/month credits)

Day 1:

LLM APIs — The Brain of Your Agent

Goal: Understand how LLMs work via APIs, make your first API calls, and build a conversational chatbot.

Shorts to Watch First

- GPT-4 in 100 Seconds - Fireship
- OpenAI Tokenizer — interactive tool to see how tokens work

Part 1: LLM Fundamentals (Learn from Claude)

Ask Claude (browser) to explain these concepts:

Tokens:

- LLMs don't see words, they see "tokens" (word pieces)
- "Hello world" = 2 tokens, "Authentication" = 1-2 tokens
- Why it matters: You pay per token, context has token limits
- Rule of thumb: 1 token ≈ 4 characters or ¾ of a word

Context Window:

- The "memory" of a single conversation
- GPT-5.2: 128k tokens
- Everything (system prompt + conversation history + your message) must fit
- For voice AI: Keep it small for low latency

Temperature:

- Controls randomness/creativity (0.0 to 2.0)
- 0.0 = deterministic, same input → same output
- 1.0 = creative, varied responses
- For voice AI: Usually 0.7-0.8 (natural but not wild)

Streaming:

- Normal: Wait for full response, then display
- Streaming: Get response token-by-token as it's generated

- For voice AI: Essential! Start speaking before full response is ready

System Prompts:

- Instructions that define the AI's personality and behavior
- Set at the start of conversation, persists throughout
- For voice AI: "You are a helpful receptionist for Acme Corp..."

Verification: Can you explain to someone what happens if your context window fills up? (Answer: Oldest messages get dropped or you get an error)

Part 2: API Structure (Learn from Claude)

Ask Claude (browser) to explain the anatomy of an API call:

OpenAI Chat Completions API:

python

```python
response = openai.chat.completions.create(
    model="gpt-4o",
    messages=[
        {"role": "system", "content": "You are a helpful assistant."},
        {"role": "user", "content": "Hello!"},
        {"role": "assistant", "content": "Hi there! How can I help?"},
        {"role": "user", "content": "What's the weather?"}
    ],
    temperature=0.7,
    max_tokens=150,
```

```
    stream=True  # For streaming
)
```

Message Roles:
- `system`: Instructions for the AI (set once)
- `user`: Human messages
- `assistant`: AI responses (include previous ones for context)

Verification: Why do we include previous assistant messages in the messages array?

```python
### Mini-Projects

---

### Project 1: First OpenAI API Call

Ask Claude Code to create a Python script that:

**Requirements:**
- Loads API key from `.env` file
- Makes a simple call to OpenAI GPT-5.2
- Prints the response
```

```
- Measures and prints response time

**Your job:**
1. Create `.env` file with your API key:
```
   OPENAI_API_KEY=sk-...
```
2. Run the script
3. Try changing the prompt
4. Try changing temperature (0.0 vs 1.0) — notice the difference?

**Verification Checklist:**
- [ ] OpenAI API call works
- [ ] Response printed
- [ ] Response time measured
- [ ] Tried different temperatures

---

### Project 2: Streaming Responses

Ask Claude Code to modify the script to:

**Requirements:**
- Use streaming API
- Print tokens as they arrive (character by character effect)
- Measure time-to-first-token (TTFT) vs total time

**Your job:**
1. Run the streaming version
2. Notice how text appears incrementally
3. Note the TTFT — this is when voice AI can start speaking!
4. Compare TTFT to total response time

**Verification Checklist:**
- [ ] Streaming works
```

```
- [ ] Can see tokens arriving one by one
- [ ] Measured TTFT
- [ ] Understand why TTFT matters for voice AI

---

### Project 3: CLI Chatbot

Ask Claude Code to build a terminal chatbot:

**Requirements:**
- Interactive loop: user types → AI responds → repeat
- Maintains conversation history (sends all previous messages)
- Uses OpenAI GPT-5.2 with streaming
- System prompt: "You are a helpful assistant. Keep responses
concise."
- Type "quit" to exit
- Shows token count after each response

**Your job:**
1. Run the chatbot
2. Have a multi-turn conversation (5+ exchanges)
3. Notice how it remembers context from earlier messages
4. Watch the token count grow with each turn
5. Try to fill up the context (keep talking until it gets slow or
errors)

**Verification Checklist:**
- [ ] Chatbot runs in terminal
- [ ] Maintains conversation context
- [ ] Streaming responses work
- [ ] Token count displayed
- [ ] Can quit gracefully

---
```

```
### Project 4: Function Calling (Tools)

Ask Claude Code to add function calling to your chatbot:

**Requirements:**
- Add a "get_current_time" function that returns the current time
- Add a "get_weather" function (mock it — return fake data)
- LLM decides when to call functions based on user query
- Display when a function is called and its result

**Your job:**
1. Ask "What time is it?" — verify function is called
2. Ask "What's the weather in Tokyo?" — verify function is called
3. Ask "Tell me a joke" — verify NO function is called
4. Understand how the LLM decides when to use tools

**Verification Checklist:**
- [ ] Time function works
- [ ] Weather function works
- [ ] LLM correctly chooses when to call functions
- [ ] Regular questions work without function calls

---

### End of Day 1 Checklist

| Item | Verified? |
|------|-----------|
| Can explain tokens, context window, temperature | ☐ |
| Can explain streaming and why it matters | ☐ |
| OpenAI API calls working | ☐ |
| Streaming responses working | ☐ |
| CLI chatbot with history working | ☐ |
| Function calling working | ☐ |
| Understand time-to-first-token concept | ☐ |
```

Day 2:

Speech AI – STT & TTS

Goal: Understand speech-to-text (STT) and text-to-speech (TTS), work with audio formats, and build scripts that transcribe and generate speech.

Part 1: Audio Fundamentals (Learn from Claude)

Ask Claude (browser) to explain:

Audio Formats:
- **WAV:** Uncompressed, high quality, large files
- **MP3:** Compressed, smaller files, slight quality loss
- **PCM:** Raw audio data (what microphones capture)
- **Opus/WebM:** Modern, efficient, used in real-time streaming

Sample Rate:
- How many audio samples per second (measured in Hz)
- 8000 Hz (8kHz): Phone quality
- 16000 Hz (16kHz): Common for speech recognition
- 44100 Hz (44.1kHz): CD quality
- Higher = better quality but larger files

Channels:
- Mono (1 channel): Sufficient for voice
- Stereo (2 channels): Left/right, not needed for voice AI

```
**Bit Depth:**
- 16-bit: Standard for speech
- 24-bit: Higher quality, larger files

**For Voice AI:**
- Input (STT): 16kHz, mono, 16-bit PCM or WAV
- Output (TTS): Usually MP3 or PCM for streaming

**Verification:** If someone says "send me 16kHz mono PCM audio,"
do you understand what they mean?

---

### Part 2: Speech-to-Text (STT) with Deepgram

Ask Claude (browser) to explain:

**How STT Works:**
- Audio goes in → text comes out
- Models trained on millions of hours of speech
- Deepgram uses neural networks optimized for real-time

**Deepgram Features:**
- **Real-time streaming:** Transcribe as audio arrives
- **Pre-recorded:** Upload file, get transcript
- **Diarization:** Identify different speakers
- **Punctuation:** Automatic punctuation
- **Language detection:** Auto-detect language

**Key Deepgram Parameters:**
- `model`: "nova-3" (best accuracy)
- `language`: "en" for English
- `punctuate`: true/false
- `diarize`: true/false (speaker separation)
```

```
**Verification:** What's the difference between streaming STT and
pre-recorded STT? (Answer: Streaming gives you words as they're
spoken; pre-recorded processes entire file at once)

---

### Part 3: Text-to-Speech (TTS) with ElevenLabs

Ask Claude (browser) to explain:

**How TTS Works:**
- Text goes in → audio comes out
- Modern TTS uses neural networks for natural-sounding speech
- ElevenLabs is known for highly realistic voices

**ElevenLabs Features:**
- **Voice cloning:** Create custom voices
- **Pre-made voices:** Library of ready-to-use voices
- **Streaming:** Generate audio chunk by chunk
- **Multilingual:** Supports many languages

**Key Parameters:**
- `voice_id`: Which voice to use
- `model_id`: "eleven_turbo_v2_5" (fast) or
"eleven_multilingual_v2" (quality)
- `stability`: How consistent the voice is (0-1)
- `similarity_boost`: How close to original voice (0-1)

**For Voice AI:**
- Use streaming TTS to start playing audio before full generation
- Use "turbo" models for lower latency

**Verification:** Why do we want streaming TTS for voice AI?
(Answer: Start playing audio sooner, reduces perceived latency)

---
```

```
### Mini-Projects

---

### Project 1: Transcribe Audio File with Deepgram

Ask Claude Code to build a script that:

**Requirements:**
- Takes an audio file path as input (WAV or MP3)
- Sends to Deepgram's pre-recorded API
- Returns the transcript with timestamps
- Saves transcript to a text file

**Your job:**
1. Record a short audio clip (use your phone or QuickTime)
2. Save it as WAV or MP3
3. Run the script on your recording
4. Verify the transcript is accurate
5. Try with different audio qualities

**Verification Checklist:**
- [ ] Script accepts audio file
- [ ] Deepgram API call works
- [ ] Transcript is accurate
- [ ] Timestamps are included
- [ ] Saved to text file

---

### Project 2: Real-Time Transcription with Deepgram

Ask Claude Code to build a script that:

**Requirements:**
```

```
- Opens your microphone (use `pyaudio` or `sounddevice`)
- Streams audio to Deepgram's real-time API via WebSocket
- Prints words as they're recognized (live!)
- Press Ctrl+C to stop

**Your job:**
1. Run the script
2. Speak into your microphone
3. Watch words appear in real-time as you speak
4. Notice the slight delay (latency)
5. Try speaking fast vs slow

**Verification Checklist:**
- [ ] Microphone input works
- [ ] WebSocket connection to Deepgram established
- [ ] Words appear in real-time
- [ ] Can stop gracefully with Ctrl+C

---

### Project 3: Generate Speech with ElevenLabs

Ask Claude Code to build a script that:

**Requirements:**
- Takes text input (from command line or file)
- Sends to ElevenLabs API
- Saves generated audio as MP3
- Plays the audio automatically (use `playsound` or `pygame`)

**Your job:**
1. Run with a simple sentence
2. Listen to the output — does it sound natural?
3. Try different voices (get voice IDs from ElevenLabs dashboard)
4. Try different stability/similarity settings
5. Generate a longer paragraph
```

```
**Verification Checklist:**
- [ ] Text sent to ElevenLabs
- [ ] Audio file generated
- [ ] Audio plays automatically
- [ ] Tried different voices
- [ ] Quality sounds natural

---

### Project 4: Streaming TTS with ElevenLabs

Ask Claude Code to modify the script for streaming:

**Requirements:**
- Use ElevenLabs streaming API
- Play audio chunks as they arrive (not after full generation)
- Measure time-to-first-audio

**Your job:**
1. Compare time-to-first-audio: streaming vs non-streaming
2. Notice how streaming starts playing sooner
3. Try with a long paragraph — streaming advantage is more
obvious

**Verification Checklist:**
- [ ] Streaming API works
- [ ] Audio plays incrementally
- [ ] Measured time-to-first-audio
- [ ] Understand the latency benefit

---

### Project 5: Full Pipeline — Speech In → Text → Speech Out

Ask Claude Code to combine STT and TTS:
```

```
**Requirements:**
- Record audio from microphone (5 seconds)
- Transcribe with Deepgram
- Pass transcript to OpenAI for a response
- Generate speech from response with ElevenLabs
- Play the audio response

**Your job:**
1. Run the script
2. Speak a question (e.g., "What is the capital of France?")
3. Wait for the AI to respond with speech
4. Measure total end-to-end latency

This is your first "voice AI" — it's slow and not real-time, but
it's the full pipeline!

**Verification Checklist:**
- [ ] Microphone recording works
- [ ] Transcription works
- [ ] LLM response generated
- [ ] TTS audio generated and plays
- [ ] Measured end-to-end latency (probably 5-15 seconds — that's
okay for now!)

---

### End of Day 2 Checklist

| Item | Verified? |
|------|-----------|
| Understand audio formats (WAV, PCM, sample rate) | ☐ |
| Deepgram pre-recorded transcription working | ☐ |
| Deepgram real-time transcription working | ☐ |
| ElevenLabs TTS working | ☐ |
| ElevenLabs streaming TTS working | ☐ |
```

```
| Full pipeline (STT → LLM → TTS) working | ☐ |
| Understand where latency comes from | ☐ |
```

Here's the corrected and reformatted Day 3-7 for Google Docs:

Day 3

Voice AI Frameworks — Pipecat

Goal: Learn Pipecat framework for building real-time voice AI agents with proper latency handling.

Shorts to Watch First

- WebSockets in 100 Seconds - Fireship (https://youtube.com/watch?v=1BfCnjr_Vjg)
- Search: "Pipecat AI voice agent demo" on YouTube

Part 1: Why Frameworks? (Learn from Claude)

Ask Claude (browser) to explain:

The Problem with DIY Voice AI:

- Your Day 2 pipeline had 5-15 second latency
- Real conversations need <500ms response time
- Managing streaming, interruptions, turn-taking is complex
- Handling WebSocket connections, audio buffers is error-prone

What Pipecat Solves:

- Pipeline architecture: Audio flows through processors
- Streaming by default: Everything streams end-to-end
- Interruption handling: User can interrupt the AI mid-speech
- Transport agnostic: Works with Plivo, Daily, LiveKit, local audio
- Service integrations: Built-in support for Deepgram, ElevenLabs, OpenAI

Pipecat Architecture:

```
None
Audio In → VAD → STT → LLM → TTS → Audio Out
```

```
                ↓
(Voice Activity Detection - detects when user stops speaking)
```

Key Concepts:

- Frames: Units of data flowing through pipeline (audio, text, control)
- Processors: Transform frames (STT processor, LLM processor, etc.)
- Pipeline: Chain of processors
- Transport: How audio gets in/out (Plivo, Daily, LiveKit, local)

Verification: Why is streaming important for low-latency voice AI? (Answer: Each component can start processing before the previous one finishes)

Part 2: Pipecat Components We'll Use (Learn from Claude)

Ask Claude (browser) to explain each component:

Our Stack:

- Transport: PlivoTransport (for phone calls via WebSocket audio streaming)
- VAD: SileroVAD (voice activity detection)
- Turn Detection: SmartTurn (semantic turn detection - knows when user is done speaking)
- STT: DeepgramSTTService (real-time transcription)
- LLM: OpenAILLMService with GPT-4.1-mini (fast, streaming)
- TTS: ElevenLabsTTSService (high-quality, streaming)

SileroVAD - What & Why:

- Detects when someone starts speaking
- Detects when someone stops speaking
- Runs locally, very fast
- Essential for knowing when to start/stop listening

SmartTurn - What & Why:

- Goes beyond simple silence detection
- Uses semantic understanding to detect turn completion
- Knows difference between "thinking pause" and "done speaking"
- Reduces false interruptions

Verification: What does VAD do and why is it important? (Answer: Detects speech vs silence, so the system knows when user finished talking)

Mini-Projects

Project 1: Install Pipecat and Run Example

Ask Claude Code to:

Requirements:

- Create a new project folder for Pipecat experiments
- Install Pipecat with required dependencies
- Set up .env with all API keys (OpenAI, Deepgram, ElevenLabs)
- Run the basic local audio example from Pipecat
-

Your job:

Review the example code structure

Verification Checklist:

- Pipecat installed successfully
- All dependencies resolved
- .env file configured
- Can import pipecat without errors

Project 2: Local Voice Bot (Microphone/Speaker)

Ask Claude Code to build a Pipecat bot that:

Requirements:

- Uses LocalAudioTransport (your mic and speakers)
- Pipeline: Microphone → SileroVAD → Deepgram STT → OpenAI GPT-4.1-mini → ElevenLabs TTS → Speaker
- System prompt: "You are a helpful assistant. Keep responses under 2 sentences."
- Handles interruptions (if you speak while AI is talking, it stops)

Your job:

1. Run the bot
2. Wait for it to initialize
3. Speak a question
4. Listen for the response
5. Try interrupting mid-response — does it stop?
6. Have a 5-turn conversation

Verification Checklist:

- Bot starts and listens
- Transcription works (you can add logging to see it)
- LLM generates response
- TTS plays through speakers
- Interruption handling works
- Latency feels conversational (under 2 seconds)

Project 3: Add SmartTurn for Better Turn Detection

Ask Claude Code to enhance the bot with SmartTurn:

Requirements:

- Add SmartTurn processor after VAD
- Configure for natural conversation flow
- Log when turns are detected vs simple silence

Your job:

1. Run bot without SmartTurn, note behavior
2. Run bot with SmartTurn, compare behavior
3. Test with sentences that have natural pauses (e.g., "I want to order... hmm... maybe a pizza")
4. Verify SmartTurn waits for you to finish vs interrupting at pauses

Verification Checklist:

- SmartTurn integrated
- Bot waits for complete thoughts
- Doesn't interrupt during thinking pauses
- Feels more natural than VAD alone

Project 4: Measure and Optimize Latency

Ask Claude Code to add latency tracking:

Requirements:

- Log timestamp when user stops speaking (VAD end-of-speech)
- Log timestamp when first TTS audio plays
- Calculate and display end-to-end latency
- Display latency after each exchange

Your job:

1. Run multiple conversations
2. Note average latency
3. Try GPT-4.1-mini vs GPT-4o, compare latency
4. Target: under 1.5 seconds end-to-end

Verification Checklist:

- Latency logging implemented
- Measured average latency
- Understand where latency comes from (STT, LLM, TTS)
- Latency under 2 seconds consistently

Project 5: Add Function Calling to Voice Bot

Ask Claude Code to add tools to your Pipecat bot:

Requirements:

- Add a "get_current_time" function
- Add a "tell_joke" function (returns a random joke)
- Add a "lookup_order" function (mock data — return fake order status)
- LLM decides when to call functions based on voice query

Your job:

1. Ask "What time is it?" — verify function is called
2. Ask "Tell me a joke" — verify joke is told
3. Ask "What's the status of order 12345?" — verify lookup works
4. Ask a general question — verify no function is called

Verification Checklist:

- Time function works via voice
- Joke function works via voice
- Order lookup works via voice
- Natural questions work without functions
- Function results are spoken naturally

End of Day 3 Checklist

- Understand Pipecat architecture (frames, processors, pipeline)
- Understand our stack: SileroVAD, SmartTurn, Deepgram, OpenAI, ElevenLabs
- Pipecat installed and running
- Local voice bot working (mic/speaker)
- SmartTurn integrated for better turn detection
- Interruption handling working
- Latency measured (target: under 2 seconds)

- Function calling working via voice

Day 4

Pipecat + Plivo — Phone Calls via WebSocket Audio Streaming

Goal: Connect your Pipecat voice bot to real phone calls using Plivo's WebSocket audio streaming.

Part 1: How Plivo Audio Streaming Works (Learn from Claude)

Ask Claude (browser) to explain:

Plivo Audio Streaming Flow:

```
None
Caller's Phone
    ↓
Plivo (receives call)
    ↓
WebSocket connection to your server
    ↓
Your Pipecat bot receives audio frames
    ↓
Pipecat processes: VAD → STT → LLM → TTS
    ↓
Audio frames sent back via WebSocket
    ↓
Plivo plays audio to caller
```

Key Concepts:

Audio Format from Plivo:

- Format: mulaw (μ-law) or linear16
- Sample rate: 8kHz (phone quality)
- Mono channel

Bidirectional WebSocket:

- Plivo sends audio TO your server
- Your server sends audio BACK to Plivo
- Real-time, full-duplex communication

Answer URL vs Stream URL:

- Answer URL: Plivo calls this when someone dials in, returns XML
- The XML tells Plivo to start streaming to your WebSocket URL

Verification: What happens when someone calls your Plivo number? (Answer: Plivo hits your Answer URL, gets XML instructions, then opens WebSocket to stream audio)

Part 2: Plivo Transport in Pipecat (Learn from Claude)

Ask Claude (browser) to explain:

PlivoTransport:

- Built-in Pipecat transport for Plivo audio streaming
- Handles WebSocket connection from Plivo
- Converts Plivo audio format to Pipecat frames
- Converts Pipecat audio frames back to Plivo format

Configuration Needed:

- WebSocket server endpoint (your server URL)
- Audio format settings (mulaw/linear16, sample rate)
- Answer URL that returns Plivo XML

Verification: What does PlivoTransport handle for you? (Answer: WebSocket connection, audio format conversion, bidirectional streaming)

Mini-Projects

Project 1: Basic Plivo WebSocket Server

Ask Claude Code to build a basic server that:

Requirements:

- Flask app with an /answer endpoint that returns Plivo XML
- XML tells Plivo to stream audio to a WebSocket endpoint
- WebSocket server that logs when Plivo connects
- Logs when audio packets arrive (don't process yet)

Your job:

1. Run the server locally
2. Start ngrok: ngrok http 5000
3. Configure Plivo phone number:
    - Go to Plivo console → Phone Numbers → your number

    - Set Answer URL to: https://your-ngrok-url/answer
    - Set Method to: POST
4. Call your Plivo number from your phone
5. Watch the logs — see WebSocket connect and audio packets

Verification Checklist:

- Flask server running
- ngrok tunnel active
- Plivo Answer URL configured
- Can call and see WebSocket connection
- Audio packets logging

Project 2: Pipecat Bot with Plivo Transport

Ask Claude Code to create a full Pipecat bot with Plivo:

Requirements:

- Use PlivoTransport for audio I/O
- Pipeline: PlivoTransport → SileroVAD → SmartTurn → Deepgram STT → OpenAI GPT-4.1-mini → ElevenLabs TTS → PlivoTransport
- System prompt: "You are a helpful phone assistant. Keep responses brief and conversational."
- Handle the Plivo audio format correctly (mulaw, 8kHz)
- Flask endpoint for /answer that returns streaming XML

Your job:

1. Run the Pipecat bot server
2. Start ngrok
3. Update Plivo Answer URL to your ngrok URL
4. Call your Plivo number
5. Have a conversation!

Verification Checklist:

- Bot answers phone call
- You hear a greeting (or bot waits for you to speak)
- Transcription works over phone
- LLM responds appropriately
- You hear TTS response clearly
- Multi-turn conversation works

Project 3: AI Receptionist

Ask Claude Code to build a complete AI receptionist:

Requirements:

System Prompt:

```
None
You are a friendly receptionist for Acme Corp.

When someone calls:
1. Greet them: "Hello, thank you for calling Acme Corp. How can I help you today?"
2. Listen to their request
3. If they want sales: Say "I'll connect you to our sales team. One moment please." then end gracefully.
4. If they want support: Say "I'll connect you to support. Can you briefly describe your issue?"
5. If they ask about hours: Say "We're open Monday to Friday, 9 AM to 5 PM Pacific time."
6. If they ask about location: Say "We're located at 123 Main Street, San Francisco."
7. If unclear: Say "I'm sorry, could you repeat that?"

Keep responses brief and natural. Don't be robotic.
```

Function Calling:

- get_business_hours() - returns hours
- get_location() - returns address
- transfer_to_sales() - logs intent, says transferring
- transfer_to_support() - logs intent, says transferring

Logging:

- Log each call to Vercel Postgres (from Week 1)
- Store: caller_number, transcript, detected_intent, duration, timestamp

Your job:

1. Deploy with ngrok
2. Test all scenarios:
    - Call and ask about sales
    - Call and ask about hours
    - Call and ask about location
    - Call and say something unclear
3. Check Vercel Postgres for logs after each call

Verification Checklist:

- Answers with greeting
- Sales intent handled correctly
- Support intent handled correctly
- Hours FAQ works

- Location FAQ works
- Unclear queries handled gracefully
- All calls logged to Postgres with transcript

Project 4: Improve Conversation Quality

Ask Claude Code to add these enhancements:

Requirements:

- Better interruption handling: Stop TTS immediately when caller speaks
- Conversation context: Remember what was discussed earlier in the call
- Graceful ending: "Is there anything else I can help you with?" → "Thank you for calling. Goodbye!"
- Error recovery: If STT fails, say "I'm having trouble hearing you, could you repeat that?"

Your job:

1. Test interruption: Start speaking while bot is talking
2. Test context: Ask a follow-up question ("What about weekends?" after asking about hours)
3. Test ending: Say "That's all, thanks"
4. Test error: Make noise/mumble, see if it recovers

Verification Checklist:

- Interruptions stop TTS immediately
- Bot remembers context within call
- Ending is natural
- Errors handled gracefully

End of Day 4 Checklist

- Understand Plivo WebSocket audio streaming
- Basic WebSocket server receiving Plivo audio
- Pipecat bot with PlivoTransport working
- Can have phone conversations with AI
- AI Receptionist with intent detection working
- Function calling working (hours, location, transfers)
- Call logging to Vercel Postgres working
- Conversation quality improvements implemented

Day 5

LiveKit with Plivo SIP Trunking — Alternative Approach

Goal: Learn LiveKit as an alternative to Pipecat, using Plivo SIP trunking for phone connectivity.

Shorts to Watch First

- WebRTC in 100 Seconds - Fireship (https://youtube.com/watch?v=WmR9IMUD_CY)

Part 1: LiveKit vs Pipecat (Learn from Claude)

Ask Claude (browser) to explain:

Two Different Approaches:

Pipecat + Plivo WebSocket (Day 4):

- Plivo streams raw audio over WebSocket
- Your Pipecat server processes audio directly
- You manage the audio pipeline
- More control, more code

LiveKit + Plivo SIP (Day 5):

- Plivo forwards calls via SIP trunk to LiveKit
- LiveKit handles WebRTC/media
- Phone caller appears as a "participant" in a LiveKit room
- Your AI agent is another "participant"
- LiveKit manages audio routing
- Less code, more abstraction

When to Use Each:

Use Pipecat + Plivo WebSocket when:

- You want full control over audio processing
- You need custom audio manipulation
- You're building on Plivo's infrastructure

Use LiveKit + SIP when:

- You want simpler architecture
- You also need browser/mobile WebRTC support
- You want LiveKit's built-in features (recording, rooms, etc.)

Verification: What's the main difference between these approaches? (Answer: Pipecat handles audio directly; LiveKit treats phone as a room participant)

Part 2: LiveKit Fundamentals (Learn from Claude)

Ask Claude (browser) to explain:

LiveKit Concepts:

- Room: Virtual space where participants connect
- Participant: Anyone in a room (human, bot, phone caller)
- Track: Audio or video stream
- Token: JWT for authentication

LiveKit Cloud vs Self-Hosted:

- LiveKit Cloud: Managed service, free tier, no setup
- Self-hosted: Run your own (more complex)
- For learning: Use LiveKit Cloud

SIP Trunking with LiveKit:

- Plivo SIP trunk connects to LiveKit
- Incoming phone call → LiveKit creates participant
- Your agent joins same room
- LiveKit routes audio between them

LiveKit Agents Framework:

- LiveKit's own agent framework (similar to Pipecat)
- Integrates with Deepgram, OpenAI, ElevenLabs
- Handles VAD, turn detection, pipeline

Verification: In the LiveKit model, what is a phone caller? (Answer: A participant in a LiveKit room, just like a browser user)

Mini-Projects

Project 1: Set Up LiveKit Cloud

Your job (manual setup):

1. Go to cloud.livekit.io
2. Create a new project
3. Note your:
    - API Key
    - API Secret
    - WebSocket URL (wss://your-project.livekit.cloud)
4. Add to your .env:

```
None
LIVEKIT_API_KEY=your-api-key
LIVEKIT_API_SECRET=your-api-secret
LIVEKIT_URL=wss://your-project.livekit.cloud
```

Ask Claude Code to create a token generation script:

Requirements:

- Generate a LiveKit access token for a room
- Token allows publishing and subscribing audio
- Print the token

Verification Checklist:

- LiveKit Cloud account created
- API credentials saved in .env
- Token generation script works

Project 2: LiveKit Agent (Browser Test First)

Ask Claude Code to create a LiveKit agent using LiveKit Agents framework:

Requirements:

- Use LiveKit Agents SDK (not Pipecat)
- Pipeline: VAD → Deepgram STT → OpenAI GPT-4.1-mini → ElevenLabs TTS
- Agent joins a room and waits for participants
- When someone joins and speaks, agent responds
- System prompt: "You are a helpful assistant. Keep responses brief."

Your job:

1. Run the agent
2. Open LiveKit Playground: https://agents-playground.livekit.io/
3. Connect to your LiveKit project
4. Join a room
5. Speak to the agent
6. Verify you hear responses

Verification Checklist:

- Agent running and connected to LiveKit
- Can join room from browser playground
- Agent hears your speech
- Agent responds with voice
- Latency is acceptable

Project 3: Configure Plivo SIP Trunk to LiveKit

Your job (manual setup with Claude's guidance):

Ask Claude (browser) to explain the steps, then do them:

1. In LiveKit Cloud dashboard:
    - Go to SIP settings
    - Create an inbound SIP trunk
    - Note the SIP URI (something like sip:...@livekit.cloud)
2. In Plivo console:
    - Create a SIP endpoint pointing to LiveKit's SIP URI
    - Configure your Plivo phone number to forward to this SIP endpoint
3. Configure LiveKit to dispatch calls to your agent:
    - Set up dispatch rules so incoming SIP calls create rooms
    - Your agent auto-joins when a room is created

Ask Claude Code to help update configuration files as needed.

Verification Checklist:

- LiveKit SIP trunk configured
- Plivo SIP endpoint created
- Plivo number forwards to SIP endpoint
- Test call reaches LiveKit

Project 4: Phone to LiveKit Agent

Your job:

1. Ensure your LiveKit agent is running
2. Call your Plivo number from your phone
3. The call should:
    - Go to Plivo
    - Forward via SIP to LiveKit
    - Create a room with you as participant
    - Your agent joins the room
    - You can talk to the agent!

Verification Checklist:

- Phone call connects via SIP
- Agent responds to phone caller
- Multi-turn conversation works
- Audio quality acceptable

Project 5: LiveKit AI Receptionist

Ask Claude Code to build the same receptionist but using LiveKit Agents:

Requirements:

- Same functionality as Day 4 receptionist
- Greeting, intent detection (sales/support/FAQ)
- Function calling for hours, location
- Log calls to Vercel Postgres
- Use LiveKit Agents framework (not Pipecat)

Your job:

1. Deploy the agent
2. Test via phone call
3. Compare to Pipecat version — any differences?
4. Verify logging works

Verification Checklist:

- Receptionist works via phone
- Intent detection works
- FAQs work
- Logging to Postgres works
- Can compare to Pipecat approach

Part 3: Comparing Approaches (Learn from Claude)

After completing both Days 4 and 5, ask Claude to help you compare:

Pipecat + Plivo WebSocket:

- Pros: Full control, works with any transport, flexible
- Cons: More code, manage audio pipeline yourself

LiveKit + Plivo SIP:

- Pros: Simpler, LiveKit handles media routing, easy browser support
- Cons: Depends on LiveKit, less control

For Plivo customers:

- Pipecat approach is more direct (Plivo → your server)
- LiveKit approach adds a layer but simplifies browser support

Verification: Which approach would you use for a production Plivo voice AI? (Answer: Depends on requirements — Pipecat for control, LiveKit for simplicity and browser support)

End of Day 5 Checklist

- Understand LiveKit architecture (rooms, participants, tracks)
- Understand difference between Pipecat and LiveKit Agents
- LiveKit Cloud account set up
- LiveKit agent working via browser
- Plivo SIP trunk configured to LiveKit
- Phone calls working via LiveKit SIP
- AI Receptionist working on LiveKit
- Can compare both approaches (Pipecat vs LiveKit)

Day 6

Deployment with Railway — Make It Public

Goal: Deploy your voice AI agent to Railway so it runs 24/7 without your laptop.

Part 1: Why Railway? (Learn from Claude)

Ask Claude (browser) to explain:

The Problem: Vercel Won't Work for Voice AI

Vercel Limitations:

- Serverless functions timeout after 10 seconds (hobby) or 60 seconds (pro)
- No persistent WebSocket connections
- Can't handle long-running processes

Voice AI Requirements:

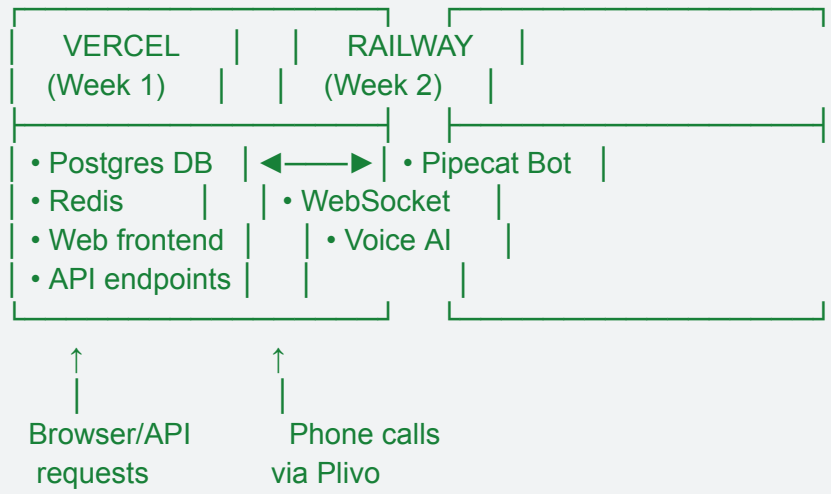
- Phone calls last 2-10 minutes
- Need persistent WebSocket connection for entire call
- Bot must run continuously, waiting for calls

Why Railway Works:

- Runs persistent processes (like a traditional server)
- No timeout limits
- WebSocket connections stay open indefinitely
- Your bot runs 24/7
- Free tier: $5/month credits (enough for learning)

Architecture:

None



Verification: Why can't we deploy Pipecat to Vercel? (Answer: Vercel has timeout limits; voice calls need persistent connections lasting minutes)

Part 2: Railway Basics (Learn from Claude)

Ask Claude (browser) to explain:

Railway Concepts:

- Project: A collection of services
- Service: A single app/container (your bot)
- Deployment: A version of your service
- Variables: Environment variables

Railway vs Vercel:

Vercel:

- Best for: Frontends, short API calls
- Model: Serverless (spin up/down per request)
- Timeout: 10-60 seconds

Railway:

- Best for: Backends, long-running processes
- Model: Container (runs continuously)
- Timeout: None

Verification: What's the main difference between Vercel and Railway? (Answer: Vercel is serverless with timeouts; Railway runs containers continuously)

Mini-Projects

Project 1: Set Up Railway Account and CLI

Your job:

1. Go to railway.app
2. Sign up with GitHub
3. Install Railway CLI:

```
None
npm install -g @railway/cli
railway login
```

4. Verify login:

```
None
railway whoami
```

Verification Checklist:

- Railway account created
- CLI installed
- Logged in via CLI

Project 2: Prepare Pipecat Bot for Railway

Ask Claude Code to prepare the project:

Requirements:

- Create a Dockerfile for the Pipecat bot
- Ensure all config is via environment variables
- Add a /health endpoint for monitoring
- Create requirements.txt with all dependencies
- Ensure the server binds to 0.0.0.0 and uses PORT env var

Your job:

1. Review the Dockerfile
2. Verify requirements.txt is complete
3. Test locally with Docker (optional):

```
None
docker build -t pipecat-bot .
docker run -p 5000:5000 --env-file .env pipecat-bot
```

Verification Checklist:

- Dockerfile created
- requirements.txt complete
- Health endpoint works
- Bot runs in container (if tested)

Project 3: Deploy to Railway

Your job:

1. Initialize Railway project:

```
None
cd your-pipecat-project
railway init
```

2. Link to project:

```
None
railway link
```

3. Set environment variables:

```
None
railway variables set OPENAI_API_KEY=sk-...
railway variables set DEEPGRAM_API_KEY=...
railway variables set ELEVENLABS_API_KEY=...
railway variables set PLIVO_AUTH_ID=...
railway variables set PLIVO_AUTH_TOKEN=...
railway variables set POSTGRES_URL="postgres://..."
```

4. Deploy:

```
None
railway up
```

5. Get your public URL:

```
None
railway domain
```

(Or go to Railway dashboard → Settings → Generate Domain)

6. Check logs:

```
None
railway logs
```

Verification Checklist:

- Railway project created
- Environment variables set
- Deploy successful
- Got public URL (e.g., https://your-app.up.railway.app)
- Logs visible

Project 4: Update Plivo to Use Railway

Your job:

1. Copy your Railway URL
2. Update Plivo phone number:
    - Go to Plivo console → Phone Numbers → your number
    - Set Answer URL to: https://your-app.up.railway.app/answer
    - Set Method to: POST
    - Save
3. Test with a phone call:
    - Call your Plivo number
    - Talk to your AI receptionist
    - Verify it works!
4. Watch logs while calling:

```
None
railway logs --tail
```

Verification Checklist:

- Plivo pointing to Railway URL (not ngrok!)
- Phone call connects
- Voice AI responds
- Logs show call activity

Project 5: Verify Database Logging

Your job:

1. Make a few test calls to your Railway-deployed bot
2. Check Vercel dashboard → Storage → Postgres → Data tab
3. Verify calls are logged with:
    - Caller number
    - Transcript
    - Detected intent
    - Duration

Verification Checklist:

- Calls logged to Vercel Postgres
- Transcript saved correctly
- Intents detected correctly
- Duration recorded

Project 6: Deploy LiveKit Agent to Railway (If Using LiveKit)

If you built the LiveKit version on Day 5, deploy that too:

Ask Claude Code to prepare the LiveKit agent for Railway:

Requirements:

- Dockerfile for LiveKit agent
- Environment variables for LiveKit credentials
- Health endpoint

Your job:

1. Deploy to Railway (same process as Pipecat)
2. Ensure SIP trunk still works with Railway-hosted agent
3. Test phone calls

Verification Checklist:

- LiveKit agent deployed to Railway
- SIP calls still work
- Agent responds to phone calls

End of Day 6 Checklist

- Understand why Railway is needed (vs Vercel)
- Railway account created
- Railway CLI installed and working

- Pipecat bot deployed to Railway
- Plivo pointing to Railway URL (not ngrok!)
- Phone calls working 24/7
- Call logs saved to Vercel Postgres
- No local processes needed — everything in cloud
- (Optional) LiveKit agent also deployed

Day 7

Polish & Capstone Demo

Goal: Polish your AI receptionist, add final features, and record a demo video.

Final Enhancements

Ask Claude Code to add these final features:

Requirements:

1. Better interruption handling:
    - Stop TTS immediately when caller speaks
    - Don't restart from beginning, continue naturally
2. Conversation memory:
    - Remember context within the call
    - "As I mentioned earlier..." type references
3. Graceful ending:
    - After handling request: "Is there anything else I can help you with?"
    - When done: "Thank you for calling Acme Corp. Have a great day!"
4. Error handling:
    - If STT fails: "I'm having trouble hearing you, could you repeat that?"
    - If LLM fails: "One moment please..." then retry
    - If TTS fails: Fallback to simpler response
5. Call summary:
    - At end of call, generate a 1-2 sentence summary
    - Save summary to database along with transcript

Your job:

1. Implement each enhancement
2. Test thoroughly:
    - Make 10+ test calls
    - Test interruptions
    - Test context memory
    - Test error scenarios (speak gibberish, make noise)
3. Redeploy to Railway:

```
None
railway up
```

Verification Checklist:

- Interruptions handled smoothly
- Context remembered within call
- Calls end gracefully
- Errors handled without crashing
- Summaries saved to database

Capstone: Record Demo Video

Your job:

1. Record a screen capture with audio showing:
    - You dialing your Plivo number
    - Phone ringing and AI answering
    - You asking about sales → AI responds appropriately
    - You asking about hours → AI gives FAQ answer
    - You asking a random question → AI handles it
    - Natural ending of the call
    - Full conversation (2-3 minutes)
2. Show the data:
    - After the call, open Vercel dashboard
    - Show the database entry
    - Show the transcript
    - Show the detected intent
    - Show the call summary
3. Recording tools:
    - QuickTime (Mac): File → New Screen Recording
    - OBS (free): For more control
    - Loom: Easy sharing

Demo Script Example:

```
None
[Show phone dialing Plivo number]

AI: "Hello, thank you for calling Acme Corp. How can I help you today?"

You: "Hi, I'm interested in learning about your products."
```

AI: "I'd be happy to connect you to our sales team. Before I do, may I ask what product you're interested in?"

You: "Actually, first can you tell me what your hours are?"

AI: "Of course! We're open Monday to Friday, 9 AM to 5 PM Pacific time. Would you still like to speak with sales?"

You: "No, that's all I needed. Thanks!"

AI: "You're welcome! Is there anything else I can help you with?"

You: "Nope, bye!"

AI: "Thank you for calling Acme Corp. Have a great day!"

[Show Vercel dashboard with call log]
[Show transcript]
[Show detected intent: sales → faq → end]
[Show summary: "Caller inquired about sales, then asked about business hours. No transfer needed."]

Verification Checklist:

- Video recorded
- Shows complete conversation
- Shows database logs in Vercel
- Audio quality is clear
- Demo is under 5 minutes

Share with Someone

Your job:

1. Send your Plivo phone number to a friend or colleague
2. Ask them to call and talk to your AI receptionist
3. Don't tell them what to say — let them interact naturally
4. Check logs to see their call
5. Get feedback:
    - Did it feel natural?
    - Was latency acceptable?
    - Did it understand them?
    - Any issues?

Verification Checklist:

- Someone else successfully called
- Bot worked for external caller
- Call logged correctly
- Got feedback
- Noted any issues to fix

Week 2 Complete!

By the end of Week 2, you can:

- Call OpenAI APIs with streaming
- Transcribe audio in real-time with Deepgram
- Generate speech with ElevenLabs (streaming)
- Build voice AI pipelines with Pipecat (SileroVAD, SmartTurn)
- Connect phone calls via Plivo WebSocket audio streaming
- (Alternative) Use LiveKit with Plivo SIP trunking
- Deploy voice AI to Railway for 24/7 availability
- Understand when to use Vercel vs Railway
- Build a complete AI receptionist that handles intents and FAQs
- Log calls with transcripts to a database

Architecture Summary

Option A: Pipecat + Plivo WebSocket

```
None
Phone Call
   ↓
Plivo
   ↓ (WebSocket audio streaming)
Railway (Pipecat Bot)
   ├── SileroVAD
   ├── SmartTurn
   ├── Deepgram STT
   ├── OpenAI GPT-4.1-mini
   └── ElevenLabs TTS
   ↓
Vercel Postgres (call logs)
```

Option B: LiveKit + Plivo SIP

```
None
Phone Call
   ↓
Plivo
   ↓ (SIP Trunk)
LiveKit Cloud
   ↓ (Room participant)
Railway (LiveKit Agent)
   ├── VAD
   ├── Deepgram STT
   ├── OpenAI GPT-4.1-mini
   └── ElevenLabs TTS
   ↓
Vercel Postgres (call logs)
```

End of Week 2 Checklist

- OpenAI API working with streaming
- Function calling working
- Deepgram STT working (real-time)
- ElevenLabs TTS working (streaming)
- Pipecat with SileroVAD and SmartTurn working
- Pipecat + Plivo WebSocket working
- (Optional) LiveKit + Plivo SIP working
- AI Receptionist with intent detection working
- Deployed to Railway (24/7)
- Call logs saved to Vercel Postgres
- External person tested the system
- Demo video recorded