# Optimizing AI Voice Agents

28 MIN · YOUTUBE · HTTPS://WWW.YOUTUBE.COM/WATCH?V=I86DFIVLZXY
https://www.youtube.com/watch?v=I86dFivLzXY

SUMMARY

*The discussion focuses on building and evaluating real-time voice AI agents, highlighting the architecture, challenges, and practical implementations using the Pipecat framework. The speaker shares insights from their work at Daily and emphasizes the importance of managing conversation context and response times in voice interactions.*

- *Voice agents operate in a multi-turn loop, converting audio to text, processing it, and generating audio responses.*
- *Key challenges in voice AI include maintaining low response times (ideally under 800 milliseconds) and managing complex multi-turn conversations.*
- *The speaker introduces Pipecat, an open-source project designed to simplify the creation of real-time voice agents.*
- *Evaluation of voice agents is complicated by the need to assess multiple models (e.g., transcription, LLM, voice generation) and their performance across conversation turns.*
- *The architecture can vary from using bundled platforms to custom solutions, with many developers opting for hosted APIs for transcription and LLMs.*
- *Observability and data logging are crucial for debugging and improving voice agents, with suggestions for using tools like OpenTelemetry.*
- *The speaker discusses common failure modes, particularly around instruction following and function calling, which tend to degrade in longer conversations.*
- *The importance of iterative development and experimentation in optimizing voice AI performance is emphasized throughout the talk.*

All right, I think we're set. first, thank you so much for having me. I'm a big fan of what everything that's going on in this course. I follow everything you all do. I'm a little behind on some of the sessions, but I'm going to catch up. and I'm a disciple of the look at your data model. as you said, I work at a company called Daily. We make global infrastructure for real-time audio and video. A lot of the use cases these days are real-time voice and real-time video conversational AI agents. I also work on an open source vendor neutral project called Pipecat, which came out of the work we started doing about the time GPT4 was first released two years ago to make it easier to build real-time agents. I wrote a guide that's online at voiceAI andvoiceagents.com where we tried to write down like everything we'd learned in the last two years building real time voice AI. I'm gonna reference that guide a couple times as I talk today to try to avoid going down rabbit holes that I would normally want to go down and geek out about. so I want to talk for about 15 minutes and show a couple demos and then leave time for questions. And I made a repo for this class. it's github.comquendla-course.

Maybe somebody can drop it in the chat. The code I'm going to show is all in that repo. And it was super great to have this session as a forcing function to clean up code enough to share it publicly that I have you know in

bits and pieces floating around in various temporary directories on my machine. so let's just talk a little bit about voice architecture because if you haven't built voice agents before it's worth understanding kind of what we do in a typical voice agent. This is the block diagram for a single conversation turn. we run in a loop because a voice conversation is multi-turn and as you know from building with LLMs LLMs are stateless so you have to give the conversation history to the LLM each turn you may do things like modify and summarize the conversation in flight those are slightly more advanced topics which we won't cover today but we can talk about them in the Q&A but the basic block diagram is audio comes in from the user you turn that audio into text In parallel, you do turn detection to know when the user is done talking and you should trigger inference and have the voice agent respond. that user text needs to go into whatever context management logic you're using so that you're tracking it. the LLM inference comes out as text and goes into a texttospech or a voice generation model that generates the audio that the user actually hears. Again, you have to put the text or the audio or both in your context management logic. And then you run that loop again when you get new user input. Every voice agent in production looks like this. We're not going to talk about new speechtoech models today. although again we can talk about them in the QA if people are super interested. The core principles are the same. The the block diagram looks a little different. But I'll just say that 99% or more of production voice AI bots today look roughly like this.

And here's what this looks like in code. This is actual Pipecat code. this is from a fully working demo. this is that block diagram. But in Python code, there's, you know, 50 lines of imports and 30 lines of event handlers after this. But this is the core of a production voice agent. In production though, agents often get a lot more complicated. So here's a pretty typical medium complexity pipeline. You can see we've got actually two parallel pipelines going on here. One is the main conversation inference. The other is a kind of LLM as a judge that's happening in a second pipeline. It needs to be fast but not real time is the way to think about that. And this is a pretty common emerging pattern as use cases get more complicated. you have multiple inference loops going on for at least some parts of a conversation that complicates evals complicates how you think about the data how you organize like things like open telemetry traces so things are getting more and more complicated and we're in some ways getting farther behind in being able to do good eval sort of very high level thing when you build voice agents I tell people sort of you need to think about these three things you have to write the You have to deploy the code somewhere.

It has to run in the cloud somewhere. and then you have to connect users over the network to that code running in the cloud so you can get the audio or the audio and video or the audio and video and images and text in and out of that voice loop. you these days have a lot of good options actually for getting started. There are some platforms that bundle all of this together. A couple companies that I work closely with that do this are Vappy and Phonely. You may have heard of one or both of those.

they give you this full package, this full all three of these things bundled together with a bunch of other stuff. Makes it really easy to get started. Is a great solution if what you are doing matches the design and the features of that platform. You can also go all the way the other direction, write all the code yourself and then decide how to deploy and host and autoscale and monitor it and maintain your own network infrastructure for connecting, you know, websockets or WebRTC to those bots. you have lots of options for mixing and matching or for going with the sort of bundled approach.

So, very quickly, I'm not actually going to go over everything on this slide, but just a few placeholders for you to pull on this threads later if you're interested. So, what's hard about voice AI? The the big thing is everything has to happen fast. You have pretty strict voiceto response time requirements or users will just not use a voice agent. I tell people aim for 800 milliseconds voicetovoice response on the client side. That's everything. That's all the LLM inference plus all the network transport. 800 milliseconds is really good. a little faster is even better. A little slower is okay. Much slower is probably not okay. it does depend a little bit on your use case, but you know, 800 millisecond. Yeah, you have to have a stake in the ground for these things.

800 millonds is a good stake in the ground. a couple other things just to highlight here. You're always using multiple models for voice AI and multiple models as we talked about a little bit complicates how you think about eval. Do you evaluate the transcription step and then the inference step and then the voice step separately? Do you evaluate the whole thing end to end? I mean obviously you need to do both. so you have to figure out how to think about kind of the collection of data for each turn and for a full conversation. Also everything's multi-turn.

multi-terney valves are another layer of complexity. And in fact, one super interesting thing to me kind of philosophically because I I get to work with kind of everybody in the space at kind of every level. so we work a lot with the Deep Mind team, we work a lot with OpenAI. long multi-turn, long context in general and audio modality all take you out of distribution for how the big models are trained today. practically speaking, what that means is that instruction following and function calling are worse for voice AI use cases than they are on any of the published benchmarks you will see for any models. So, your function calling may work great for the first three turns of the conversation or five turns or seven turns of the average conversation and then your function calling and instruction following will fall off a cliff at some point for every model today, even the very very best ones. And I think that's a function of two things. Partly it's just model architecture. It's like how like what the compromises that are being made on the transformer tweaks, the model architectures, the inference stacks, all that stuff. We are not the most important use case for this stuff.

Shrea, do you want to jump in? Yeah, I have a question. how long is long context? What does that mean in the voice case? Do you mean it's context that you're retrieving as part of rag given some voice snippet or like it's like people giving a soliloquy of five minutes and that becomes long context? So it's two things. It's not usually a siloquy for most use cases. Although there are some voice use cases where people talk and talk and talk dictation style and then you do something with it.

But for most use cases it's two things. One is the easiest way to do rag for voice AI is to pull a bunch of stuff into the first part of the context at the very beginning of the conversation. So often you're adding you know 50,000 tokens or something at the beginning of the conversation if you can do it like if your eval survive it right. The other thing is that there are a lot of voice AI use cases where there's a 20 minute or a 30 minute conversation like think of a customer support call where you're on the phone a long time trying to get help in a customer support context. So the multi-turn you're just adding like n if you do that naively you are adding a user input and an LM output message every single conversation turn and that may not add a huge number of tokens relative to like the theoretical context length of today's LLMs but it definitely adds some tokens and it also adds

like a lot of latent space complexity just empirically. So you have these workflows where you've got a lot of user assistant, user assistant, user assistant, user assistant messages and some function calls and that just makes it hard for the LLMs just based on what we've seen. Yeah, that's really interesting. It's different from a lot of the stuff we've been teaching in the course of, you know, it's it's not like 200 back and forth turns in practice. Do people actually have all of these back and forth turns or you try to like compress it? compress if not the right word but condense it perhaps into one term turn where you say like system user assist whatever it is just to make it be less multi-turn if that makes any sense it totally makes sense so we started just saving the whole context naively we've started to move all of us in the voice AI ecosystem over the last few months towards figuring out how to do focusing compression like processing at particular moments in the conversation with specific summarization goals in mind. So there's a there's a pattern I really like that is getting a lot of use now that I think of as a state machine. So model the conversation as a series of states. Every state is a system instruction, some transformation of the entire conversation context so far, a set of tools that you're going to use at that point in the conversation, and a set of exit nodes to whatever legal other next states there are. That tends to really help, at least me, think about how to focus the context so that the LLM does what you want. I mean, if you're stepping back a little bit and trying to figure out what you're trying to do here, LLMs are amazing conversational partners. like that's what has opened up this voice capability. You you really want to leverage the fact that the LMS can have an open-ended conversation. On the other hand, for most enterprise use cases, you have a specific set of things you need to get done in a specific session.

There's some tension between those like letting the LLM do its conversational thing and making sure you get the business logic like check boxes all checked off. a state machine abstraction kind of lets you try to find that sweet spot where you're getting the best of both worlds and like everything like everything you're teaching in this class it takes a bunch of iteration but at least if you can swap out all those four pieces you can model it does that make sense yeah I think that's super helpful and also grounding and kind of what we've talked about so far in the course thanks for sharing there's a library called pipcat flows which is a communitymaintained state machine abstraction of exactly this type built on top of pipecat if want to sort of look at how people are building this this kind of thing. And I think you're starting to see that that abstraction in the nonvoice space as well. There's a lot of stuff that kind of crosses the boundaries between the voice agents and the non-voice agents.

So, what are we going to try to do here? We're going to try to look at our data and listen to it. And I'll just say that there are several observability integrations in Pipcat already. You in production probably want to use one of these. Brooke from Kovalt is talking next, I believe. I am a huge fan of Brook's platform and the work they do on kind of simulationdriven evaluation of voice AI, but we're not going to use any of these today because I'm also a big fan of what Hammel has said in every context I've bumped into him in, including when he came to our voice course, which is like get your hands dirty. Extract the data yourself in whatever format you think it might be useful. Write some ways of looking at I love Shrea's post on Twitter over the weekend maybe that you should vive code web frontends to look at your data which I totally should be doing. so we're going to do that today. and then I will say I do think you should use like somebody's platform as they as the platforms add better and better audio support in production. But I also think you should start by like figuring out what that data looks like yourself.

so here's the GitHub repo again. And this repo has three bots in it. I we it's everything's agents in 2025, but

we've been doing this long enough that we just call these bots. so there's like bot py files. There's a basic bot that just logs to the console. There's a bot that has minimal additional code. It's like 10 lines of code or something to to push an open telemetry collector. And then there's a bot that you we sort of started more from scratch and like extracted just the data we wanted from the conversation and put it somewhere and and tried to figure out how to look at it. we'll skip the demo of just the bot that goes to the console because like you've all seen a voice agent at this point. We'll jump straight to the open telemetry bot because a video is worth a thousand words. this is the diff though of between bot 01 and bot2 that adds open telemetry support. It doesn't take much to add the tracing. I say that just because like there's no excuse for not putting this data somewhere. And I'm speaking to myself cuz I usually write some stuff and actually sometimes deploy it to production and then go back and add the open telemetry tracing. Don't be like me. Be better than me. here is a little example. I'm right here and ready to help you with anything you need. Audio coming through. Thanks. I think you got the initial greeting wrong. We'll come back to that.

Oops. My apologies for that. I'm here and ready to help in any way I can. Let's check and see if we've got your traces in hotel. It sounds like we're on an intriguing mission. I'm going to scan through the grand chandeliers and ornate carpets of the hotel for traces. So, I have no idea where that came from. I'm actually really glad I captured that because that was GPT40 going totally off the rails. We could dig in and try to figure out why or how often that happened and we will come back to that. But that was that was super weird and not a normal interaction. but we can look in the traces and just sort of see that in the Langfuse dashboard, which is the hotel collector I was using here, that is in fact the conversation we just had. If you've used open telemetry, so there's nothing in your prompt about hotels and chandeliers. No, no. Like this is one of those one in 20 or one in 40 or you know good to be able to characterize it right of it it it misheard what I said some combination of bad transcription background noise GPT40 weirdness does happen sometimes even if the transcription is clean. so I was actually kind of glad I I caught that one. That's really surprising because you know you're using a really high quality microphone and your audio is very clear at the moment. Yeah. So, this was late last night. I was on my Linux box. I definitely had like a slightly weird sample rate thing going on with my input microphone. I mean, you know, I don't know. It was a local connection, though, so it shouldn't have been a network issue. but we'll come back to it. yeah, just making sure I have the right slides. Yes. Okay. so let's jump into doing something similar to that, but more from scratch. There's more code here, so I won't show code, but you can look at o3-bot-sqlite.py for the code. It's probably another hundred lines of code to save exactly the data I wanted to save. And here's the list of things that I was like, oh, I want to save this stuff because this is the kind of thing I always need to look at either in an individual debugging session like what just why did that bot just do that or aggregates that I care about across a bunch of a bunch of sessions. So like a a unique session ID, the turn number in the conversation, the turn start time, the turnedin time, the user text, the LLM text, the voicetovoice response time because latency matters so much, and whether the user interrupted the bot before it was done talking. So it's a pretty good basic set. Plus, we'll save the entire wave file so that for any chunk of the conversation, we can sort of play back the audio as we heard it. And up in the top right is the SQLite database creation command. I ran from from bash to to set this up. and then I vibe coded some look at your data scripts.

HML posted today about like getting over the the chasm of like going from oh I I used AI to to to write some code to I used AI to write some code. I'm trying to get there myself as a lifelong programmer with complicated feelings about code as craft. Claude wrote these scripts for me. It did a pretty good job. There was a little bit of

multi-turn back and forth, but it actually wrote the scripts for me last night at like midnight. And I didn't even change the file names here. So, I might have chosen different file names, but pretty good job, Claude. So, let's look at these three scripts.

We're going to do a quick conversation first just to something in the database that we recognize. can you tell me a joke? This my go-to. Why don't scientists trust atoms? Because they make up everything. Oh, that's a classic. how about the one about the janitor jumps out of the closet? Sure. When the janitor jumps out of the closet, he shouts, "Supplies." I love that one. Thank you. You're welcome. I'm glad. So, the terminal will get bigger in a second. I should have made it bigger earlier but so this is the first script that just dumps all the convers a list of all the conversations or gives you console friendly output of an individual conversation. So this is the conversation we just heard in a console friendly or you know terminal friendly format. Obviously if you can get it in terminal friendly format then you can do all kinds of things with this data. and you can see that's the conversation we just had. The next thing we'll do is actually play a snippet.

We'll actually play a snippet of a previous conversation that really a good suggestion for a break. How about a quick stroll outside for some fresh air? Or maybe just a fiveminute dance party to your favorite song. I love a dance party. And now we'll use that same script we just used before to actually look at the text mode version of that conversation and find that particular turn. apologies for the sample rate issue on my microphone. I didn't go back and re-record it once I heard that. you can see the whole conversation here. All these test conversations were pretty short. and then you can see the exact phrase we just heard. That's turn four in this particular conversation.

And then the last thing we'll do is go back to that like not following system instructions properly in the very first turn that we heard in the first video I played. I asked Claude to write a LLM as a judge script to do a very specific test. look at the first LLM response and judge whether it followed my system instruction. Again, the source code for all this stuff is in the repo. We're running that script. We have 22 like conversations in this test database that happened two times. We can go back and look at the individual times when that happened. One where my mouse is there actually. Yeah, that's my It looks like it was just a contraction. Maybe that's okay. you can decide what's okay and not, of course, in your emails. The second one, though, is quite a bit worse. So, there's the session ID. We could look at that whole conversation, see if starting off wrong makes you go wrong, all that kind of stuff. the second one is worse. It said not only the phrase I asked it to say, it said, "What can I do for you today?" which I ask it explicitly to only say that phrase in the system instruction. And this is something that I actually see pretty commonly with GPT40. If you ask it to say something specific, it will refuse some percentage of the time.

I'll stop my screen share. so this is a real world example of something you want to do an eval on. if you ask GBT4 to say something relatively specific, at some point in the conversation, you want to know if it actually does that. It might not. I I think it's an outgrowth of GBT40 safety stuff, not wanting to be prompted very very very specifically, although I don't know 100%. But I do know that it happens. This particular prompt I use at the beginning of the conversation, it happens about 10% of the time because I got clawed to vibe code an ealfcript.

That's all I got. That's really great. I mean, yeah, I really love the fact that you shared the code and you have this demo. That's really useful for everyone. I'll be going through it, too. It looks like a lot of fun, actually. Super fun. Voice is so fun. may we have some time for questions actually, so let's Yeah, this is great. Stephen has a great question. Well, Stephen thinks it's a bad question, but I don't think it's a bad question.

so he said, "Why why would the first turn ever be generated by an LLM if you want it to say the same thing for the first time and just for that first?" Yeah. No, that's totally right. If I really really really wanted to say exactly the same phrase, I would probably prompt the speech model, the voice generation model rather than the LLM. often you want it to say something approximate. So there's a looser version of this test that I think is real world relevant. the real reason is I'm really lazy and when I write demos, I often just like stick it in a user prompt rather than in in Pipcat, you stick a text generation frame in the pipeline. It goes through the pipeline. I could do that. I probably should start doing that.

but I often ask the LM to say something specific because I'm lazy. Do other folks have questions or I'm going to ask my own questions. So, I encourage people to unmute themselves, raise their hand maybe. All right. Well, I'll go. So, some rapid fire questions. What are the What are some types of failure modes that people do not expect to get out of voice agents that they see when they start looking at their data? I I do think it's the the single biggest thing I see is that instruction following and function calling fall off way faster with multi-turn voice than they do than than people expect them to. That's the thing that we work we work to solve for most often in production today in 2025. There also when you a lot of people really want to use the new speechtospech models and so there's a whole class of failure modes in the new speech to speech models that are super surprising in the way that all the newest models are. like I think Brooke and I were actually talking about this over the weekend like the GPT4 real-time model will often just start speaking in Spanish and you're like there's nothing in the prompt or in any of the any audio input that a human can perceive that would have you sort of switch into the Spanish angle in the latent space, but like you do it a lot, so there's something there.

so there's a bunch of stuff like that in the very newest models. Yeah, it's actually my my grandma likes speaking the chat GBT in her native language. and every now and then it'll say like, "Oh, I'm sorry. Like, I can't say that in English." And then it'll go back and speak in the native language, which is so bizarre. And she just like doesn't know how to deal with it, but she's gotten used to it. Yeah. No, I'm sure she's building up a really good model of how the the model works, like a good mental model of how the language model works from just using it a lot. It's super jagged. It's the jagged frontier thing. It's really interesting.

Yeah. All right. Other folks have come in with questions. does Pipcat or any vendor have an eval database facility that works with logging? What we try to do with Pipcat is build great bridges to the tools that specialize in all the parts of the voice AI space including eval. So like Brooke is going to talk about exactly how to build voice agent to to Koval like bridges from maybe multiple platforms. Certainly there's a great pipecat integration with coal. Is that the question? I'm not sure. Sorry with pipecat and co pipcat cobalt and langfuse. And so you I think a lot of the tracing a lot of tracing and the like standard evals actually don't change with voice where if you're just trying to test a prompt or you're trying to do the tracing that remains the same.

So I think building on top of that with pipecat and cobalt can help you there. Makes sense. another question on deployment. Do you run everything on a single Kubernetes cluster and route internally or do you call various APIs? yeah maybe broadly like what are the common architectures that you see for complex kind of parallel I think is what you mentioned but multi-step pipelines. It's a great question. So generally people in production are running a you know a voice agent process container on a CPU machine and calling out two hosted APIs. it's that's partly because the best models actually aren't available for you to run yourself. And it's partly because we're also new in this space that you know building the thing that works and then over time making it better better along various angles that are complicated to to to figure out and cheaper is like a second step and most people even at scale aren't on that like second step yet. so most people do something use deepgram for transcription, GPD40 or Gemini 20 flash for the LLM and Cartisia or 11 Labs for the voice model.

That's the combination we see most often today. Every one of those blocks though is more and more competitive and more and more exciting a space. So there's more and more good transcription models, more and more good voice models. One of the reasons we built Pipcat was to make it super easy to like swap out and test different models and even use different models for different parts of a conversation or for different routed conversations in a complex environment.

if you're hosting this stuff yourself, you do need to like do a bunch of Kubernetes work to figure out like how the shape of this particular longunning UDP dependent or networking dependent workflow works. It's a lot of work. we did launch at daily a Pipcat cloud hosting service a couple months ago, which you can use if you want to. Nothing about Pipcat depends on any vendor though, so you definitely don't have to use it. and being multi-reion is also complicated and important for a lot of big at scale use cases. There's a tension. You want to you want your voice agents to run close to your users so that the network latency is low going back and forth to the voice agent. Most of the big providers though don't have inference in very many locations and if you have to choose you'd rather be close to your inference provider rather than close to your users because you're making multiple inference provider round trips if you're using hosted APIs. So there's a bunch of things to balance and we spent a lot of time talking about this in like the voice AI course for example because there's no perfect answer right now. Thank you everybody. Thank you so much. This was so much fun. This talk was amazing. Thank you. Byebye.