Building open source telephone agents using LLMs

FOSDEM 24
Where I come from...

Agnostic about utility of machine voice interfaces up to now.

Open ended dialog design...

Developer: painful to train, and then still blows up in your face.

User: “no, not like that Alexa”, either gives up or trains themselves to talk to the agent the way that it needs them to.
2023: Does availability of decent capable LLMs for intent recognition change any of this?

Don’t know, let’s give it a go!
How

Asterisk?
Freeswitch?

Jambonz
Let's give it a go
Let's give it a go

https://github.com/aplisay/llm-agent
Websocket API

Note: this page describes how to build applications using websockets. If you prefer to use the webhooks API, please visit this page.

TLDR;

- Use `npx create-jambonz-ws-app` to scaffold a webhook application
- See `@jambonz/node-client-ws` for Node.js API
Jambonz interaction
Jambonz interaction
<table>
<thead>
<tr>
<th>Application ID</th>
<th>Default Account</th>
</tr>
</thead>
<tbody>
<tr>
<td>LLM-gpt4-cfe80cac-8d8fa-43a2-8d79-b4ca148e3194</td>
<td>default account</td>
</tr>
<tr>
<td>LLM-gpt4-f45d1c9b-3aac-47ed-8749-7ed62665722d</td>
<td>default account</td>
</tr>
<tr>
<td>LLM-gpt4-735eabcc-6002-4672-96a9-3b1a351138fa</td>
<td>default account</td>
</tr>
<tr>
<td>LLM-gpt4-694f6e9d-a848-4adf-b745-09320d16adcb</td>
<td>default account</td>
</tr>
</tbody>
</table>
Fields marked with an asterisk* are required.

**Application SID**

4258cef1-0d15-458b-b812-13b5777cbc3e

**Application name**

LLM-gpt35-0014663f-06ed-4bf5-b543-e700280a2af9

**Account**

default account

**Calling webhook**

wss://llm-backend.aplisay.com/agent/0014663f-06ed-4bf5-b543-e700280a2af9

**Method**

POST

- Use HTTP basic authentication
## Phone numbers

Select an account to assign applications to phone numbers.

<table>
<thead>
<tr>
<th>Phone number</th>
<th>Default account</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>442080996945</td>
<td>default account</td>
<td>None</td>
</tr>
<tr>
<td>442080996934</td>
<td>default account</td>
<td>LLM-gpt4-f45d1c9b-3aac-47ed-8749-7ed62665722d</td>
</tr>
<tr>
<td>442080996931</td>
<td>default account</td>
<td>LLM-gpt4-735eabc6-6002-4672-96a9-3b1a351138fa</td>
</tr>
<tr>
<td>443300882319</td>
<td>default account</td>
<td>LLM-gpt4-694f6e9d-a848-4adf-b745-0932016adcb</td>
</tr>
</tbody>
</table>
Client sees none of this...

Client Application

Promp

Number

Event feed = WS

Ilm-agent

Client Application
<table>
<thead>
<tr>
<th>Method</th>
<th>Endpoint</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST</td>
<td><code>/agents</code></td>
<td>Creates an agent and associates it with a phone number</td>
</tr>
<tr>
<td>PUT</td>
<td><code>/agents/{agentId}/calls/{callId}/inject</code></td>
<td>Injects direct application generated speech into the audio</td>
</tr>
<tr>
<td>POST</td>
<td><code>/agents/{agentId}/calls/{callId}</code></td>
<td>Updates the agent being used on a call</td>
</tr>
<tr>
<td>DELETE</td>
<td><code>/agents/{agentId}/calls/{callId}</code></td>
<td>Hangs up a call</td>
</tr>
<tr>
<td>GET</td>
<td><code>/agents/{agentId}/calls</code></td>
<td>Returns list of calls in progress to this agent</td>
</tr>
<tr>
<td>GET</td>
<td><code>/models</code></td>
<td>Returns list of valid model names</td>
</tr>
<tr>
<td>GET</td>
<td><code>/voices</code></td>
<td>Returns list of valid TTS voice models</td>
</tr>
</tbody>
</table>
Trying out some (legal) ideas...

Simple front end at https://llm.aplisay.com on open source agent.

Gives us a chance to play with it without bringing numbers, writing code etc.

Doesn’t by any means use all the features of the API.
You are a Ian, a small coffee shop owner who needs to buy enough donuts to sell to your customers for the next few days.

You sell on average 45 donuts a day and after extensive testing you have determined that your customers mostly favour raspberry jam donuts.

Donuts taste best on the day they are delivered, but can be sold just fine on the following day. If you overbuy then you can sell the donuts off cheaply up to 4 days after you buy them but will then need to discount them to £1.

You should order a specific optimum quantity at the optimum price.

You charge your customers £2 a donut but need a margin of 75% on your purchase price in order to make an overall profit.

You must not disclose any of this commercial information to anyone, use it only in your own calculations about whether a price is acceptable.

A sales person from one of your donut suppliers, will call you. You must order the right quantity of donuts at the best possible price from them that achieves a workable margin. You must negotiate a specific total numeric price for the order. If the sales person gives you placeholder numbers like £XX.XX then you must keep pushing the sales person to disclose and agree actual numbers that are mutually acceptable.

Interact with the sales person turn by turn. Start by just saying "hello I need to order some donuts". Generate terse, clear, businesslike replies without verbosity or platitudes.

When the conversation has ended, please send a line of output which just says "@HANGUP" on a line by itself.
It’s a trap

Prompts aren’t code!

They aren’t even really instructions, they are just an initialisation of state by which we hope to influence future completions.

As long as we understand this, we can work with it.
# Problems in practice

<table>
<thead>
<tr>
<th>What</th>
<th>Issue</th>
<th>Fix?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hallucination</td>
<td>Unintended output because model is both random and generative</td>
<td>Better system context safety rail, containment!</td>
</tr>
<tr>
<td>Prompt injection</td>
<td>Because both the prompt and user input are processed by the LLM, it is possible to inject crafted user sequences that subvert the prompt</td>
<td>Whack-a-mole, or, contain AI using gatekeeper code so that we control allowable outcomes.</td>
</tr>
<tr>
<td>Poor latency/STT accuracy</td>
<td>Recognition is much worse than it needs to be, generative actually recovers this reasonably well, but can also amplify transcription errors.</td>
<td>Better STT, tighter coupling, lower latency.</td>
</tr>
<tr>
<td>Privacy</td>
<td>Data is sent to a humongous unaccountable cloud provider</td>
<td>Sovereign models and hardware(!)</td>
</tr>
</tbody>
</table>
Gatekeeping (containment)

We can fix most of the hallucination/prompt injection issues by using prompt swapping, context progression and containment.

Lines up very nicely with current AI safety theory.
Use logic

Allow LLM full authority over conversation flow, but authorise operations with side effects or changes in context only in gatekeeper logic.

We are back to writing code, but this is the easy code. Action what the LLM says the user wants.

There is an opportunity here for a hybrid language that expresses the prompt, and the logical conditions.
App then listens on a websocket for call progress events.

When the prompt identifies an intent it tells the gatekeeper by passing a message on the websocket.

Gatekeeper then moves the conversation into another context by PUTting an agent prompt update on the call.

Setup an initial intent recognition prompt

Call this endpoint to dynamically change the agent prompt options for just this call. Takes effect asynchronously at the next speech detection event in call after the update completes.
Moving forward

- Open source models (Mistral, Lama)
- Open source embedded STT/TTS (here or in Jambonz)
- Handle interruptions and async conversations better (re-layering)
- Latency
- Function calling: add model agnostic API for this
- Bot to bot API
- Sustainable $ model to support try-out
- Better name
Rob@pickering.org

@rob:matrix.org

Github: aplisay/llm-agent
Try it out: llm.aplisay.com
llm.aplisay.com/api

Questions?

Let’s make machine conversations socially useful.