Agent Design  
  
OK, so I did not explain myself properly. My idea of an AI agent is one that I can message with using SMS (e.g. Twilio). It would interact with me over time, and based on my responses to the questions and interaction that the AI got from the SMS messages, it would build up an understanding of what I would be interested in. I need some help fleshing this idea out.

**ChatGPT said:**

Updated saved memory

Got it — that makes your vision clearer. You're looking to build a **conversational SMS-based AI agent** that:

* Interacts with you **via SMS** (e.g., using Twilio),
* **Learns about your preferences over time** based on the back-and-forth conversation,
* **Surfaces relevant social dance events** (from your social\_dance\_app database),
* And can **ask you questions** to refine what you're looking for or nudge you with helpful suggestions.

Let’s flesh this out at a high level.

**🧠 Conceptual Architecture**

java

CopyEdit

YOU (via SMS)

⬍

TWILIO WEBHOOK (FastAPI endpoint)

⬍

AGENT CONTROLLER (LLM + Memory + Personalization)

⬍ ⬌

PostgreSQL (social\_dance\_app DB) Vector store / memory (your preferences)

**🔧 Components**

**1. SMS Frontend (Twilio)**

* Inbound: Messages from you get sent to a webhook on your backend (e.g., FastAPI).
* Outbound: Your backend sends replies via Twilio’s API.

**2. FastAPI Webhook Server**

* Endpoint: /sms/inbound
* Receives Twilio POSTs → parses message → routes to AgentController
* Sends back TwiML or API response to reply with a message

**3. AgentController (Core Logic)**

This is your brain logic. Each message triggers:

* **Message understanding** (e.g., "I'm looking for something casual this weekend")
* **Preference updating**: Learn from what you like/dislike
* **Response generation**: Could ask a clarifying question, recommend events, or confirm interest
* **Message logging**: Save interaction to a history DB or memory

It should:

* Keep a **conversational history** (even across days)
* Use an **LLM (e.g., GPT-4o)** to parse intents and generate friendly replies
* Store/update your **interest profile**

**4. Interest Profile & Memory**

Examples of what the agent might learn over time:

* Preferred dances (West Coast Swing, Bachata, etc.)
* Preferred days/times (Fridays after 7pm)
* Favorite venues
* Radius/area of interest
* Event types (socials vs. workshops)
* Prior thumbs-up/thumbs-down events

Store this as:

* JSON blob or table in PostgreSQL
* Optionally vectorize preferences (e.g., with sentence-transformers or OpenAI embeddings)

**5. LLM + Tools**

The LLM can:

* Parse your message
* Summarize preferences
* Ask clarifying questions (“Do you want something Friday or Saturday?”)
* Generate replies
* Call tools:
  + search\_events(query)
  + save\_preference(key, value)
  + get\_recommendations()

**6. Optional Admin Dashboard**

For debugging or tuning, you could have a Streamlit app that:

* Shows current "understood preferences"
* Lets you simulate messages
* Shows recent message history

**🧪 Example Workflow**

**You send (via SMS)**:

“Any good Westie dances this Friday night?”

**System does**:

* Parses that you want:
  + dance\_style = "West Coast Swing"
  + date = next Friday
  + time = evening
* Queries DB → finds matches
* Replies via Twilio:

“There’s Westie Night at Centennial Hall, 7:30pm–11:00pm. Want me to remember that Fridays are good for you?”

**🗃️ Data Storage**

| **Table / Store** | **Purpose** |
| --- | --- |
| message\_log | Logs all inbound/outbound SMS |
| user\_profile | Stores long-term preferences |
| conversation\_state | Tracks ongoing interaction state |
| event\_recommendations | Cached or past recommendations |

**🧭 Next Steps**

To help you design this concretely, I’d recommend we decide:

1. **Where to run the FastAPI service** (Render is a good choice).
2. **Whether you want persistent memory** (PostgreSQL + maybe a vector DB).
3. **Whether to start with rule-based intent detection or go full LLM+tools.**

Would you like a starter repo or diagram next? Or should we dive into the SMS webhook + controller design?