

OpenAI

Customer Service Transformation Accelerator

July 2025

How to use this Accelerator

This accelerator offers a robust starting point for anyone looking to quickly ramp up to an intermediate level understanding of the tools, methodologies, and outcomes involved. You'll gain practical insights from proven reference architectures. To help you take the next step, we've curated a selection of additional resources tailored to deepen your expertise.

- ✔ Showcasing the options you have to build this solution
- ✔ Giving you a high-level approach to know where to start
- ✔ Showing you how to approach next steps

Overview

Customer Service Transformation is a multi-stage journey that organizations take to modernize support operations using LLMs, evolving from basic self-service tools to highly autonomous AI agents. This includes augmenting capabilities, increasing efficiency, and automating significant work.

Example use cases

- Customer Service Chat
- Customer Service Calls

What we'll cover

- Value Proposition
- Building:
 - Process flow
 - High-level architecture
 - Production examples
 - Features & limitations
 - Key challenges & pitfalls
 - Solution Deep Dive
- Success & Key Takeaways

Value Proposition

Value Proposition

Improved Resolution Speed

LLM assistants can cut average response times dramatically—often resolving queries in under 2 minutes compared to traditional multi-minute workflows.

Higher Scalability

LLMs handle high volumes of routine tasks, reducing the need for large support teams and enabling cost-effective growth without sacrificing quality.

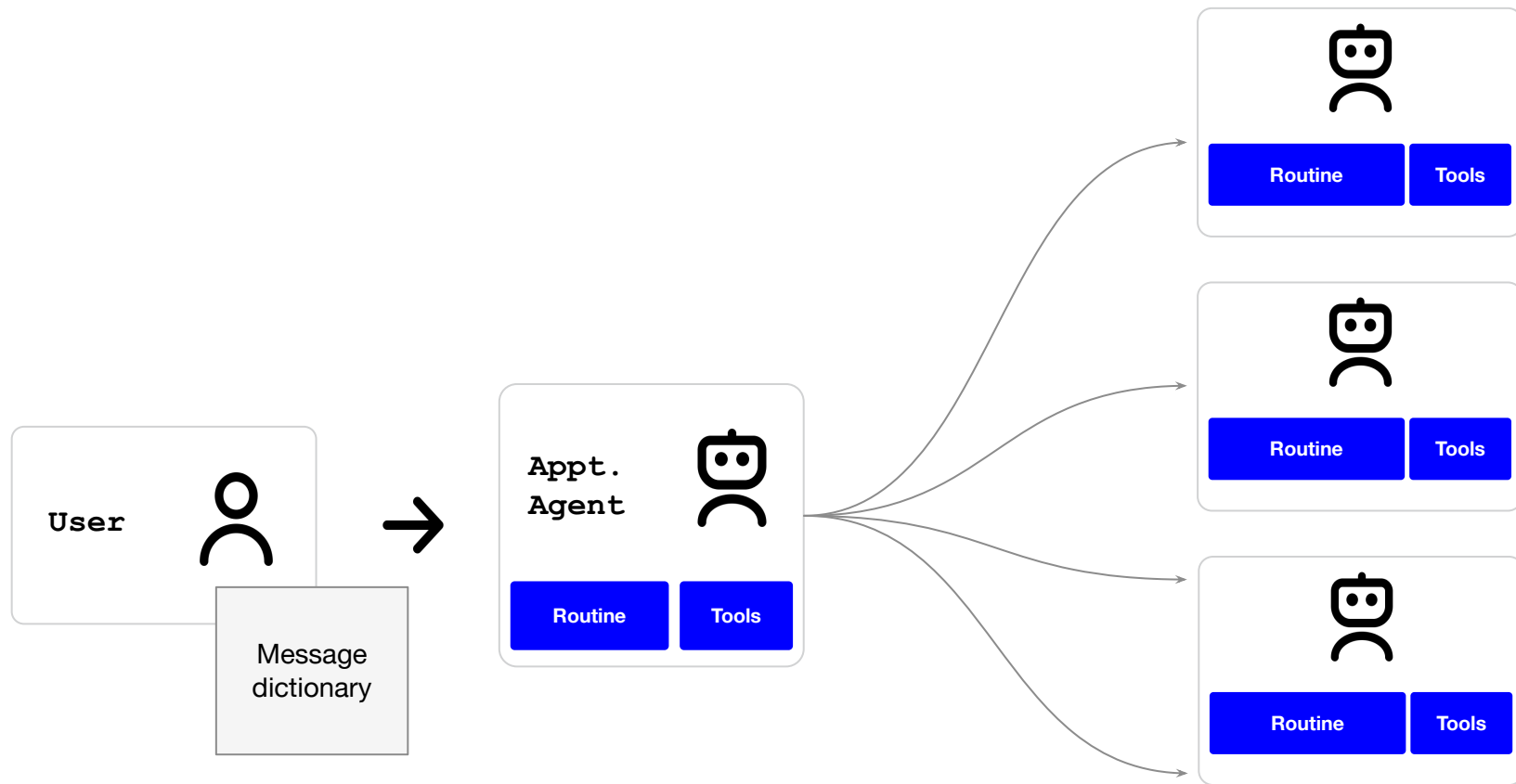
Increased Customer Satisfaction

By delivering consistent, helpful, and personalized answers, LLMs raise satisfaction levels and maintain quality even during high demand.

Building

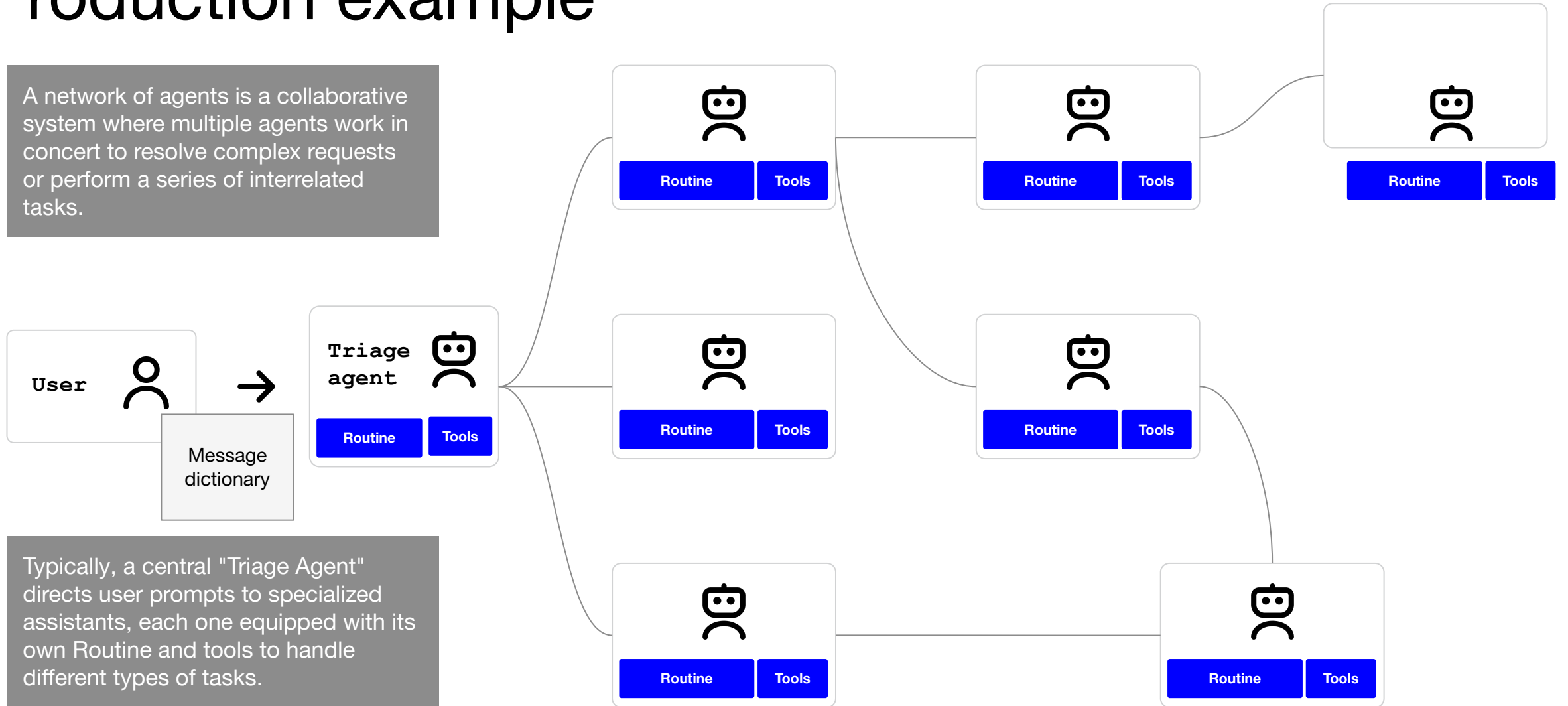
Process flow

High-level architecture



Production example

A network of agents is a collaborative system where multiple agents work in concert to resolve complex requests or perform a series of interrelated tasks.



Typically, a central "Triage Agent" directs user prompts to specialized assistants, each one equipped with its own Routine and tools to handle different types of tasks.

Features and limitations

Good for

- **Incremental Automation:** Step by step working towards more automation, scaling and saving costs incrementally.
- **Scalability:** Once a task is automated, it can be scaled at very low cost;

Not good for

- **Cold Start:**
 - Lots of data is needed ahead of time, and this will not address missing capabilities.
- **System Consolidation:**
 - The need to consolidate information and systems is a prerequisite, and not something solved by this accelerator.

Key Challenges & Pitfalls

CHALLENGES

- Creating good evals to properly assess the accuracy and risk.
- Ensuring performance is consistent enough to justify the automation
- Shifting from human to AI support demands careful change management to preserve customer trust and employee alignment.

PITFALLS & MITIGATIONS

- **Too much too soon**
 - Ensure that simpler tasks are automated first, and build upon those. Fight the urge to automate too much upfront.
- **Unprepared Organizations**
 - Before beginning a project, check that data/systems are ready and accessible, or ensure that shifts are made.

Solution Deep Dive

What is an agent?

An agent is a **model** with a **routine** that guides its behaviour, and has access to **tools** to extend its capabilities

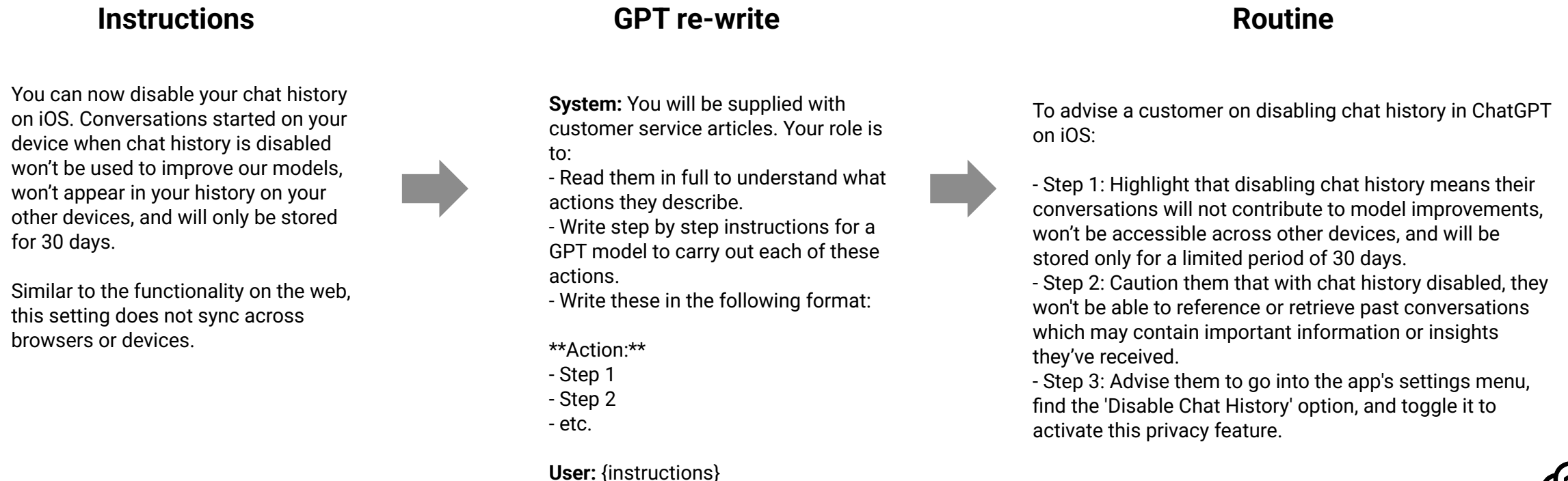
- A routine is a prompt that contains clear instructions for the agent to follow
- A tool (or function) allows the model to access external systems or functionalities. (e.g. allows model to query an internal SQL database)

Agent equals

+ Model
+ Routine
+ Tools

How to define routines

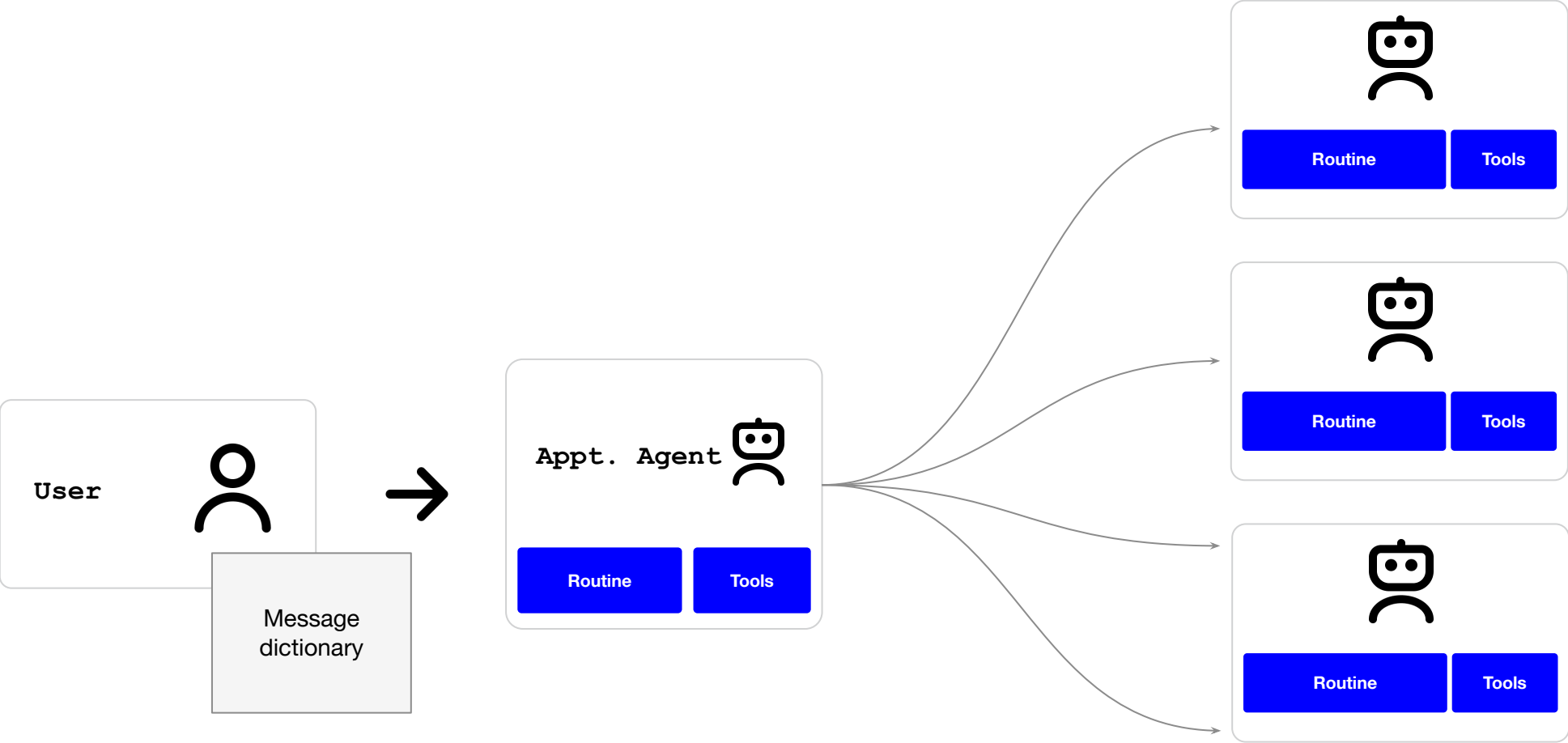
- Routine generation should be driven by the quality of the SOPs...
 - If SOP's are polished, complete, and logical, you can use **SOP's as routines directly**
 - If SOP's are multi-modal (images, graphs), contain irrelevant information, and are unclear in structure, **build a routine generation pipeline, like the one below.**



How to build a customer service agent

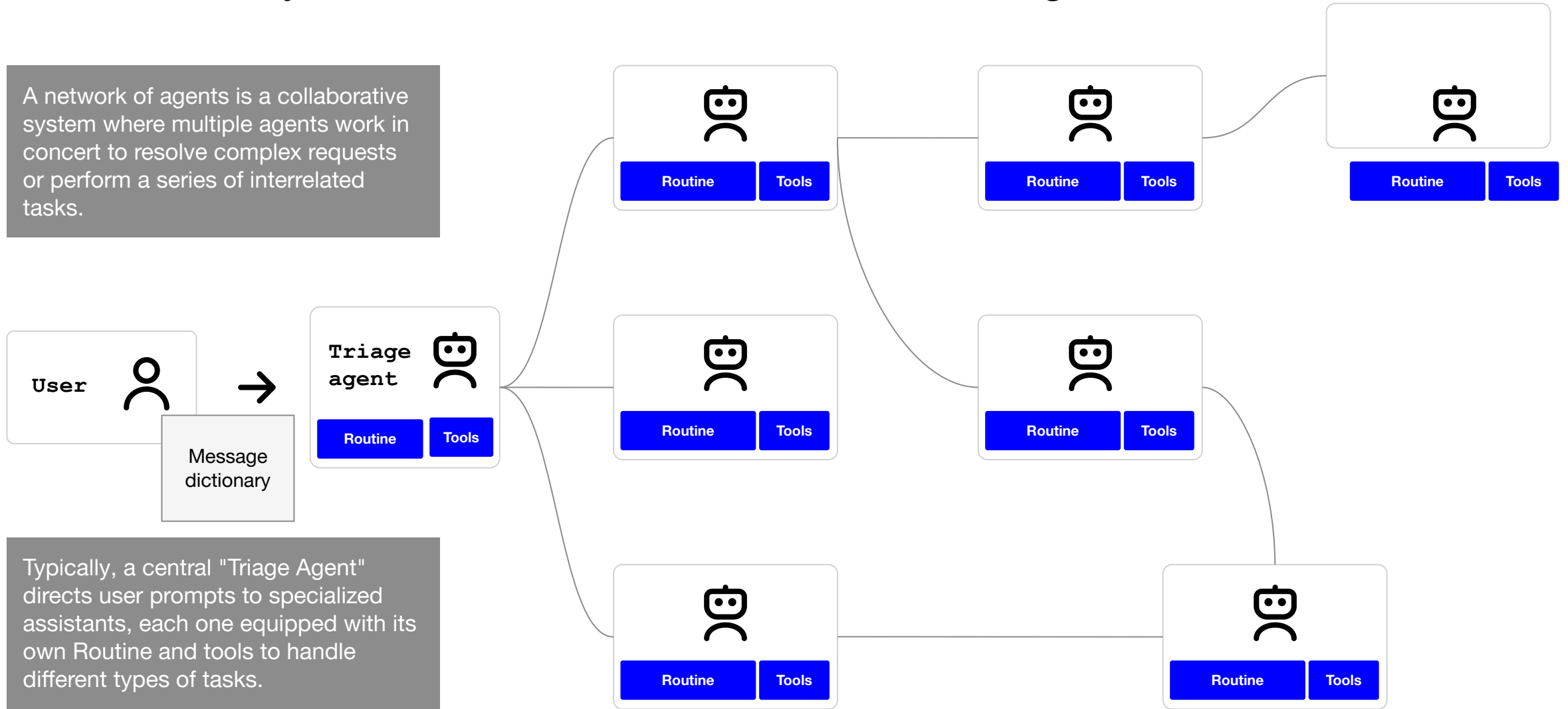


Begin with a single triage agent with multiple sub agents



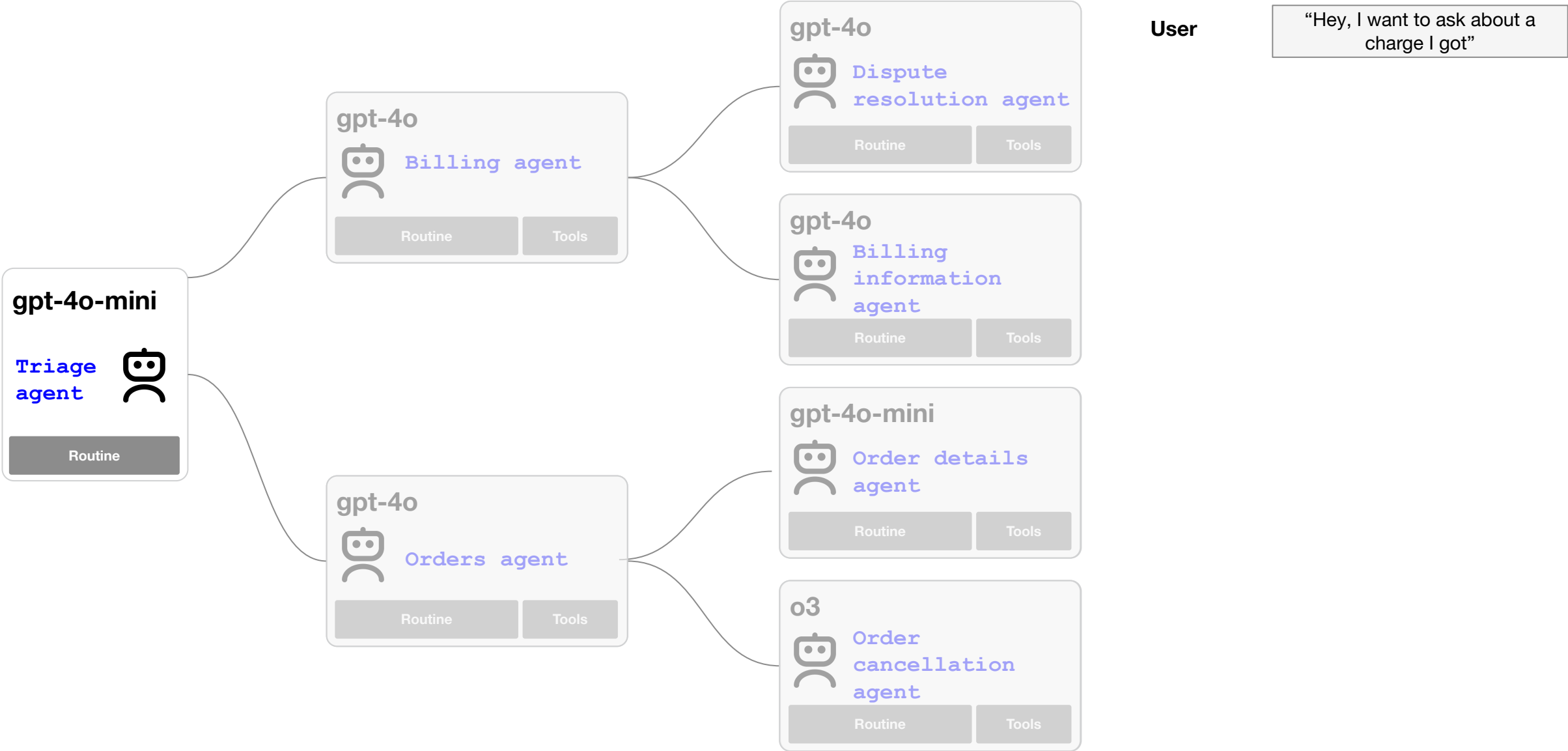
However, often you will need a hierarchical network of agents

A network of agents is a collaborative system where multiple agents work in concert to resolve complex requests or perform a series of interrelated tasks.

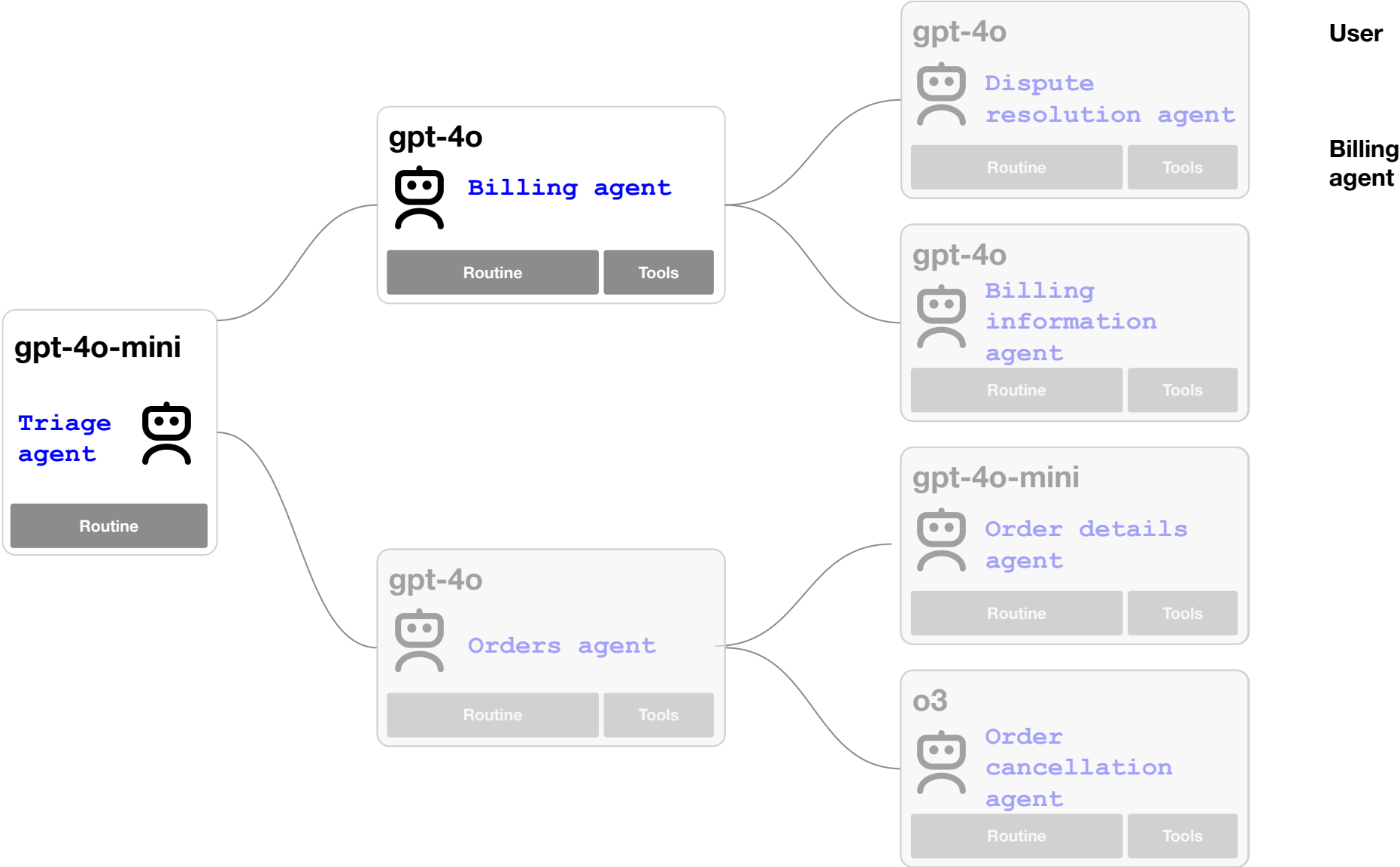


Typically, a central "Triage Agent" directs user prompts to specialized assistants, each one equipped with its own Routine and tools to handle different types of tasks.

Example: Customer service bot



Example of hierarchical agents: Customer service bot



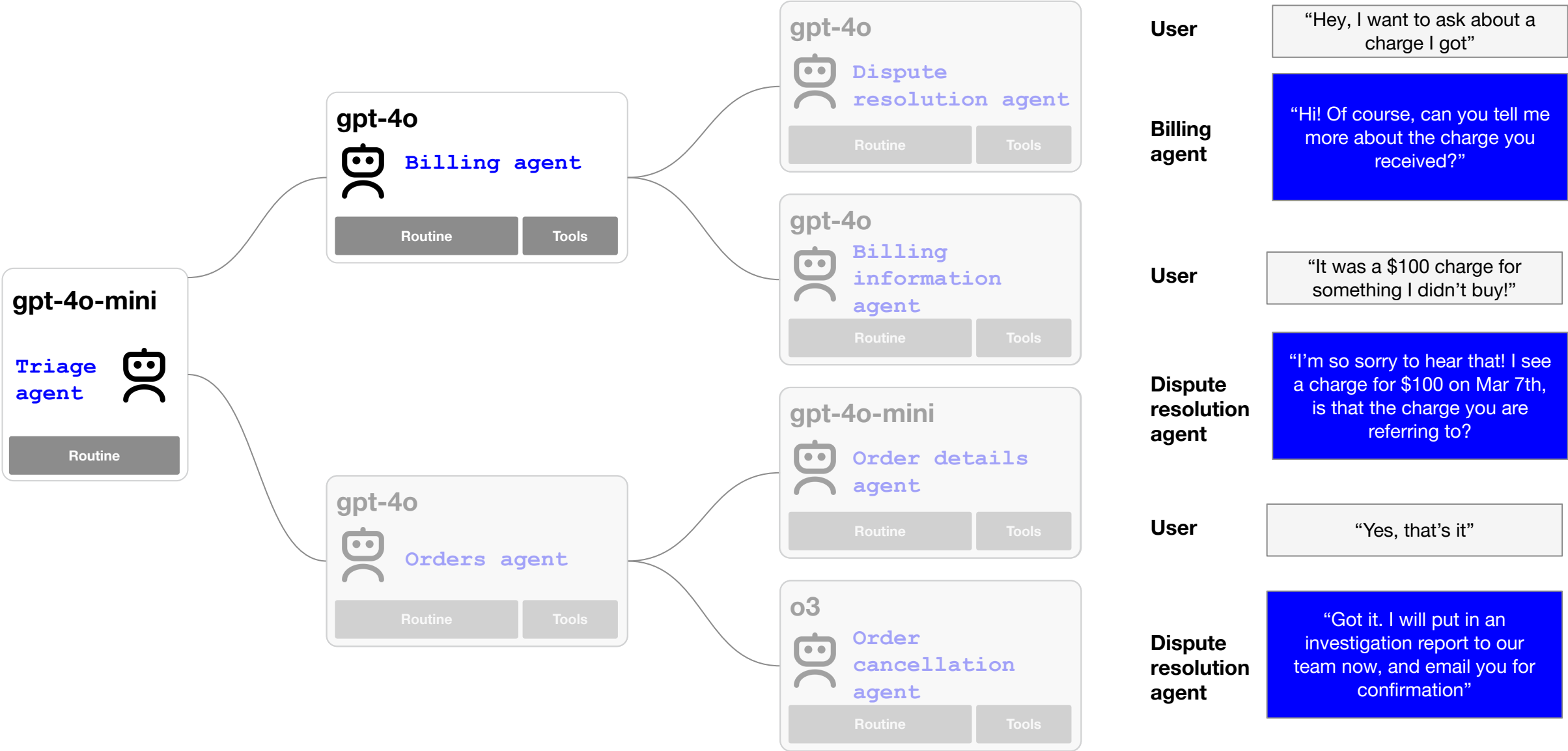
User

“Hey, I want to ask about a charge I got”

Billing agent

“Hi! Of course, can you tell me more about the charge you received?”

Example: Customer service bot



To build agents, use the Agents SDK

Robust and lightweight framework (evolution of Swarm) for managing multi-agent orchestration

- ⚡ **Lightweight** Simple and powerful features without a steep learning curve
- 🔧 **Flexible** Works great out of the box but also fully customizable
- 🔄 **Workflows** Build complex workflows and systems by defining agents, tools, handoffs and guardrails
- 🔍 **Tracing** Features built-in tracing (available in our developer platform)
- 🔄 **Optimization** In future, tracing will integrate with our evaluations and fine-tuning/distillation platforms

```
Python
from agents import Agent, Runner, WebSearchTool, function_tool, guardrail

@function_tool
def submit_refund_request(item_id: str, reason: str):
    # Your refund logic goes here
    return "success"

support_agent = Agent(
    name="Support & Returns",
    instructions="You are a support agent who can submit refunds [...]",
    tools=[submit_refund_request],
)





shopping_agent = Agent(
    name="Shopping Assistant",
    instructions="You are a shopping assistant who can search the web [...]",
    tools=[WebSearchTool()],
)

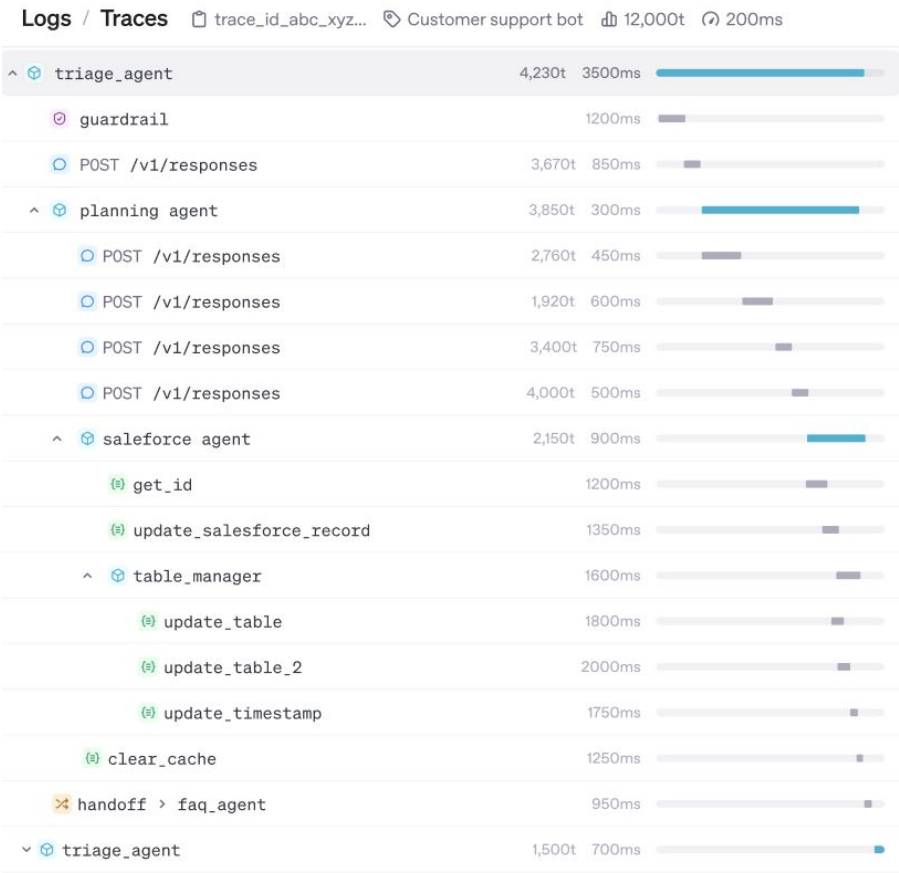
triage_agent = Agent(
    name="Triage Agent",
    instructions="Route the user to the correct agent.",
    handoffs=[shopping_agent, support_agent],
)

output = Runner.run_sync(
    starting_agent=triage_agent,
    input="What shoes might work best with my outfit so far?",
)
```

And trace your agent workflows natively

Provides end-to-end transparency into agentic tasks and orchestration for debugging and optimization

-  Observability A trace is a detailed record of a complete orchestration run, capturing all interactions, including model calls, tool invocations, handoffs and guardrails
-  Integrated Traces works out of the box with our Agents SDK
-  Dashboard Developers can view traces within the “Logs” section of the Dashboard to visually debug agent behaviors
-  Optimization In future, OpenAI will offer trace graders and trace evals, plus model customization based on trace data



Production best practices

Keep it simple

A single system prompt with good functions is often enough.

Express hard logic with code

Well-defined workflows and if/else conditionals.

Make tasks constrained

Spell out tasks step-by-step and highlight edge cases explicitly in your prompts.

Focus on the frequent user journeys

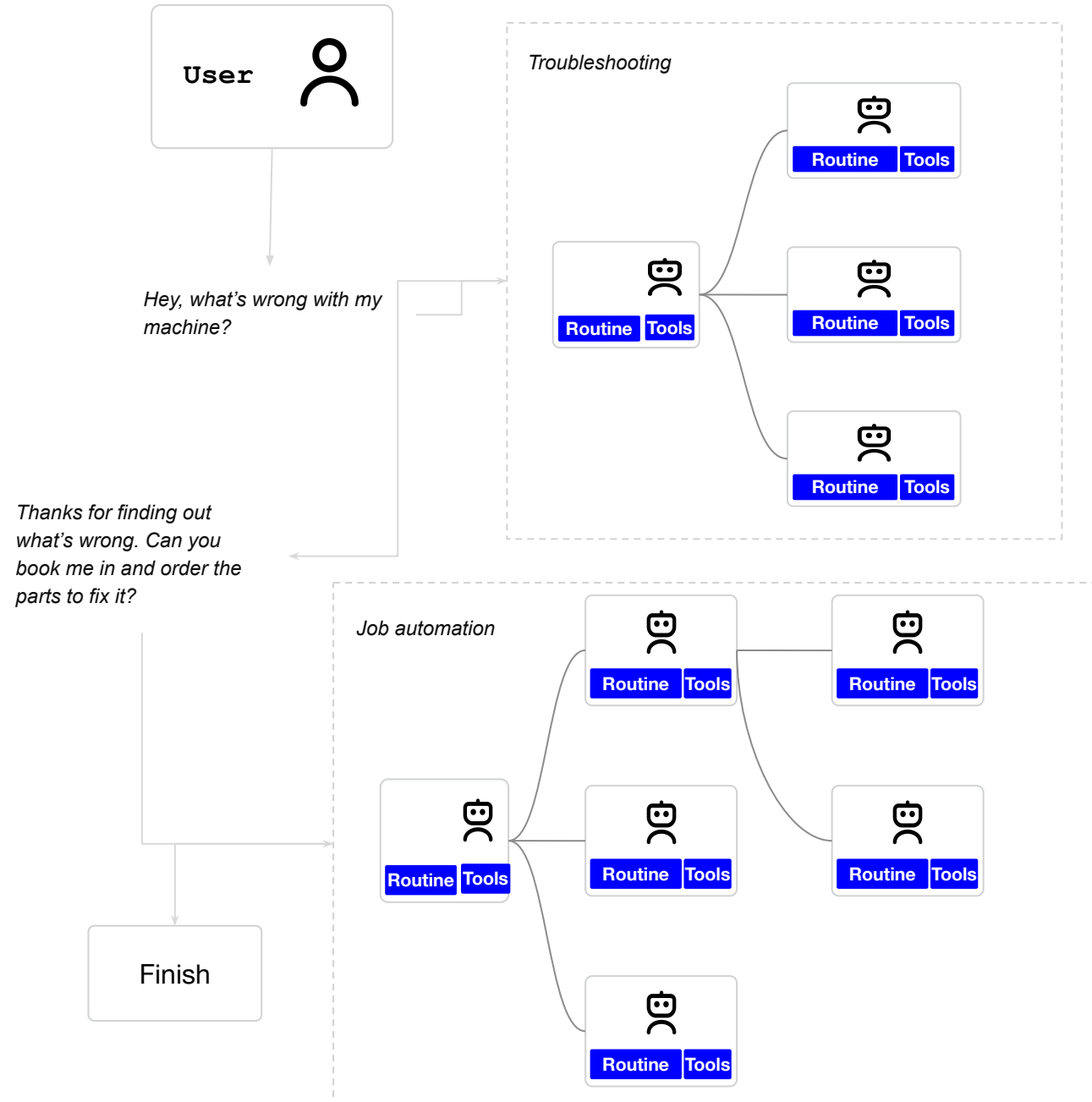
There's often fewer than you think.

Network of networks

Problem: You have an effective network that solves one family of problems, but now you want to extend. Rather than start fresh, you want to rely on other networks in the business where appropriate.

Solution:

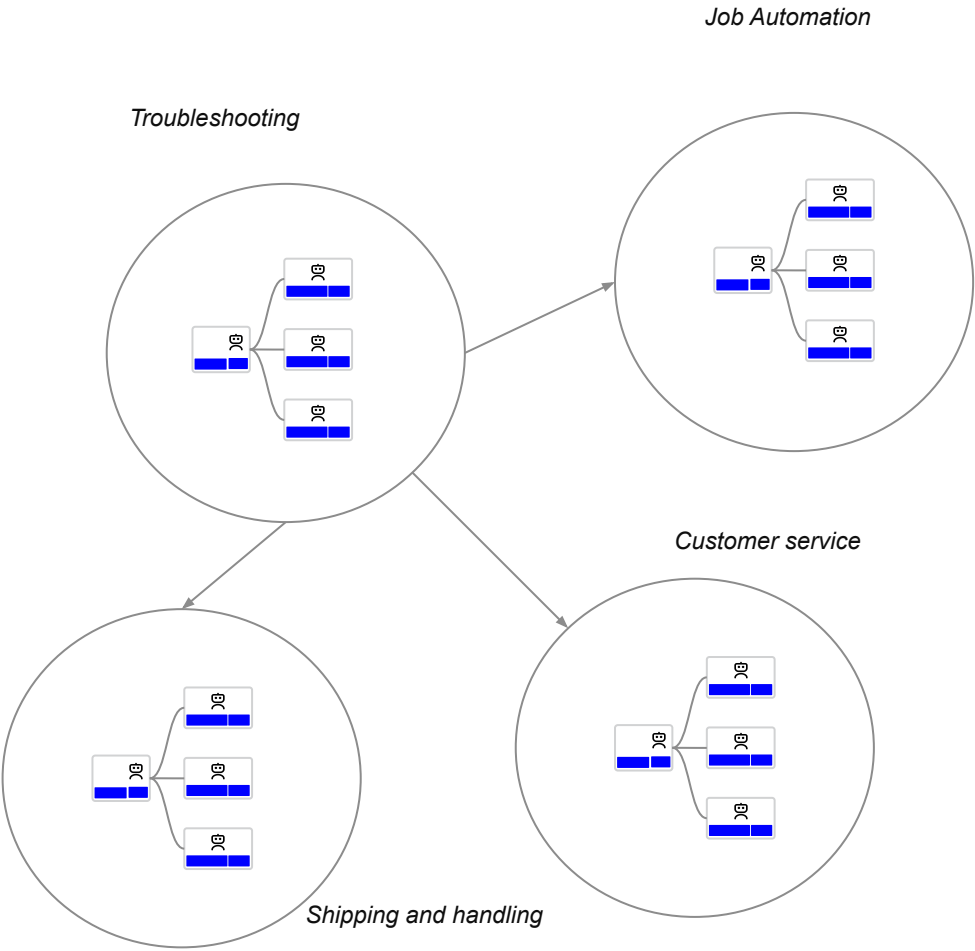
- Creating a network of networks



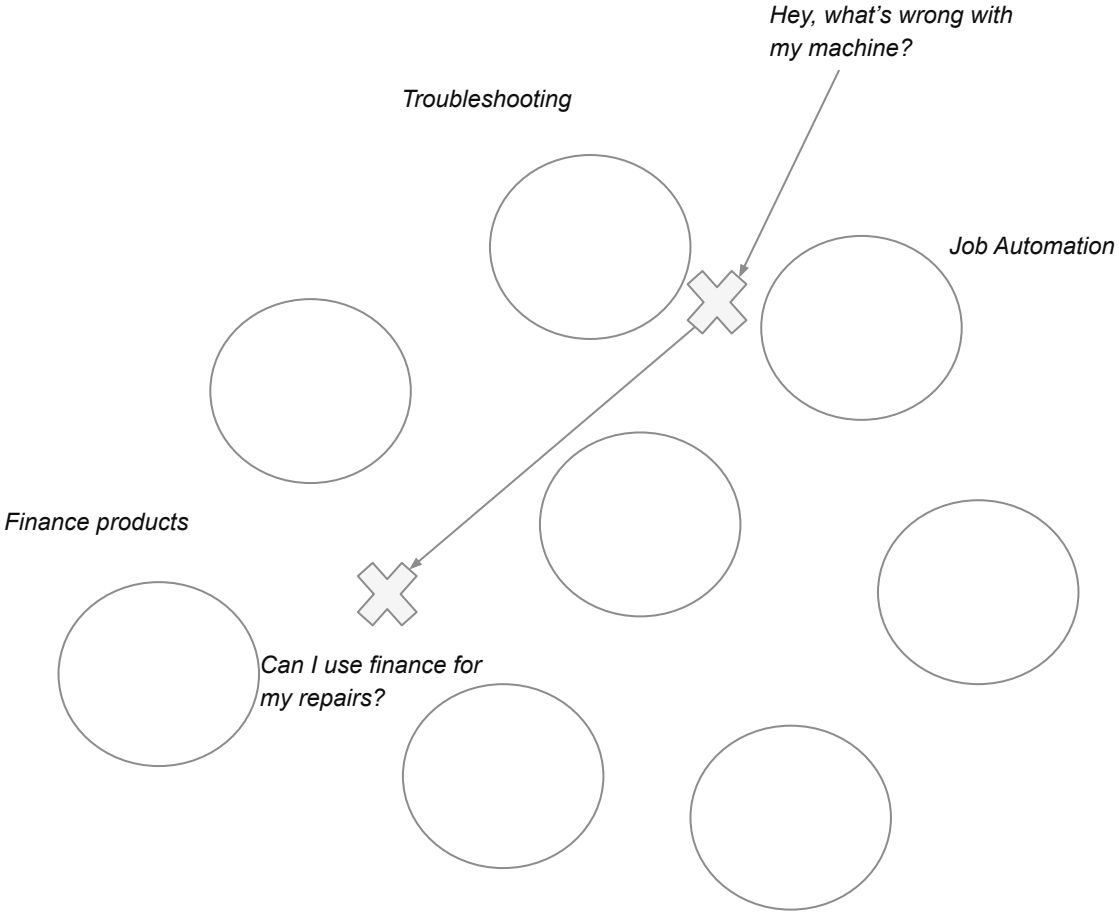
Network of networks

Hand-offs

Graph



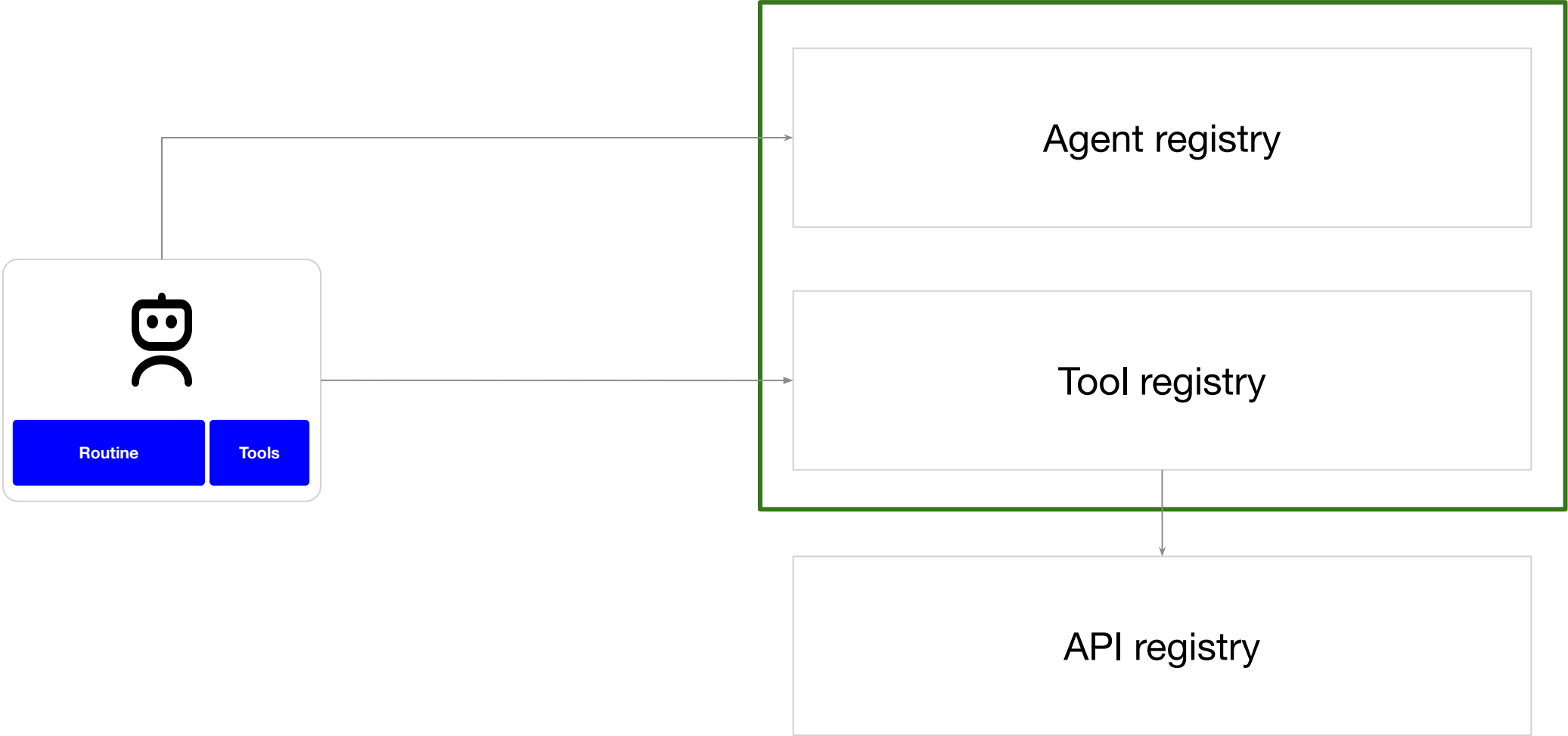
Embedding



Network of networks

Management

Evals



Best Practices

Iterate quickly

Deploy a small, scoped prototype, and scale out from there. Production traffic is always different from test data, so prioritize getting real world feedback as soon as possible.

Work with customer care experts early

Create a feedback loop between customer care and engineering, where care agents can provide qualitative feedback on the quality of chatbot or voice AI responses

Clearly establish ‘what good looks like’

Create robust PRD’s and sets of requirements, and make sure to receive buy-in across product, customer care, engineering, and the executive team.

Test RAG Components in Isolation

Evaluate NER, keyword search, and semantic search independently. This allows you to iterate, benchmark performance, and troubleshoot issues without cross-contamination.

Create a Feedback Loop

Track how users interact—clicks, likes, purchases—to refine keyword weighting and update embeddings. User signals help the system learn and continuously improve.

Success & Customer Stories

Success metrics

Process Step	Solution	Evaluation	Metric
Documentation	Accuracy	Check automated tasks accuracy (summaries, system updates taken, etc)	Accuracy of notes, correctness of system updates.
	Time Savings	Time savings on misc tasks	Estimated hours of time savings on miscellaneous tasks
Calls	Calls Automated	Measure how many call-minutes are not done by humans.	Minutes of automated calls.
	Accuracy	Test model on known set of questions and answers.	% Correct (accuracy)
	Guardrails	Test model on inappropriate prompts and check the response. Also test with regular prompts to count refusals	% incorrectly refused (False positives) % incorrectly not refused (False negatives)
	Instruction Following	Adhering to tone, expressiveness, system instructions	% Correct responses and model promptness for given query

Key Takeaways

- **Crawl, Walk Run:** Customer Service transformation works best when approached incrementally. “Crawl” involves low-risk deployments like Q&A chatbots or summarization tools. “Walk” introduces human-in-the-loop agents with scripted routines and limited tool use. “Run” means handing off end-to-end tickets to fully autonomous agents with system integrations and action capabilities
 - *Note: fully autonomous agents are required to disclose to end users that they're interacting with an AI*
- **Need for Preparation:** Success hinges on upfront work—clean documentation, well-defined service routines, system integrations, and clear escalation paths. Without this foundational layer, AI can’t act with confidence or precision. Preparation also includes aligning stakeholders, mapping flows, and setting evaluation criteria.
- **Evals Evals Evals:** Evaluations are critical at every phase. They help measure the accuracy, safety, and ROI of AI actions—especially when tools and automation are introduced. Before scaling, teams use evals to validate model performance, spot regressions, and ensure systems behave reliably in production scenarios