Oct 09, 2025

# OpenAI Agent Builder + Tinybird MCP: Building a data-driven agent workflow

OpenAI just announced AgentKit, a suite of tools for building and deploying AI agents. Here's how to make data-driven, analytical workflows with the Tinybird MCP Server on OpenAI's Agent Builder.

AI x Data

Cameron Archer
Tech Writer

OpenAI just announced AgentKit, which they describe as "a complete set of tools for developers and enterprises to build, deploy, and optimize agents."

Some proclaim that this vertical integration from OpenAI is going to "kill" many AI startups. Maybe, maybe not. I'm not here to debate it.

What I want to show you is how you can use OpenAI's Agent Builder along with the Tinybird MCP Server to build agentic workflows that don't hallucinate, because they're providing responses based on real, fresh data.

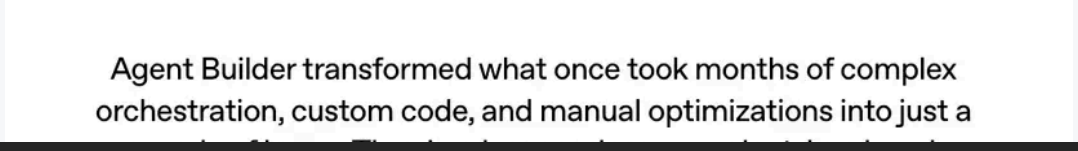
## What is OpenAI's AgentKit?

In essence, it's a UI-based workflow builder wrapped around OpenAI models and tool calls, allowing cross-functional teams to build agent chains with conditional logic, "connectors" (read: MCP servers), and even sticky notes!

You can then embed your agent workflow into your application using ChatKit (low-code) or the Agents SDK for Python or Typescript.

Finally, you can run evals on your agents and enable trace grading to score responses and optimize.

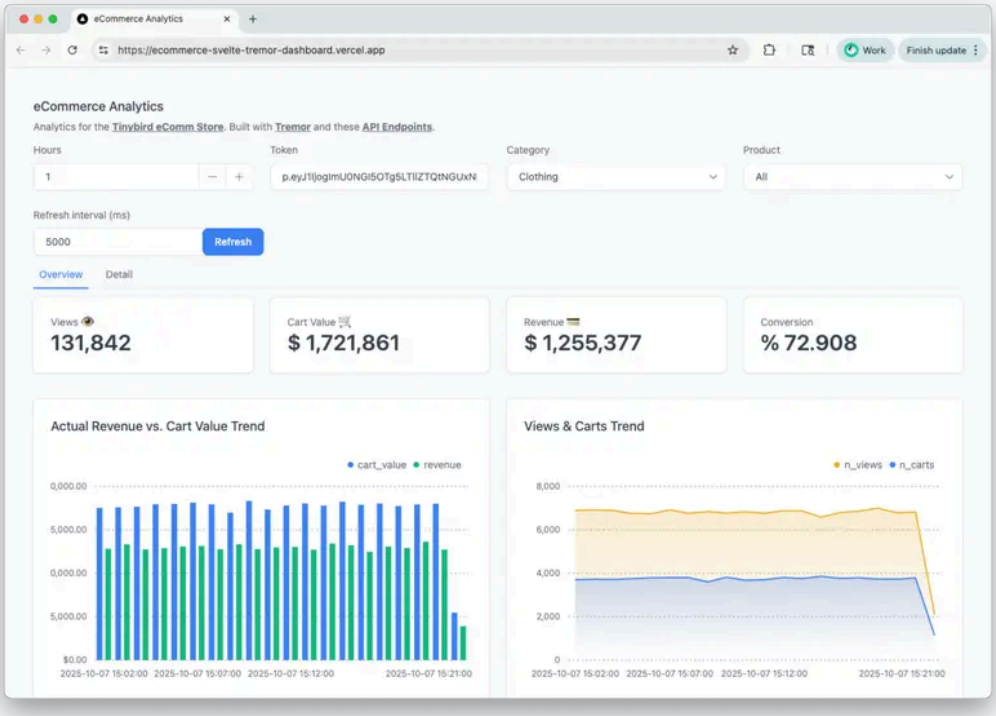
OpenAI seems to be pitching this as a cross-functional collab tool. Engineering writes the prompts, "stakeholders" drag and drop workflow blocks to connect everything, legal gets to add guardrails, and then engineering fixes everything so it actually works (maybe).



# Building a data-driven agent workflow with OpenAI Agent Builder and Tinybird MCP Server

We're going to build an example agent workflow using OpenAI's Agent Builder and the Tinybird MCP Server.

For this example, I'm going to use this public e-commerce demo built with Svelte from Tinybird. The project includes a demo Svelte e-commerce storefront as well as a backend revenue dashboard that visualizes revenue/sales metrics exposed via Tinybird API endpoints.




We can assume that somebody like a sales analyst might use this dashboard to keep track of sales performance from the store, find sales opportunities, identify stockout risks, etc.

But dashboards are dying, making way for chat-based data exploration, natural-language filtering and personalized analytics UIs. So let's add a helpful sales analyst chat agent to this demo.

We'll build a multi-agent workflow in Agent Builder, designing agents with fine-grained access to Tinybird data via the MCP Server, then embed the chat experience into the revenue dashboard using ChatKit.

The Tinybird MCP Server is secured via Tinybird auth tokens, so you can create multiple connections with fine-grained access control to resources, and even implement row-level security policies with JWT filters.

You can learn more in the official Tinybird MCP Server docs.

# TL;DR: The Worfklow

Here's the final workflow I build in Agent Builder:

The workflow has the following core steps:

[ 1 ] **Triage** the user input to assign a category ( `sales_performance` , `inventory` , or `other` )

[ 2 ] **Conditional logic** to route the request to a specialized agent based on the triage response

[ 3 ] **Specialized agents** for sales performance research and inventory research, with fine-grained access to specific Tinybird MCP tools optimized for the specific task

[ 4 ] **A general agent** for fallback, with full MCP access to the Tinybird workspace.

[ 5 ] A **guardrail** to check for any PII in the responses

[ 6 ] A **summarizing agent** to summarize the detailed research findings provided by the specialized agents

It's pretty basic, but it demonstrates a few nice things:

[ 1 ] **Using tokens to limit the data scope for specialize agents**. Rather than have a single, generalize agent that might need to make multiple tool calls to find the right data, generate a query, execute the query (and attempt retries), we build specialized agents that only have access to a small subset of tools so they can provide faster, more targeted responses.

# Triage: Routing the input to a specialized analytics agent

The first step in the workflow is to receive the user input and assign a category, one of "sales_performance", "inventory", or "other".

Take a look at the data lineage of the Tinybird data project:

There are two core pipelines here: one for inventory management, and one for sales and revenue KPIs.

Specifically, we have a Tinybird API endpoint that calculates the latest stock by comparing the latest inventory snapshot with recent sales from the online store:

This API uses a simplified lambda architecture that combines infrequent inventory snapshots with real-time transaction data to efficiently calculate real-time stock.

We also have several API endpoints that provide sales and KPIs for products, revenue trends, and conversion rate trends:

These APIs can be useful tools for answering sales performance questions.

So, we want to triage the user input to determine if their question is about inventory or about sales performance (or neither), and then route to the specialize agent.

For this, we use a lightweight "Triage" agent that uses minimal reasoning effort with this prompt:

```
You classify a user's natural language question into a category. Choose exactly one: "sales_performance" | "inventory" |
```

And it provides its response as a structured output based on this JSON schema:

```
{
  "type": "object",
  "properties": {
    "category": {
      "type": "string",
      "enum": [
        "sales_performance",
        "inventory",
        "other"
      ],
      "default": ""
```

# Specialized Agents

Rather than throw every user request at a single LLM with full access to the underlying Tinybird data, we create specialized agents that handle specific requests.

This is a common pattern in agent workflows (we do something similar in Tinybird Code) as it generally results in faster, more accurate responses.

Our workflow here is simple: we've created just two specialized agents and one general agent:

[ 1 ] **Sales Performance Agent** answers questions related to sales performance, revenue, and trends.

[ 2 ] **Inventory Agent** answers questions about inventory.

[ 3 ] **General Agent** answers all other questions.

To create our specialized agents, we use unique prompts and leverage the token-based auth of the Tinybird MCP Server, only exposing certain tools and data to the LLM.

## Sales Performance Agent

The Sales Performance agent receives user input routed to the `sales_performance` category by the triage agent.

It's prompt (basic):

```
You are an expert sales analyst that can analyze data and spot trends about sales performance for an e-commerce store. Yo
```

```
TOKEN `sales_performance_mcp_read` READ

DESCRIPTION >
  An API that calculates product-level sales performance KPIs, including views, add to cart value, revenue, and conversio

NODE filter_products
SQL >
  %
  SELECT *
  FROM products
  ...
```

With token-based auth, the agent can fetch predefined metrics using existing APIs (fast, deterministic), or conduct further exploratory analysis on the underlying dataset if needed (slow, comprehensive).

The agent configuration looks like this in Agent Builder:

So, when a user input is categorized as related to "sales performance", it gets routed to the Sales Performance agent which can then fetch these APIs or explore the underlying events table.

## Inventory Agent

The Inventory agent receives user input routed to the `inventory` category by the triage agent.

It's prompt (basic):

The agent configuration looks like this in Agent Builder:

## General agent

In the event that triage can't categorize a request as sales performance or inventory, a fallback route sends the request to a general agent.

This agent has a more generic prompt:

```
You are a data analyst with access to an explore_data agent tool from the Tinybird MCP server that can explore ecommerce
```

In addition, it can only use the `explore_data` tool, though it is given full read access to the entire dataset.

The configuration looks like this:

### Why only one tool?

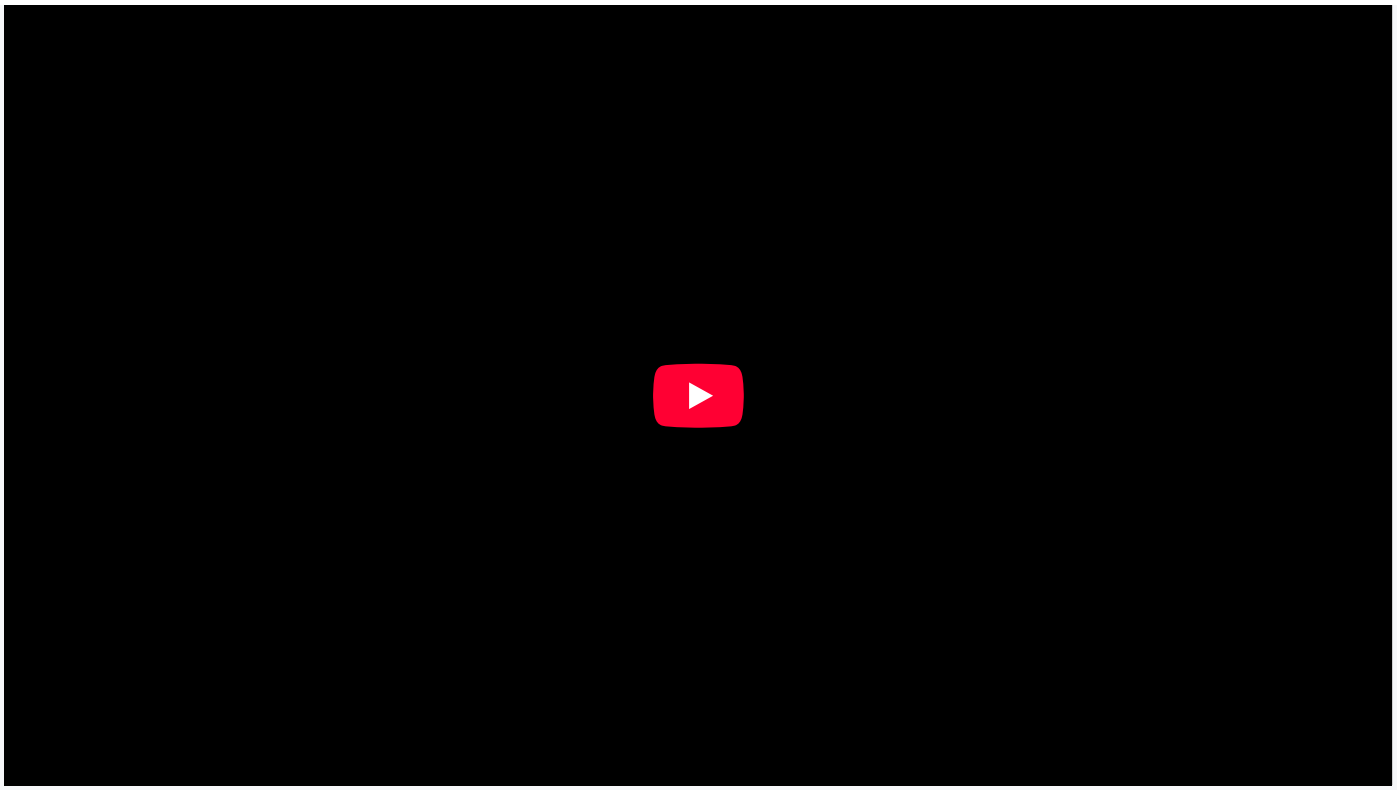
The `explore_data` tool from the Tinybird MCP Server is exceptionally powerful. In fact, it isn't just a tool, but rather a server-side agent that has deep context about the data in Tinybird, including schemas, performance logs, and other metadata.

So, if a user request is generic, we can simply pass it to this `explore_data` tool that Tinybird offers and not have to worry about error-prone LLM SQL generation, and Tinybird's agent will autonomously arrive at an answer.

This takes a bit longer, as it may require multiple tool calls and exploratory loops, so we don't prioritize this route. Still, it's perfect as a fallback to ensure the user gets an answer, even if it takes a bit longer.

## Final result

Here's a quick video showing the agent in action. Note how inventory-related requests get routed to the Inventory Agent, Sales Performance questions to the Sales Performance Agent, and anything else to the general agent. In each case, the agent has the tools it needs thanks to the Tinybird MCP Server to fetch relevant, real-time data and provide insight back to the user.



## Integrating the data-driven agent with ChatKit

The last step is to integrate this agent workflow into my revenue dashboard. I could either use ChatKit or the Agents SDK, and OpenAI provides some reference implementations:

- ChatKit Starter App
- ChatKit Advanced Samples
- OpenAI Agents JS SDK

Perhaps I'll follow up on that integration in a later post.



## Closing points

More resources here:

- Introducing the Tinybird MCP Server
- Video: Build a CLI agent for analytics with the Vercel AI SDK and Tinybird MCP Server
- Analytics Agent Template from Tinybird

Share it!

## Related posts

AI x Data

Jun 23, 2025

### Building an autonomous analytics agent with Agno and Tinybird

Alberto Romeu
Software Engineer

AI x Data

Jul 02, 2025

### How to bu by-step

Cameron Arch
Tech Writer

# Skip the infra work. Deploy your first ClickHouse project now

Get started for free

Read the docs

Vector search

All systems operational