Coding Nexus

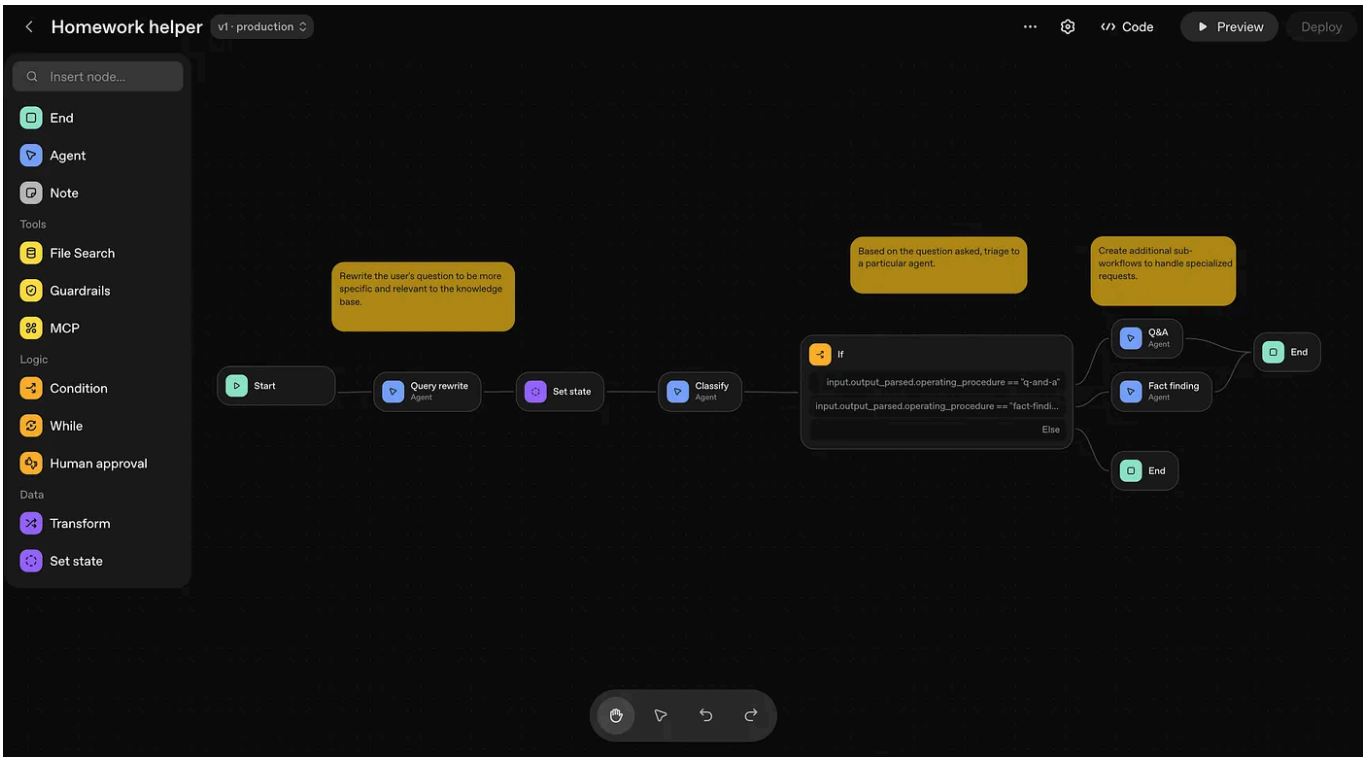
Member-only story

OpenAI's AgentKit: A Simpler Way to Build and Deploy AI Agents

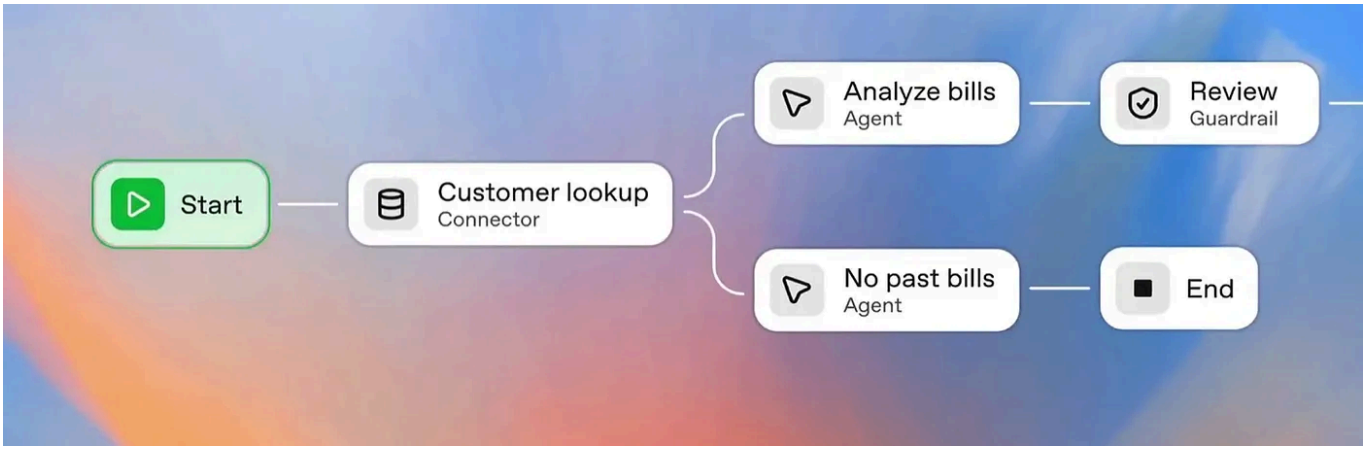
Code Coup Follow 6 min read · 2 days ago

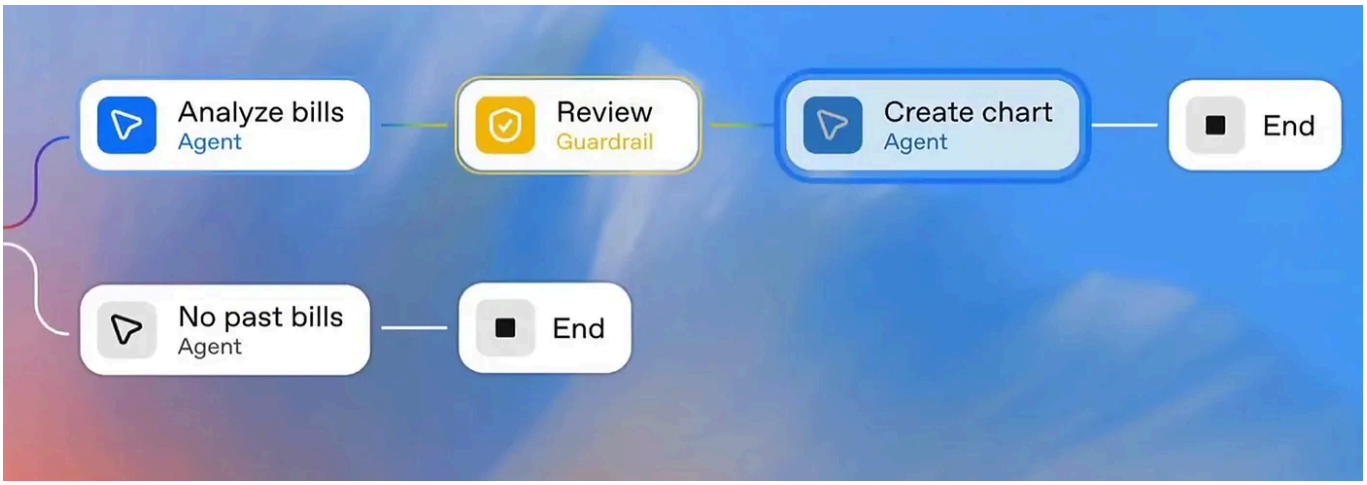
If you've been following the AI world lately, you've probably noticed a new buzzword frequently — **agents**. Everyone is either building one, talking about one, or trying to create one that actually works.

The problem? Building an AI agent from scratch is difficult. You're managing models, APIs, workflows, safety filters, and UI setups — and by the time you're finished, it feels like you've built a small operating system just to run one bot.



OpenAI clearly anticipated this. They released **AgentKit**, a toolkit that simplifies the building, testing, and launching of AI agents.
Think of it as the “no-fuss” way to go from idea to production without getting lost in boilerplate code.





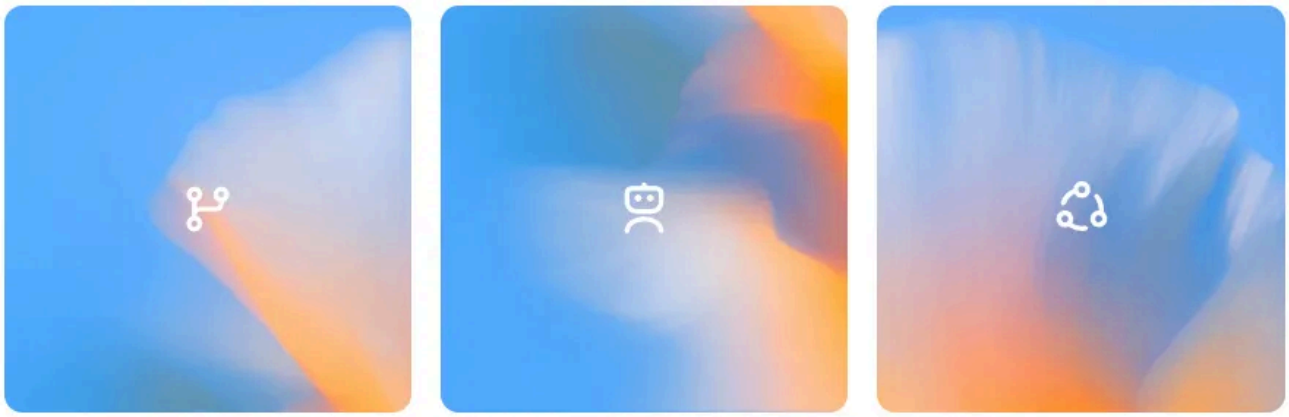
What Exactly Is AgentKit?

In simple terms, **AgentKit is a comprehensive toolkit for developing agentic systems** — everything you need to create, test, and deploy your own AI agent, all in one platform.

It comes with four main parts:

- **Agent Builder** — a visual editor for building workflows
- **ChatKit** — a chat interface you can plug straight into your app
- **Guardrails** — built-in safety and moderation tools
- **Evals** — a way to test and improve your agent's performance

You can design your agent visually, integrate it into your app, and fine-tune it — all within the same ecosystem. No extra setup, missing config files, or switching between ten different tools.



Step 1: Build the Workflow Visually

At the core of AgentKit is **Agent Builder**, a drag-and-drop editor that clearly demonstrates how your agent thinks.
Each block (or *node*) performs a function — one might run a model, another might call an API, store memory, or make a request decision.

If you've used tools like **n8n**, **LangGraph**, or **Node-RED**, you'll understand it immediately.

Say you're creating a basic "homework helper" bot. Your flow could be like this:

```
[User Question] → [Rephrase] → [Search Notes] → [Generate Answer]
```

Behind the scenes, Agent Builder is handling numerous LLM calls for you. However, you get to *watch* it happen in real-time.

Here's approximately what that would look like in plain code:

```
question = input("Ask me anything: ")
cleaned = model.generate(f"Rephrase clearly: {question}")
notes = vector_search(cleaned)
answer = model.generate(f"Answer using context: {notes}")
print(answer)
```

Instead of writing that out manually, connect those nodes visually and hit **Publish**. That's it — your agent's logic is ready to go.

Step 2: Add It to Your App Using ChatKit

Once your agent is working, you'll want people actually to interact with it. That's what **ChatKit** does — it's an embeddable chat interface that hooks directly into your workflow.

You add a few lines of code, connect it to your agent, and it's live.

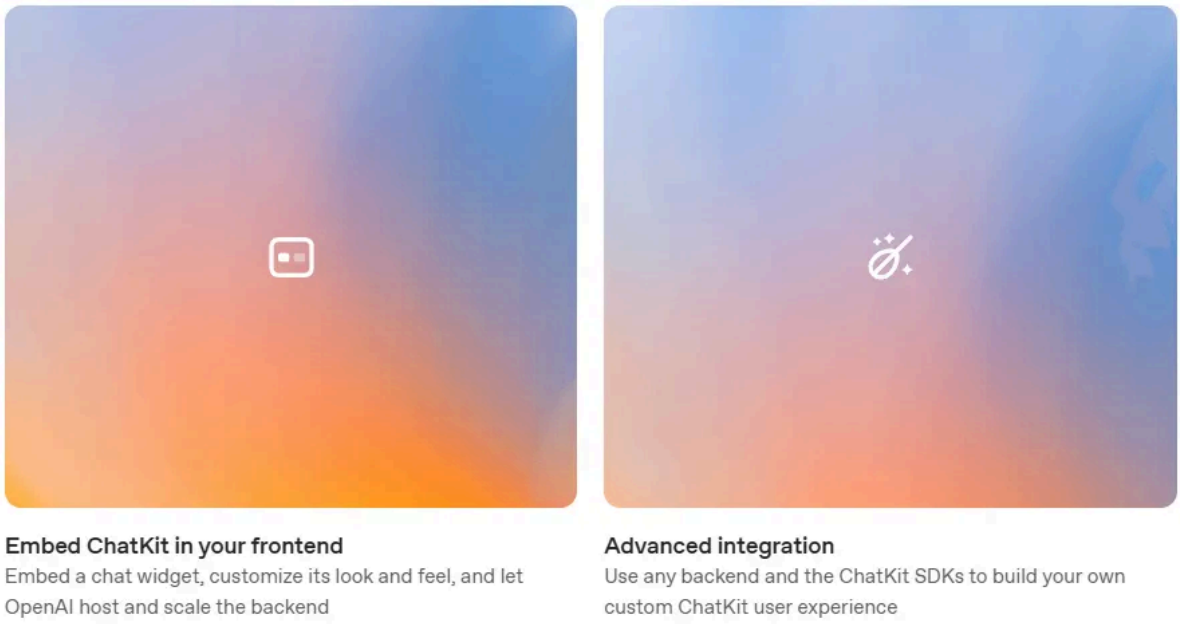
Generate a client token on your server.

This snippet initialises a FastAPI service that solely creates a new ChatKit session using the OpenAI Python SDK and returns the session's client secret.

```
from fastapi import FastAPI
from pydantic import BaseModel
from openai import OpenAI
import os
```

```
app = FastAPI()
openai = OpenAI(api_key=os.environ["OPENAI_API_KEY"])

@app.post("/api/chatkit/session")
def create_chatkit_session():
    session = openai.chatkit.sessions.create({
      # ...
    })
    return { client_secret: session.client_secret }
```



Embed ChatKit in your frontend
Embed a chat widget, customize its look and feel, and let OpenAI host and scale the backend

Advanced integration
Use any backend and the ChatKit SDKs to build your own custom ChatKit user experience

In your server-side code, provide your workflow ID and secret key to the session endpoint. See the chatkit-js repo on GitHub.

```
export default async function getChatKitSessionToken(
deviceId: string
): Promise<string> {
const response = await fetch("https://api.openai.com/v1/chatkit/sessions", {
    method: "POST",
    headers: {
    "Content-Type": "application/json",
    "OpenAI-Beta": "chatkit_beta=v1",
```

```
    Authorization: "Bearer " + process.env.VITE_OPENAI_API_SECRET_KEY,
    },
    body: JSON.stringify({
    workflow: { id: "wf_68df4b13b3588190a09d19288d4610ec0df388c3983f58d1" },
    user: deviceId,
    }),
});

const { client_secret } = await response.json();

return client_secret;
}
```

In your project directory, install the ChatKit React bindings.

```
npm install @openai/chatkit-react
```

Add the ChatKit JS script to your page. Insert this snippet into your page's section or wherever you load scripts, and the browser will fetch and run ChatKit for you.

```
<script
src="https://cdn.platform.openai.com/deployments/chatkit/chatkit.js"
async
></script>
```

Integrate ChatKit into your UI. This code retrieves the client secret from your server and mounts a live chat widget connected to your backend workflow.

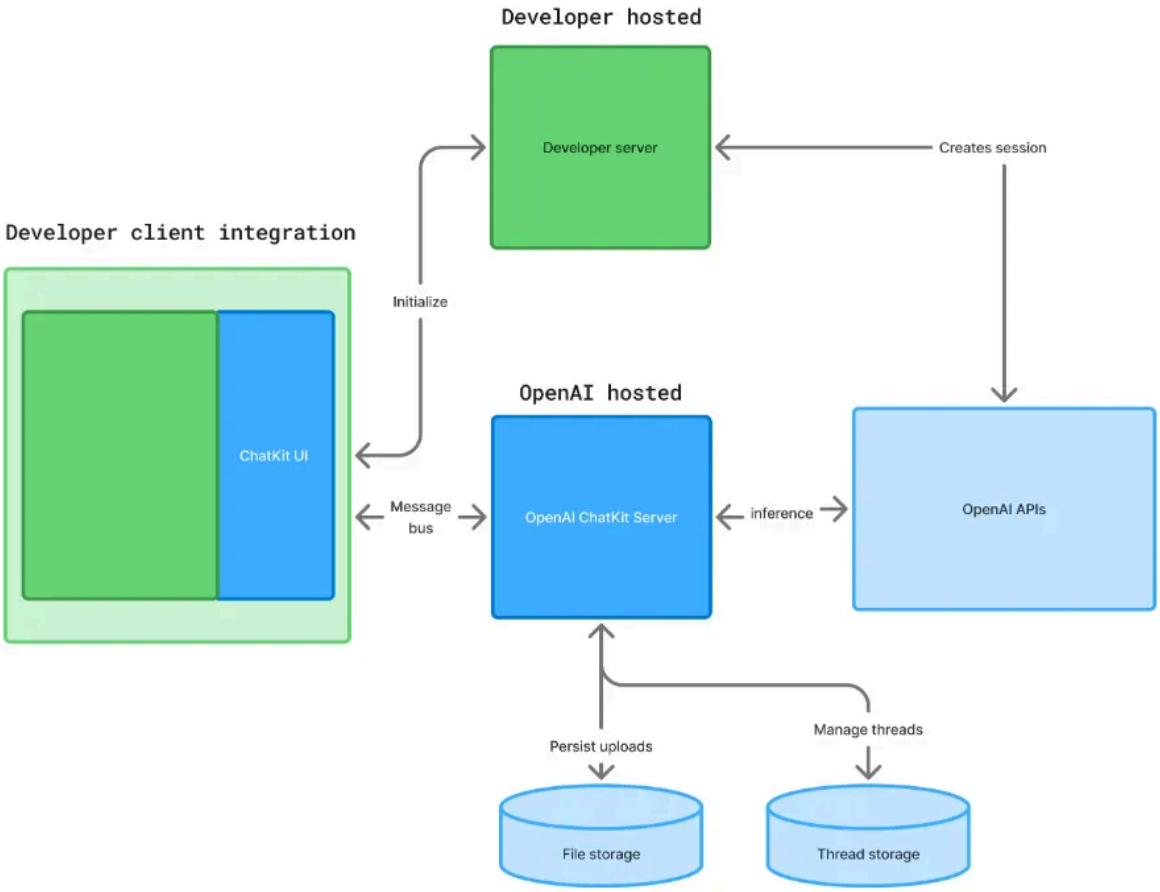
```
import { ChatKit, useChatKit } from '@openai/chatkit-react';

   export function MyChat() {
     const { control } = useChatKit({
       api: {
         async getClientSecret(existing) {
           if (existing) {
             // implement session refresh
           }

           const res = await fetch('/api/chatkit/session', {
             method: 'POST',
             headers: {
               'Content-Type': 'application/json',
             },
           });
           const { client_secret } = await res.json();
           return client_secret;
         },
       },
     });

     return <ChatKit control={control} className="h-[600px] w-[320px]" />;
   }
```

It handles history, file uploads, tool calls, and even themes automatically. You can customise the look to match your brand or keep OpenAI's default style while you prototype.



Step 3: Test and Improve with Evals

Once your agent is live, the next question is*: Is it actually good?*

That's where **Evals** comes in.
It's like automated grading for your AI agent.

You can run your agent against datasets or sample conversations, then use graders (scripts or models) to score how well it performs.

Here's a quick TypeScript example:

```
import OpenAI from "openai";

const client = new OpenAI();

const run = await client.evals.create({
  name: "knowledge-assistant-regression",
  data_source: {
      type: "responses",
      filter: {
          project_ids: ["proj_123"],
          prompt_ids: ["pmpt_456"],
      },
  },
  graders: [
      {
          type: "score_model",
          model: "gpt-4.1-mini",
          rubric: "Score from 1-5 based on factual accuracy.",
      },
  ],
});
```

You can run multiple graders — one to check for factual accuracy, another for tone or safety. It's essentially **CI/CD for AI quality**.

Built-In Guardrails for Safety

One of the smartest moves OpenAI made was baking **guardrails** right into AgentKit.
These handle input and output filtering automatically — protecting your system from prompt injection, data leaks, or off-topic responses.

So when your agent's out in the wild, it stays safe and reliable without constant babysitting.

The Big Picture

AgentKit is built around one idea: **AI agents shouldn't be hard to build.**

Here's the quick rundown:

What you do	Tool	What it handles
Design logic	Agent Builder	Visual workflow creation
Launch in app	ChatKit	Chat interface and deployment
Test quality	Evals	Performance tracking and grading

You can remain within OpenAI's ecosystem throughout — from idea to launch to performance checks. No context switching, no mess scripts.

Final Thoughts

I've tried numerous agent frameworks this year.
Most start out exciting and end up being a tangle of APIs, rate limits, and debugging nightmares.

AgentKit feels different. It's polished, consistent, and actually works out of the box. You build with blocks, hit publish, and you end up with something real — complete with testing, safety, and UI.

If you're already working with OpenAI's models, this one's a clear choice. Start small with the **Agent Builder**, integrate it into your frontend using **ChatKit**, and monitor performance with **Evals**.

It's likely the most streamlined end-to-end agent development pipeline available today — and it finally makes "building an AI agent" feel as simple as it should be.

OpenAI | ChatGPT | AI | Ai Agent | Ai Agent Development

Published in Coding Nexus

7.2K followers · Last published 1 hour ago

Follow

Coding Nexus is a community of developers, tech enthusiasts, and aspiring coders. Whether you're exploring the depths of Python, diving into data science, mastering web development, or staying updated on the latest trends in AI, Coding Nexus has something for you.

Written by Code Coup

2.1K followers · 1 following

Follow

Code Coup: Seize the Code, Stage a Coup!

Responses (1)

Bgerby

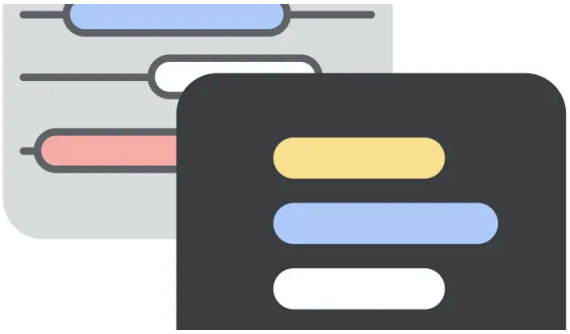
What are your thoughts?

Tom Parish
2 days ago

Very helpful, thank you!

Reply

More from Code Coup and Coding Nexus

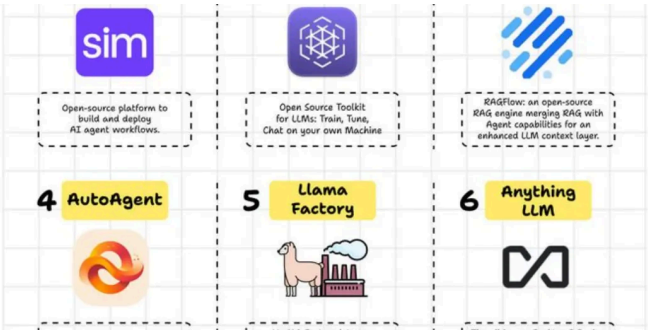


Code Coup

LangExtract: Google's Game-Changing Python Library for Data...

Ever stared at a mountain of words, a report, a book, maybe a stack of doctor's notes, and...

Aug 13 435 6

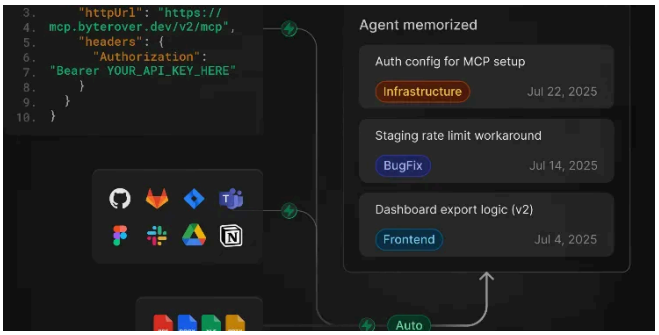


In Coding Nexus by Civil Learning

6 Open-Source AI Projects You Must Try (Agents, RAG & Fine-...

The AI world is complex right now. Every week there's a new repo, a new framework, and a...

Sep 18 647 5

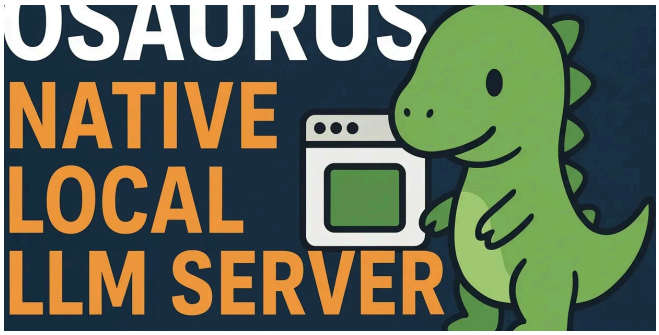


In Coding Nexus by Algo Insights

4 Open-Source Tools That Made Me Rethink My Dev Setup

I've been coding for a while now. Most of the tools we use every day... they've been the...

Sep 3 377 8



In Coding Nexus by Code Coup

Osaurus: A Native Local LLM Server for Apple Silicon

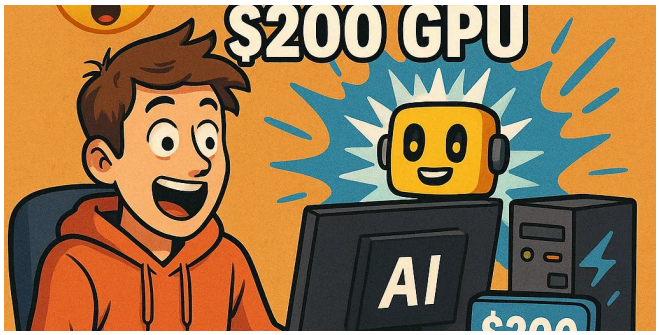
If you've ever tried running large language models (LLMs) locally on your Mac, you...

Sep 7 224 4

See all from Code Coup

See all from Coding Nexus

Recommended from Medium



In Data Science Collective by Erdogan T

Build Your Private Language Model: Local and Specialized For...

A complete step-by-step guide from setup to deployment of local language models, makin...

6d ago 644 4

In Towards Deep Learning by Sumit Pandey

Meet oLLM: The Secret Sauce to Run Huge AI on Tiny Hardware

oLLM slashes LLM memory use: Run 100k context GPTs on 8GB GPUs. A lightweight...

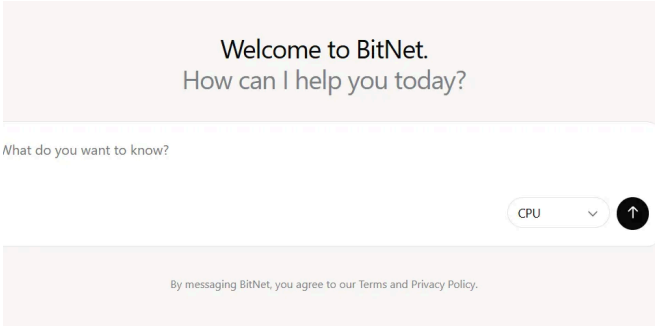
Oct 2 150 9

In Artificial Intelligence in Plain E... by Prithwish N...

5 Essential MCP Servers That Give Claude & Cursor Real Superpower...

Transform Claude & Cursor into web scraping, browser-controlling automation engines. 5...

Sep 27 182 1



In Coding Nexus by Algo Insights

Microsoft Just Declared War on the GPU Mafia: Meet bitnet.cpp

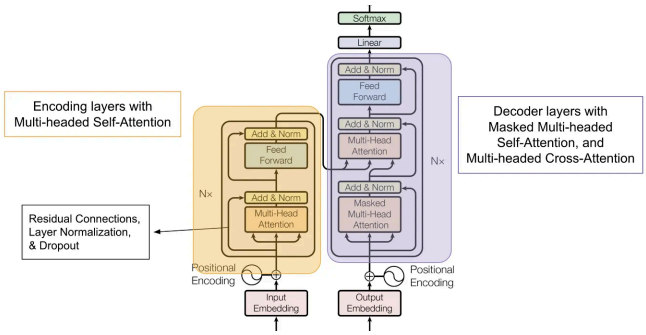
Bitnet.cpp will break the GPU Lock

4d ago 244 3

In Vibe Coding by Alex Dunlop

This Just Became the Most Important Tool in Frontend (Here'...

The goldmine I've always wanted solved



In Towards AI by Ashish Abraham

No Libraries, No Shortcuts: LLM from Scratch with PyTorch

The no BS guide to build, train, and fine-tune a Transformer architecture from scratch

Oct 1 101 5

Oct 2 646 7

See more recommendations