Member-only story

Pepper: Building Real-Time Proactive AI Assistants with Event-Driven Architecture

Civil Learning Follow 4 min read · 1 day ago

3 1

Most bots you use today are passive. You ask, they answer, and then they just… sit there. Nothing happens unless you poke them.

That is not how a proper assistant should function.

A faithful assistant watches over things. It reminds you of tasks before you forget.

It notices when a critical email arrives and highlights it. It could help you prepare notes for your next meeting.

That’s the idea behind **Pepper**.

It’s more than just another “chatbot.”

It’s a small framework that Pepper created to operate assistants in real-time. Instead of waiting for commands, it responds to events occurring around you.

What’s Different About It?

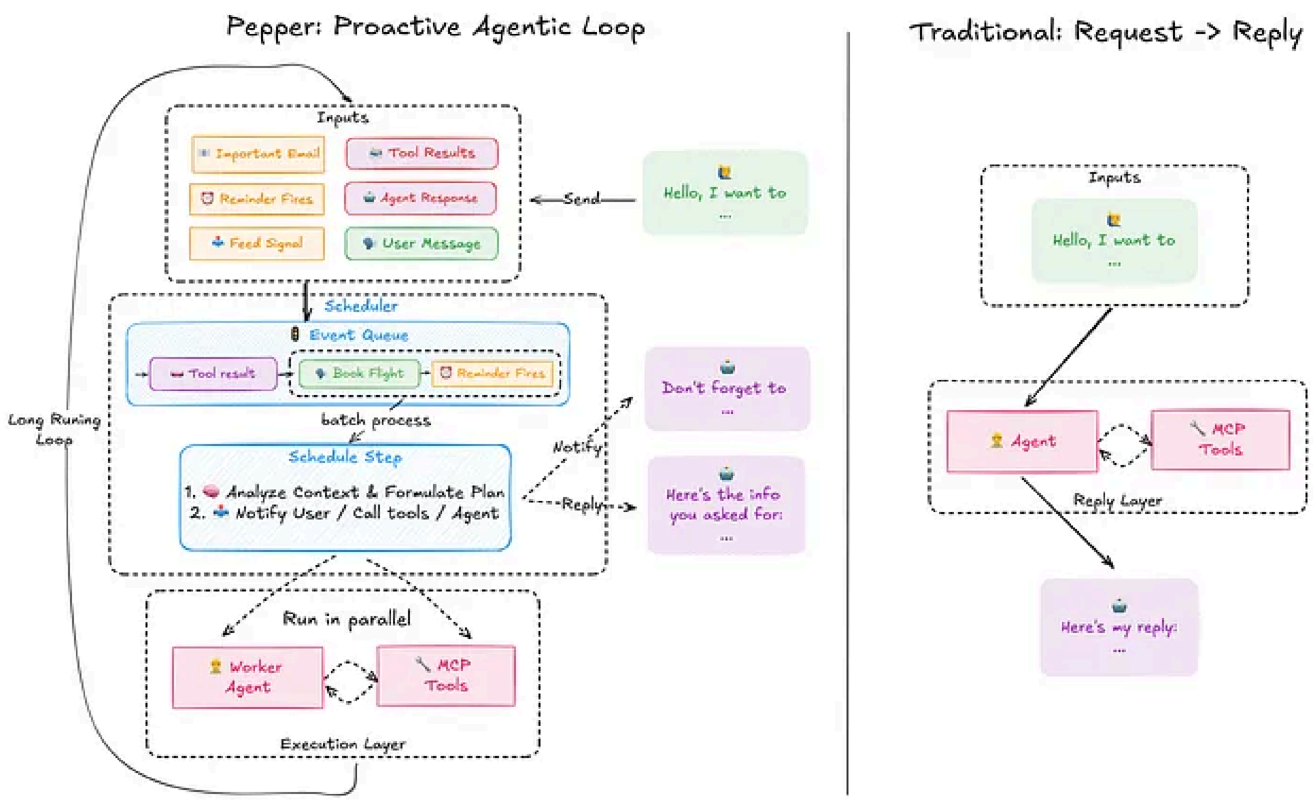
Old world:

```
You → Question → Bot → Answer → End
```

New with Pepper:

```
Email arrives → Pepper notices → Pepper summarizes → Pepper asks if you want to
```

All of that happens without you typing a single thing. That’s the shift. To make it work, Pepper had to abandon the old synchronous pipelines and switch to an event-driven loop that continually runs.



Traditional chatbots (right) wait for user prompts, while Pepper (left) operates in a continuous, event-driven loop — processing inputs, acting independently, and proactively engaging with the user.

The Pieces That Make It Work

I like to explain Pepper as three parts that keep talking to each other:

- **Feeds:** They're like sensors. They watch your inbox, calendar, and chat threads. They clean the raw mess into a short, proper signal.
- **Scheduler:** This is the brain. It examines those signals and determines what should happen.
- **Workers:** These are the doers. They send the email, run the analysis, and fetch the info.

There's also a **Context Store**, which is just shared memory. Think of it as the whiteboard where everyone writes notes so the others know what's going on.

```
[Inbox / Calendar / Chat] → Feeds → Scheduler → Workers
                                        |
                                   Context Store
```

The Scheduler: The Brain Loop

The Scheduler doesn't stop. It picks up events one by one, checks the current context, and then asks the LLM to determine the following action.

A rough sketch:

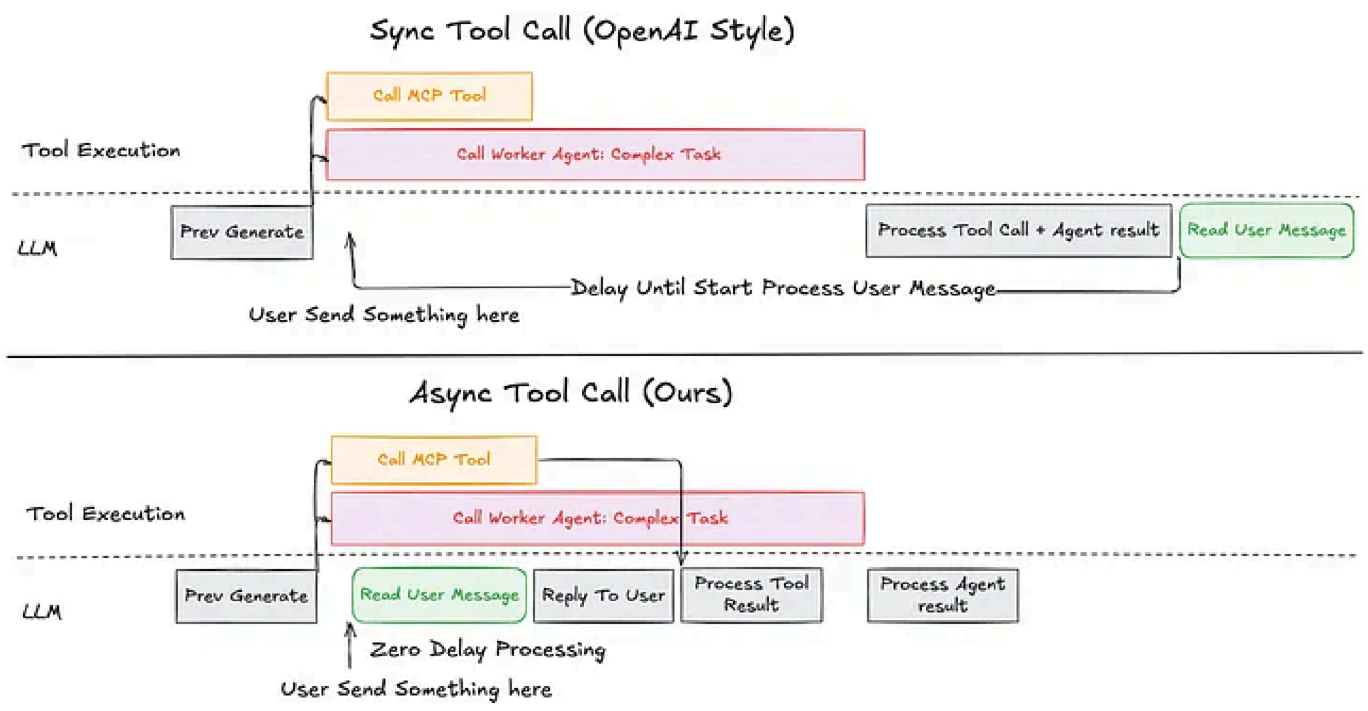
```
async def scheduler_step(self):
    events = await self.get_batch_events()
    self.state_tracker.add_events(events)

    messages = [
        {"role": "system", "content": "You are Pepper's scheduler."},
        {"role": "user", "content": self.state_tracker.get_user_prompt()},
    ]

    response = await create_completion(messages, self.tools)
    self.state_tracker.add_event(response)

    if response.tool_calls:
        for tool_call in response.tool_calls:
            self.tool_call_engine.schedule(tool_call)
```

Notice something: tool calls aren't blocking. In most systems, the assistant freezes until a tool finishes. Here, Pepper schedules it and keeps moving. That means you can still interact with it while it's busy.



The sync tool call requires the tool's result to be fully returned and processed by the LLM before it can accept a new user message, causing a noticeable delay for the user.

In contrast, the Pepper's async tool call replies to the user immediately, resulting in no delay.

Feeds: Cutting Out the Noise

Feeds are simple: they take messy input and refine it.

Example: Instead of dumping a 500-word email, the feed says:

```
{
  "id": "evt_123",
  "content": "Action requested by Alice on 'Project Phoenix', due tomorrow."
}
```

Much easier for the Scheduler to act on.

A simple example feed might look like this:

```
@subscriber.on_context_update(namespaces=["raw.email"])
async def email_feed(update):
    raw_email = update.context.data
    if "urgent" in raw_email["subject"].lower():
        signal = {
            "id": f"signal_{raw_email['id']}",
            "content": f"Urgent email from {raw_email['sender']}"
        }
        await context_store.store(signal, namespace="signals.email")
```

Feeds can be as simple as this filter or more complex agents on their own. Depends on the job.

Workers: The Hands

Workers are where things get done. The Scheduler hands them tasks, and they either:

- Remember things (stateful workers, good for ongoing threads), or

- Just do a one-off job and exit (stateless workers).

A stateful worker can even pull its own memory back from the Context Store before resuming:

```python
latest_contexts = await context_store.query(
    ContextFilter(namespaces=["memory-email-agent"], limit=1)
)

if latest_contexts:
    self.state = AgentState(**latest_contexts[0].data)
```

That way, if it were halfway through handling an email chain, it won’t start from scratch the next time.

The Glue: Context Store

Everything passes through the Context Store. New email? Store it. Does it feed processes? Store the signal. Scheduler picks it up from there. Workers update their memory.

In short, it’s the shared layer that makes Pepper feel like one system instead of disconnected pieces.

Here’s a flow:

```
@app.on_event("new_email")
async def ingest_email(data):
    await context_store.store(data, namespace="raw.email")

@subscriber.on_context_update(namespaces=["raw.email"])
async def process_email(update):
    processed = process_email_signal(update.context)
    await context_store.store(processed, namespace="signals.processed")
@subscriber.on_context_update(namespaces=["signals.*"])
async def add_to_queue(update):
    await self.priority_queue.put(update)
```

Why It Matters

Because once you have this structure, the assistant feels alive.

- It doesn't just sit around.
- It reacts instantly to new stuff.
- It handles long tasks in the background without freezing.

Imagine this:

- You get a meeting invite.
- Before you click anything, Pepper has already pulled the latest project notes and prepped a summary.
- During the meeting, take notes.
- After, it drafts the follow-up.

That's the difference between a chatbot and an assistant.

Pepper is still young, but it demonstrates what's possible when you stop treating assistants like Q&A machines and start wiring them like real-time systems.

AI | Ai Tools | Ai Agent | Generative Ai Tools | ChatGPT

Responses (1)

Bgerby

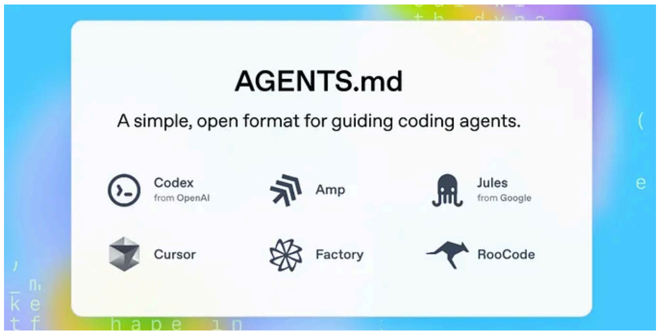
What are your thoughts?

shaytac
3 hours ago (edited)

I was wondering why no links to the repo of this frameworks. Would be nice to credit the actual paper. https://agentica-project.com/post.html?post=pepper.md.

Reply

More from Civil Learning and Coding Nexus

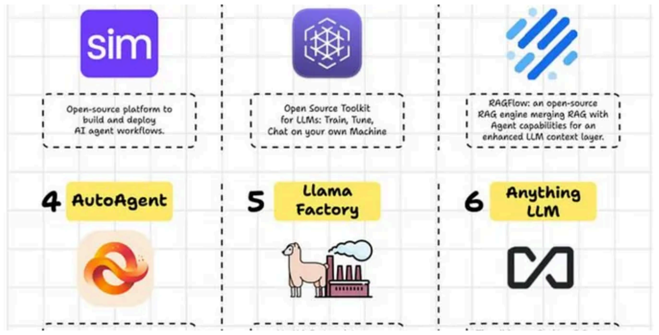


In Coding Nexus by Civil Learning

AGENTS.md: The File That Saves You From Dumb AI Code

If you've ever thought, "This AI code is smart but also dumb," you'll get this.

Sep 18 562 8

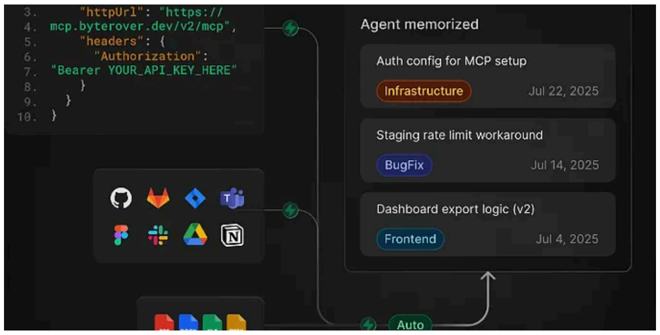

sim
Open-source platform to build and deploy AI agent workflows.
Open Source Toolkit for LLMs: Train, Tune, Chat on your own Machine
RAGFlow: an open-source RAG engine merging RAG with Agent capabilities for an enhanced LLM context layer.
4 AutoAgent
5 Llama Factory
6 Anything LLM

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



```
    "httpUrl": "https://
mcp.byterover.dev/v2/mcp",
    "headers": {
      "Authorization":
  "Bearer YOUR_API_KEY_HERE"
    }
  }
}
```

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 3 369 7

In Coding Nexus by Civil Learning

How I Stopped Paying $200 for Claude and Got the Same Results...

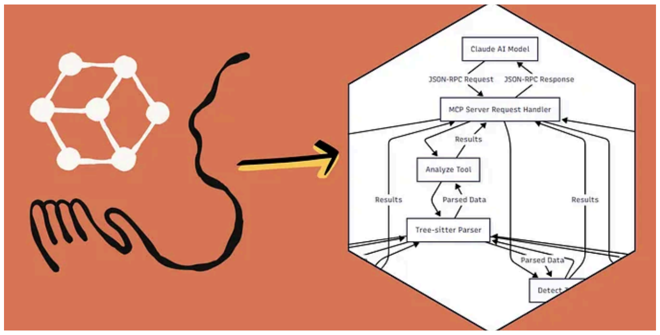
I'll be honest: my Claude bill was starting to hurt. $200 every month, just so I could get...

Sep 7 334 19

See all from Civil Learning

See all from Coding Nexus

Recommended from Medium



Joe Njenga

I Asked Claude 4.5 to Build Me a Complex MCP Server (It Was So…

If you want to build your custom MCP server, stop wasting time thinking, coding, or trying…

3d ago 359 6

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…

5d ago 465 4

5 Essential MCP Servers That Give Claude & Cursor Real Superpowers (2025)

How to set up & configure free, open-source Model Context Protocol servers that turn your AI assistant into a web scraping, browser-controlling automation engine.

Prithwish Nath

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…

In Coding Nexus by Algo Insights

DSPy: Stop Prompting, Start Programming Your AI

If you've ever spent hours tweaking prompts, you know how frustrating it can be.

Sep 27 158 1

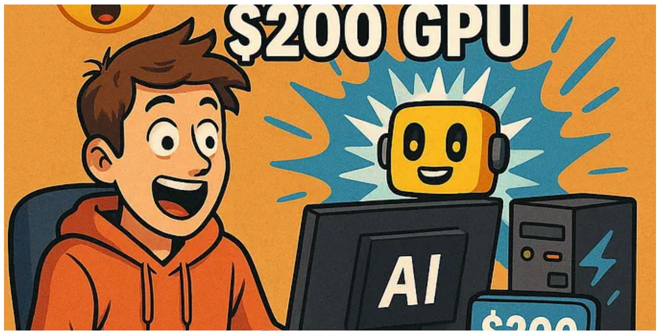
6d ago 240 7

In Vibe Coding by Alex Dunlop

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

The goldmine I've always wanted solved

6d ago 66 3



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...

6d ago 143 6

See more recommendations