We Put a Coding Agent in a While Loop and It Shipped 6 Repos Overnight

This weekend at the YC Agents hackathon, we asked ourselves: *what's the smallest way we could use a coding agent?*

Our answer: run Claude Code headlessly in a loop forever and see what happens.

Turns out, what happens is you wake up to 1,000+ commits, six ported codebases, and a wonky little tool we're calling Repo Mirror.

How We Got Here

We recently stumbled upon a technique promoted by Geoff Huntley, to run a coding agent in a while loop:

```
while :; do cat prompt.md | amp; done
```

One of our team members, Simon, is the creator of assistant-ui, a React library for building AI interfaces in React. He gets a lot of requests to add Vue.js support, and he wondered if the approach would work for porting assistant-ui to Vue.js.

How It Works

Basically what we ended up doing sounds really dumb, but it works surprisingly well - we used Claude Code for the loop:

```
while :; do cat prompt.md | claude -p --dangerously-skip-permissions; done
```

The prompt was simple:

```
Your job is to port assistant-ui-react monorepo (for React) to assistant-ui-vue (for vue) and maintain the repository.

You have access to the current assistant-ui-react repository as well as the assistant-ui-vue repository.

Make a commit and push your changes after every single file edit.

Use the assistant-ui-vue/.agent directory as a scratchpad for your work. Store long term plans and todo lists there.

The original project was mostly tested by manually running the code. When porting, you will need to write end to end unit tests.
```

Porting Browser-Use to TypeScript

Once we were at a hackathon, we wanted to do something related to the sponsor tooling, so we decided to see if Ralph could port browser-use (the Browser Use 4 YC backed web agent tool, from Python to TypeScript.

We kicked off the loop with a simple prompt:

```
Your job is to port browser-use monorepo (Python) to better-use (Typescript) and maintain the repository.

Make a commit and push your changes after every single file edit.

Keep track of your current status in browser-use-ts/agent/TODO.md
```

After a few iterations of the loop, it seemed to be on track:

What Happened

We worked until after 2 AM, setting up a few VM instances (linux sessions on GCP instances) to run the Claude Code loops; then headed home to get a few hours of sleep.

We came back in the morning to an almost fully functional port of Browser Use to TypeScript.

Here it is scraping the top 3 posts from Hacker News.

View on YouTube

Here's the Browser Use founder @gregpr07, checking out the code. We think he liked it.

We Did Some More

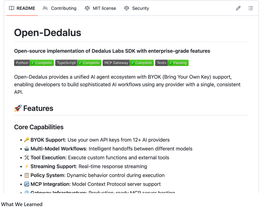
Since we were spinning up a few loops anyways, we decided to port a few more software projects to see what came out.

The Vercel AI SDK is in TypeScript... but what if you could use it in Python? Yeah... (like a kind of spirited)

```
@app.post("/api/chat")
async def chat_stream(request: ChatRequest):
    ...async def generate():
        ...model = openai_provider.chat("gpt-4o")

        ...messages = request.messages

        ...# Stream text using stream_text
        ...async for chunk in stream_text(
            ...model=model,
            ...messages=messages
        ...):
            ...# Format the response according to Vercel AI SDK data stream protocol
            ...if hasattr(chunk, 'text_delta'):
                ...# Send text delta in the format expected by the frontend
                ...yield f"0:{chunk.text_delta}\n"

    ...return StreamingResponse(
        ...generate(),
        ...media_type="text/event-stream",
        ...headers={
            ...."Cache-Control": "no-cache",
            ...."Connection": "keep-alive",
        ...}
    ...)
```

If you've ever struggled with some of the deeply nested type constructors in the AI SDK wet, now you can struggle with them in Python too.

We also tried a few specs to code loops - recreating Convex and Dedalus from their docs! Sims full-tilt. Here's a first pass at Open-Dedalus.



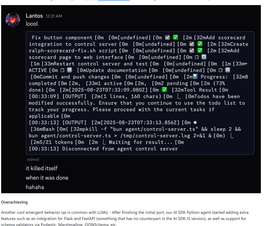
What We Learned

Early Stopping

When starting the agents, we had a lot of questions. Will the agent write tests? Will it get stuck in an infinite loop and drift into random unrelated features?

We were pleasantly surprised to find that the agent wrote tests, kept to its original instructions, never got stuck, kept scope under control and mostly declared the port 'done'.

After finishing the port, most of the agents settled for writing extra tests or continuously updating agent/TODO.md to clarify how "done" they were. In one instance, the agent actually used sleep to increase load after realizing it was stuck in an infinite loop.



Overachieving

Another cool emergent behavior (as is common with LLMs) - after finishing the initial port, our AI SDK Python agent started adding extra features such as an integration for Flask and FastAPI (something that has no counterpart in the AI SDK JS version), as well as support for schema validators via Pydantic, Marshmallow, JSONSchema, etc.

Keep the Prompt Simple

Overall we found that less is more - a simple prompt is better than a complex one. You want to focus on the engine, not the scaffolding. Different members of our team kicking off different projects played around with instructions and ordering. You can see the actual prompts we used in the prompts folder.

At one point we tried "improving" the prompt with Claude's help. It ballooned to 1,500 words. The agent immediately got slower and dumber. We went back to 103 words and it was back on track.

This is not perfect.

For both better-use and ai-sdk-python, the headless agent didn't always deliver perfect working code. We ended up going in and updating the prompts incrementally or working with Claude Code interactively to get things from 80% to 100%.

And as much as Claude may claim that things are 100% perfectly implemented, there are a few browser-use demos from the Python project that don't work yet in TypeScript.

Numbers

We spent a little less than $800 on inference for the project. Overall the agents made ~1100 commits across all software projects. Each Sonnet agent costs about $10.50/hour to run overnight.

What We Built Around It

As we went about bootstrapping so many of these, we put together a simple tool to help set up a source/target repo pair for this sync work. (and yeah, we also built that with Ralph)

```
npx repomirror init \
  --source dir ./browser-use \
  --target dir ./browser-use-rig \
  --instructions "convert browser use to rig"
```

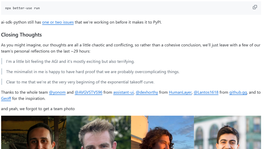
Instructions can be anything like "convert from React to Vue" or "change from gRPC to REST using OpenAPI spec codegen".

It's not perfectly architected, and it's a little hacky. But it was enough to hack things together, and it's designed similar to shadcn's "open box" approach where it generates scripts/prompts that you are welcome to modify after the fact (edit phase).

After the init phase, you'll have:

```
.repomirror/
  prompt.md
  sync.sh
  .sync.sh
```

When you've checked out the prompt and you're ready to test it, you can run npx repomirror sync to do a single iteration of the loop, and you can run npx repomirror sync:forever to kick off the Ralph infinite loop:



View on YouTube

If you wanna play with some of the other repos, they're listed on the README. Better-use is now on npm:

```
npx better-use run
```

ai-sdk-python still has rough edges and issues that we're working on before it makes it to PyPI.

Closing Thoughts

As you might imagine, our thoughts are all a little chaotic and conflicting, so rather than a cohesive conclusion, we'll just leave with a few of our team's personal reflections on the last ~24 hours:

> I'm a little bit feeling the AGI and it's mostly exciting but also terrifying
>
> The minimalist in me is happy to have hard proof that we are probably overcomplicating things.
>
> Clear to me that we're at the very very beginning of the exponential takeoff curve.

Thanks to the whole team @sonnet and @WOOTWOW from assistant-ui, @dexterity from humanLayer, @Lantos1618 from github-spy, and to Geoff for the inspiration.

and yeah, we forgot to get a team photo