

— A PERSONAL CASEBOOK

How I use LLMs

Stories from work, tools I've made, and
things that annoy me



LLMs entered four different parts of my life

Work

Lead engineer. Amazon Kiro, OpenAI and Claude models, CLI and IDE.

University of London

Computer Science, specialising in Machine Learning, and university projects.

Ardor

A custom cloud-based agent harness: functions, tools, refactoring, and debugging.

Personal projects

Social media, plugins, and small automations for everyday life.

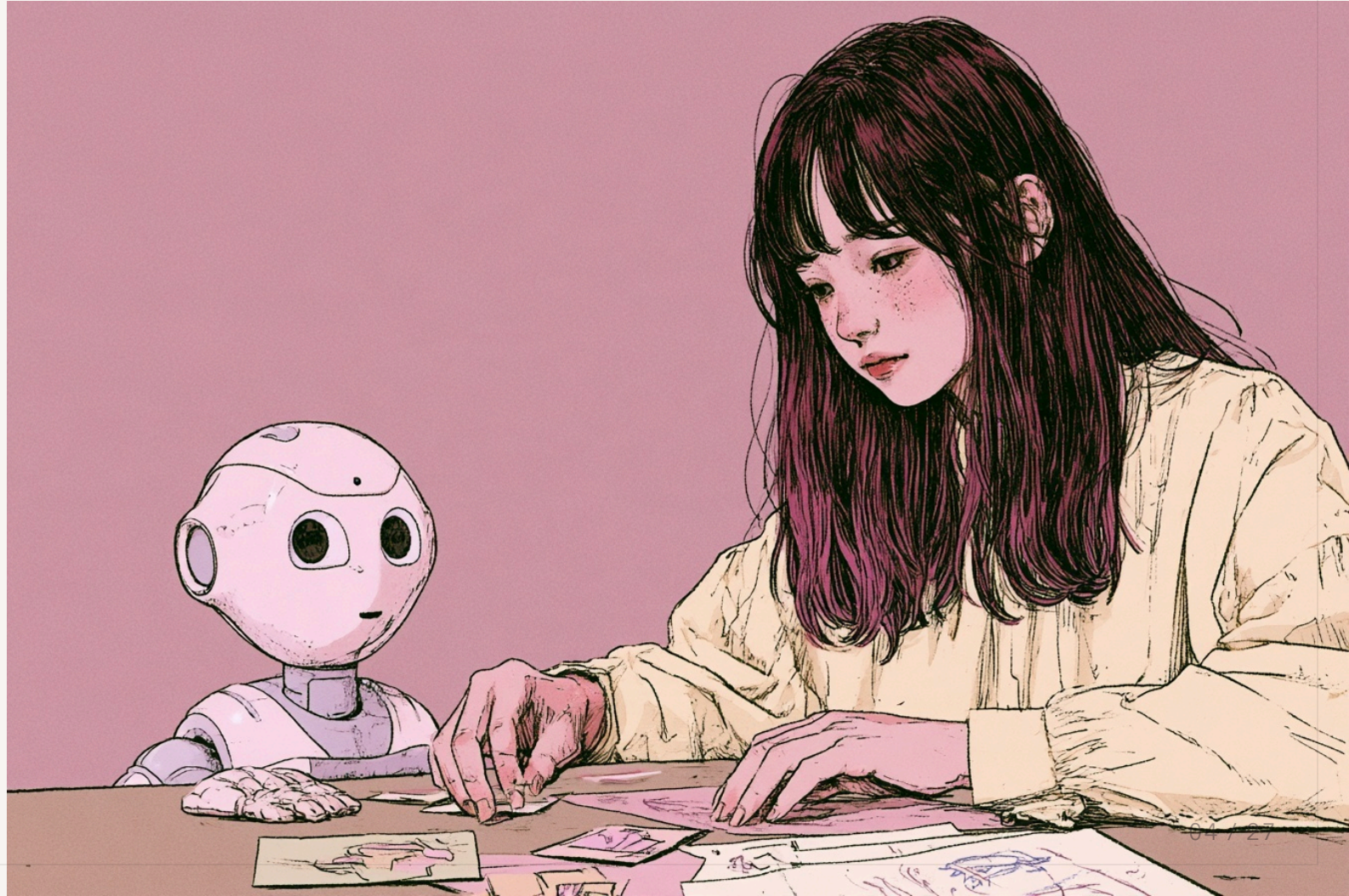
“If I need to automate something, I do it with an LLM.”

That doesn't mean I trust them with everything, or love everything they produce.

— PART 1

Work stories

Not futuristic demos, but processes that gradually became part of my everyday work.



Working with an agent starts like onboarding a new teammate

First, the basic documentation

Rules, conventions, architecture, and project quirks. As I explain them to the agent, project documentation suddenly takes shape.

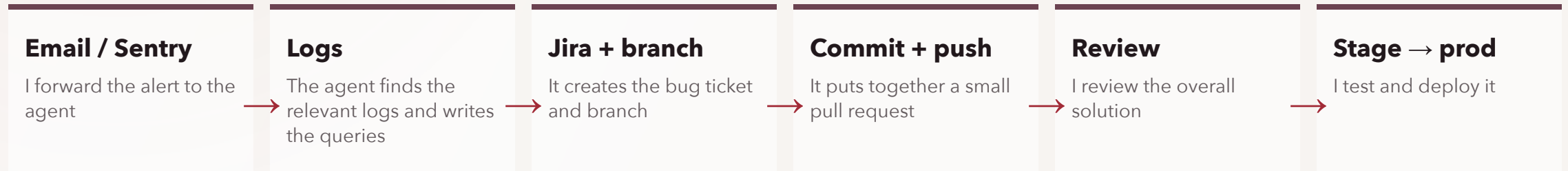
In essence, it's the same groundwork a junior developer would need on this project.

Then, access to work services

Jira · Bitbucket · Jenkins · logs

The basic skills give the agent access to the services a developer can use.

I have no qualms about giving the agent routine work



There was a time when developers felt embarrassed to even use a UI. I'm perfectly happy to ask an agent to find the logs so I don't have to write the queries myself.

“For this workflow, MCP feels too heavy to me.”

I ask the agent to write small scripts on top of the APIs.

```
skills/  
├─ jira/  
│   └─ create_bug.sh  
├─ logs/  
│   └─ search.py  
├─ bitbucket/  
│   └─ branch.sh  
└─ jenkins/  
    └─ deploy.py
```

.env – each developer uses their own tokens, so actions run under their own name.

For now, I'm still the 'human proxy'

NOW

router
+ reviewer

I pass the signal along, start the next steps, and review.



EVENTUALLY

reviewer

The workflow runs automatically, and I join to review the solution.

The current process is useful: it lets me test the workflow and find problems before automating it fully.

Every month, I need to piece together what happened in the team

Tasks

What happened with the team's tasks.

Capacity

How much capacity the team had.

Problems

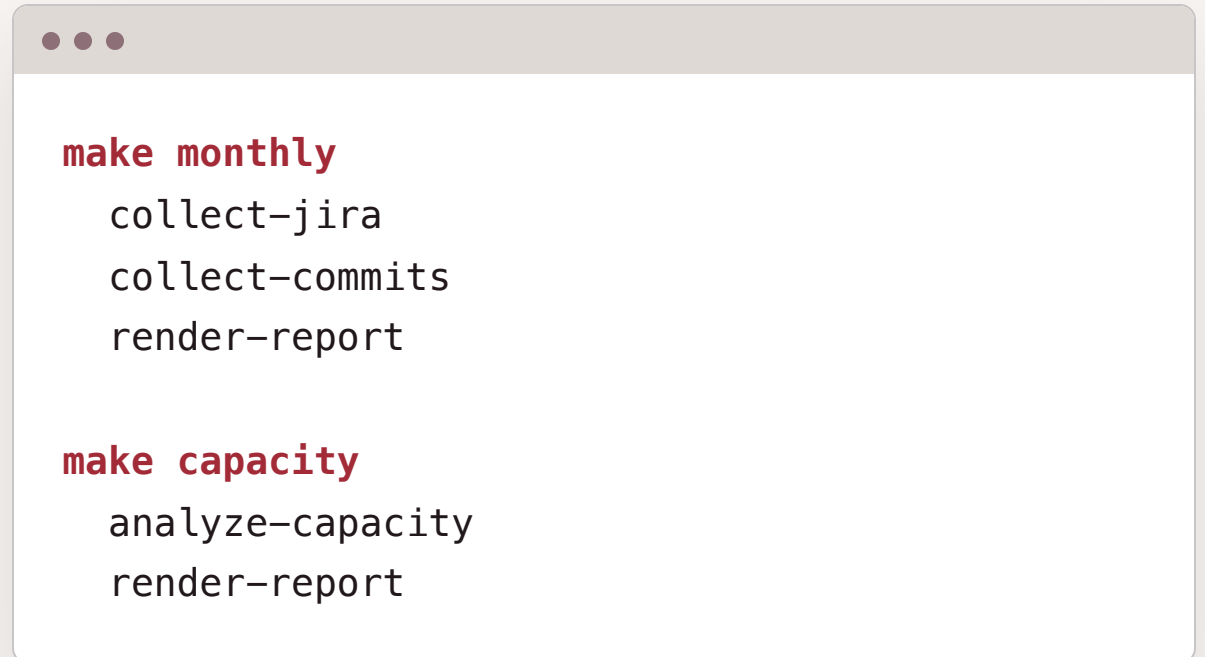
Where delays and problems appeared.

I give the agent a task the way I would brief an analyst.

The result doesn't stay in the chat

```
reports/  
├─ monthly/  
├─ capacity/  
├─ problems/  
├─ templates/  
└─ Makefile
```

Jira + commits → analysis → formatted report

A terminal window with a light gray title bar and three red window control buttons. It displays two Makefile targets: 'make monthly' and 'make capacity'. Each target has three associated commands: 'collect-jira', 'collect-commits', and 'render-report'.

```
make monthly  
  collect-jira  
  collect-commits  
  render-report  
  
make capacity  
  analyze-capacity  
  render-report
```

“Nice-looking reports, everyone's happy.”

The agent helps me think before I code

Onboarding

Complex tasks

Working across teams

I share the context and my ideas, talk through the details, and only then ask it to write the technical design.

Technical solution



Each task gets its own spec



Validation

Check the result against the original task.

Context

Keep the details across sessions.

Reuse

Hand work to another agent, or come back later.

I built a JetBrains plugin. My workflow still became a hybrid

Agent UI

The agent

- Conversation
- Doing the work
- Interactive work

PyCharm

Me and the code

- Reading and writing by hand
- Reviewing commits
- Exploring the project

github.com/mandrianova/jb-task-manager-extension ↗

“I use LLMs a lot, and
I still don't like their
code.”

When I dig into it, I often want to throw it all away and
rewrite it.



To me, the agent is a junior with excellent theory

Explain it thoroughly → show examples! → give it a bounded task

Simple tasks

Scripts

Understanding
legacy code

Low-stakes logic

Critical logic needs careful review

I review agent-written code much like a junior developer's code

Low-stakes logic

No need to read every line

When the risk is low, it's easier to run it and see whether it works.

Important logic

Read it carefully

The higher the cost of a mistake, the more closely I inspect the solution myself.

I also bring the agent into code review.

It reviews more than its own code; it can find bugs in human-written code too.

— PART 2

Tools and habits

Behind these stories are my skills, scripts, plugins, and choices about what feels comfortable to use.

At first, the agent lived in my PyCharm terminal

“Then I stopped. I find reading in a terminal uncomfortable.”

In daily work, I revisit responses, read changes, and interact with the results a lot.

```
> Analyse the changes...
```

```
First I'll read the project structure,  
then check...
```

```
[long answer continues]
```

```
scroll ↓ ↓ ↓ ↓ ↓ ↓
```

But the CLI doesn't have to be a chat



Daily UI — no.

Background tool — maybe.

The basics are converging. The directions aren't



Kiro

Built on VS Code.

Grok

Its own cloud machine,
independent of my computer.

Bionic

Local models and Kimi – cheap, but
powerful.

Codex

Interactive UI, browser, annotations,
and sessions.

This is my current map, not a product ranking.

The real difference starts beyond the chat

01

Take control

Open a browser or an app.



02

Do the work

Act inside the interface.



03

Check

Inspect the result of its own work.

“This is what still amazes me about Codex.”

I made a skill that connects Codex to a local model

Codex

Recognises when a local model might help

Skill

Finds, downloads, and runs the needed model

LM Studio

Provides a local API for LLMs, VLMs, and embeddings

Result

Returns to the main workflow as an extra check or processing step

I can't say I use this much. My real use case was making it cheaper to develop and test my own agent.

lmstudio.ai/docs/bionic ↗

A powerful computer is only one option

Locally

My own solutions, testing, and data I don't want to send to an external provider.

Through an API

Hosted open models for automation and simple tasks, without powerful hardware of my own.

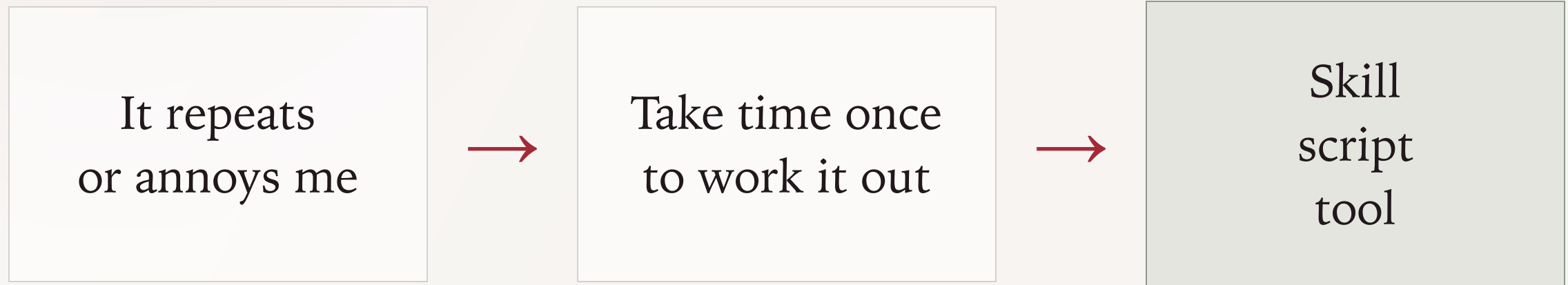
API price range · input / output per 1M tokens

gpt-oss-120b • AWS Bedrock	\$0.15 / \$0.60
DeepSeek V4.1 Flash • DeepSeek API	\$0.15–0.30 / \$0.60–1.20
GPT-5.6 Sol • OpenAI API	\$4 / \$20
Claude Fable 5.1 • Anthropic API	\$10 / \$50

List prices as of 17 Sep 2026. DeepSeek shows cache-miss pricing across the off-peak-peak range; GPT-5.6 Sol uses promotional pricing. Caching, batch, long-context and regional premiums are not shown.

[AWS Bedrock ↗](#) [DeepSeek API ↗](#) [OpenAI API ↗](#) [Anthropic ↗](#)

I turn recurring tasks into my own tools



I can adjust this way of working with each new task when something is missing.

I got tired of reading long answers in chat

Here's a detailed analysis. First I'll cover the context, then list what I found, look at possible solutions, and finally discuss the limitations...

1. First long section...
2. Another long section...
3. Yet another section...



Summary

What happened

A short explanation and context.

What matters

The main points and connections.

What to do next

Next steps in a format I can use.

The LLM builds a separate HTML page and opens it in the browser.

From the outside, it looks very simple

A new post draft

I just send it to the chat.



Skills and project rules

- my voice and style
- adapting for each platform
- image generation + a reference



Ready-to-publish material

- Telegram – Russian
- LinkedIn / X – English
- visuals in a consistent style

Project skills carry my context, so I don't have to retell it every time.

I'm not looking for one perfect way to work with LLMs

- 01 Move out of the 'human proxy' role and remain a reviewer.
- 02 Find out where background agents really work better than an interface.
- 03 Keep turning recurring annoyances into skills and tools.
- 04 Try local and cheaper open models where they make sense.

This isn't a final methodology. It's my system, and it keeps changing along with the tools.