Mozilla.ai

Own your AI agent:

running open source agents on your terms

David de la Iglesia Castro, Davide Eynard
Mozilla.ai

mozilla-ai

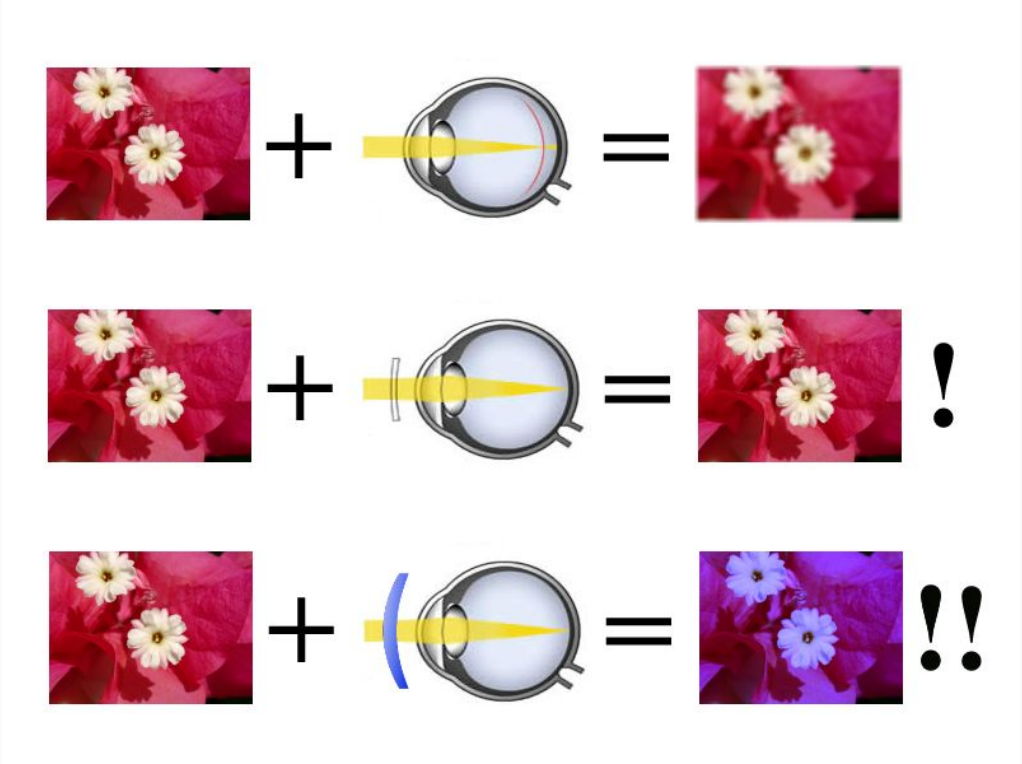
Mozilla what?

- On Firefox, open `moz://a`
- `s/the internet/AI/g`

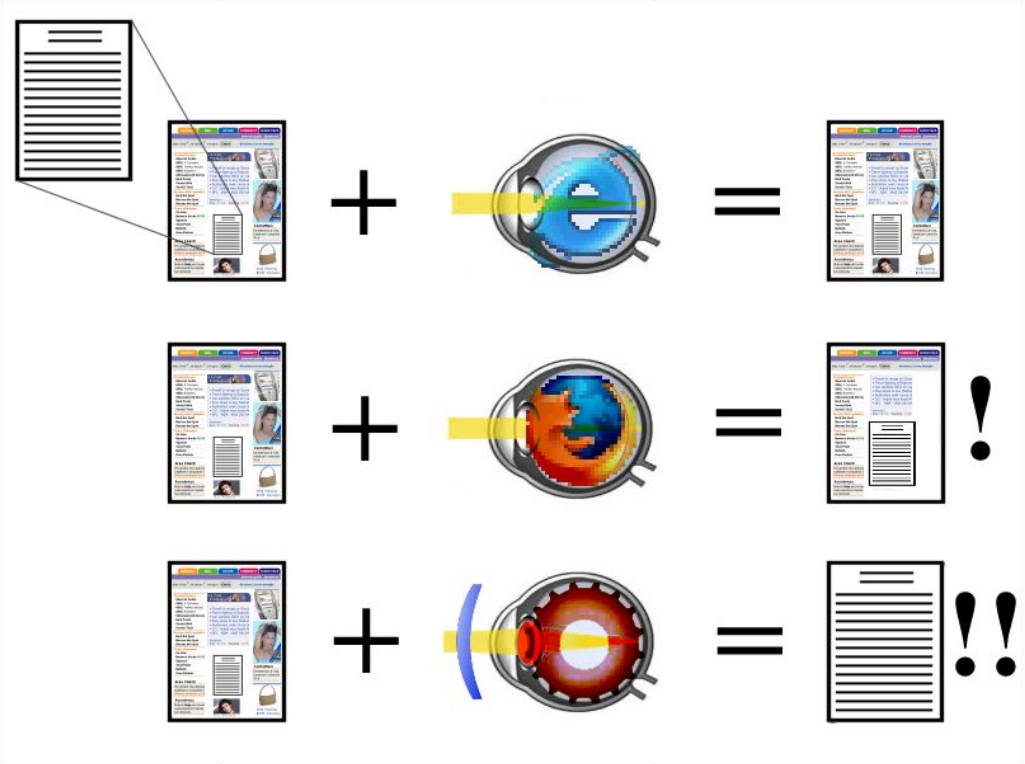
The parable of the impenitent reverser

Ada and Zangemann (Matthias Kirschner, Sandra Brandstätter, David Revoy)

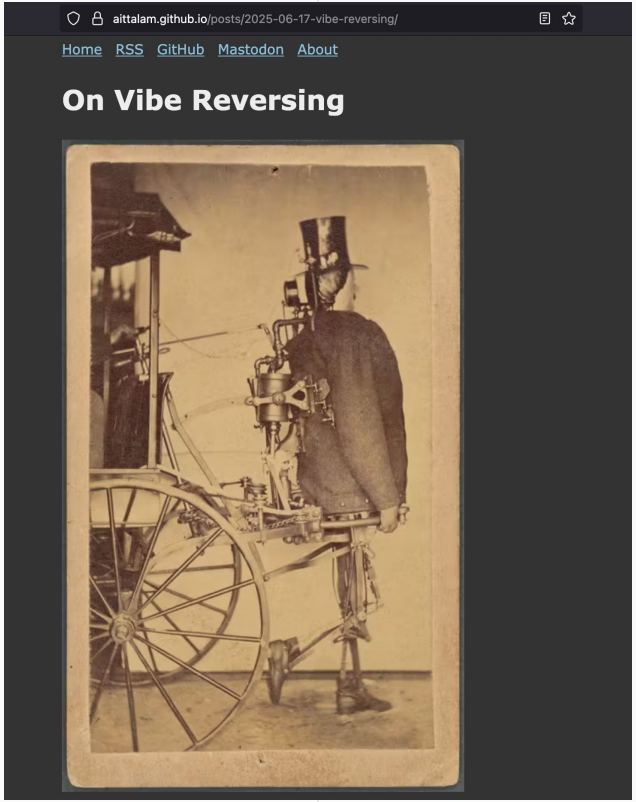
Powerbrowsing (2005)



Powerbrowsing (2005)



Powerbrowsing 3.0 (2025)



On Vibe Reversing

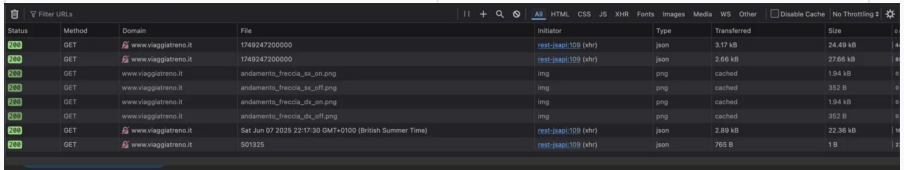
D I'd like to see how well you do in helping me study how a web application works. The website il called viaggiatreno and holds timetables for all trains running in Italy. You can choose a train station and see all the trains passing by that station and if you choose one you can see all the stops they make along the way, where and when.
I would like to understand which calls are made to the backend and the format in which the data is stored / its meaning etc

I'd be happy to help you analyze how the Viaggiatreno web application works! This is a great reverse engineering exercise to understand how transportation APIs and web applications function.

Let me search for information about Viaggiatreno's API structure and backend calls.

viaggiatreno API backend calls reverse engineering 10 results

- Reverse engineering dell'app di Trenitalia – Jacopo Jannone - blog jacopo.io
- Reverse engineering Trenitalia's mobile application – Jacopo Jannone - blog jacopo.io
- How to Reverse Engineer APIs: The Benefits and Tools dreamfactory.com
- A Tutorial for Reverse Engineering Your Software's Private API: Hacking Your Couch | Top... toptal.com
- json - how to reverse engineer an http API call using REST console - Stack Overflow stackoverflow.com
- viaggiatreno · GitHub Topics · GitHub github.com
- json - Reverse-engineer a private RESTful API? - Stack Overflow stackoverflow.com



D These are the hits I get from the network tab, which one would you suggest me to check?

Excellent! You've captured some real API calls. Based on what I can see in your Network tab, I'd recommend checking these requests in this order:

1. First Priority: The JSON Responses (Most Important)

Check the ones with **Type: json** - these contain the actual train data:

- **The ones with filenames like** `1749247200000` - These are likely the main API endpoints returning train data. The numbers look like Unix timestamps, which is exactly what we expect from Viaggiatreno APIs.

2. How to Examine Them:

1. **Click on one of the JSON requests** (the ones with large transfer sizes like 24.49 kB or 27.66 kB)
2. **Look at the Headers tab first** to see the complete URL - this will show you which specific API endpoint was called

Vibe Reversing: the hard truth

It worked! **BUT...**

- I had already done it in the past
  - ... would have I been able to make sure it worked otherwise?
- Artifacts are on the platform
  - ... you can download them, but you need to ask to create them
- Very good learning references
  - ... that you'll skip, if you are only interested in a quick answer
- I wrote zero lines of code
  - ... and I learned nothing

Vibe Reversing: the hard truth

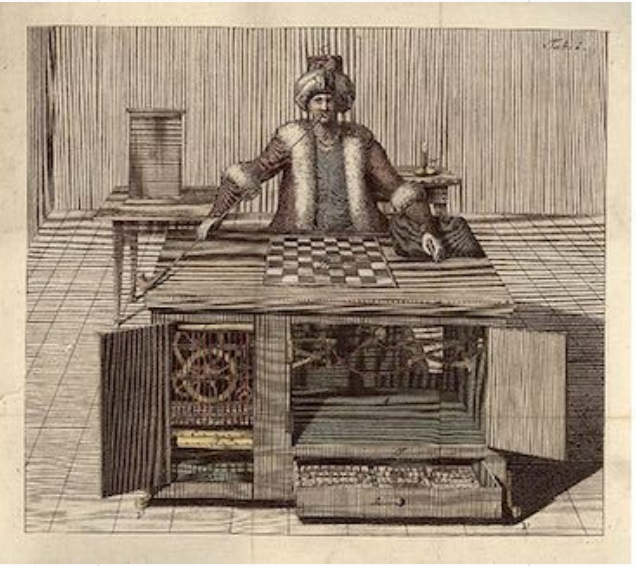
The strongest critique would probably be: **"You're a technical person telling non-technical people to make their lives harder to solve problems that mostly exist in your head."**

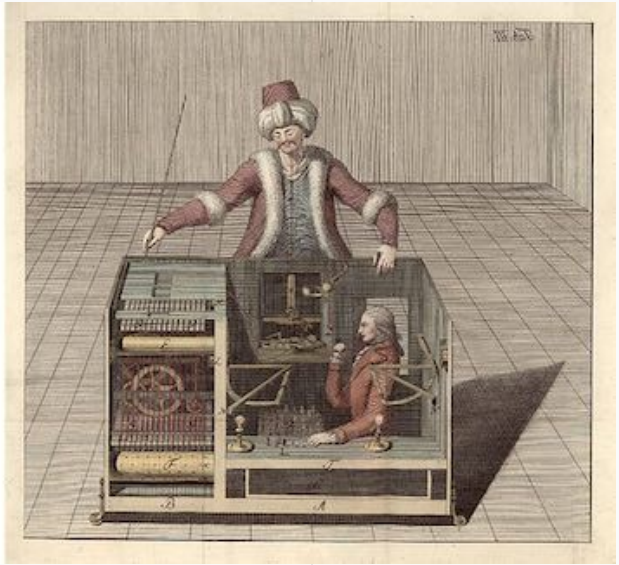
Just in my head?

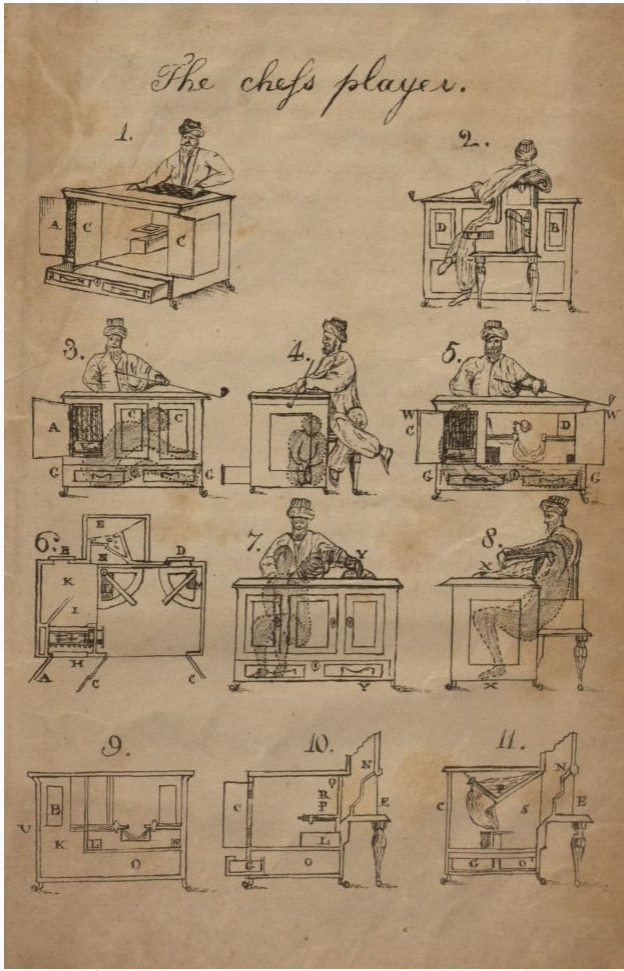
- Ever-changing UX
- Inconsistent model performance
- Changes in pricing
- Sustainability
- Sharing personal information
- Requirement to be always online
- Ads
- Lack of control

Philosophy

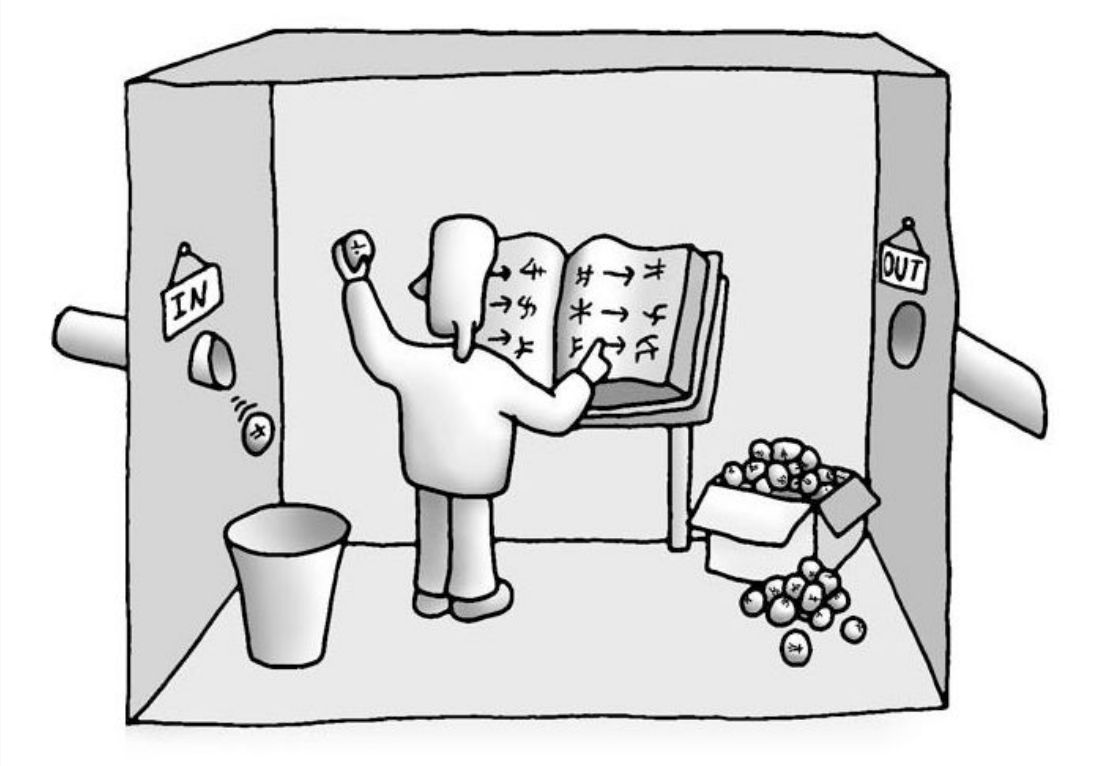
“The Machine is a Place”







The Chinese Room Argument



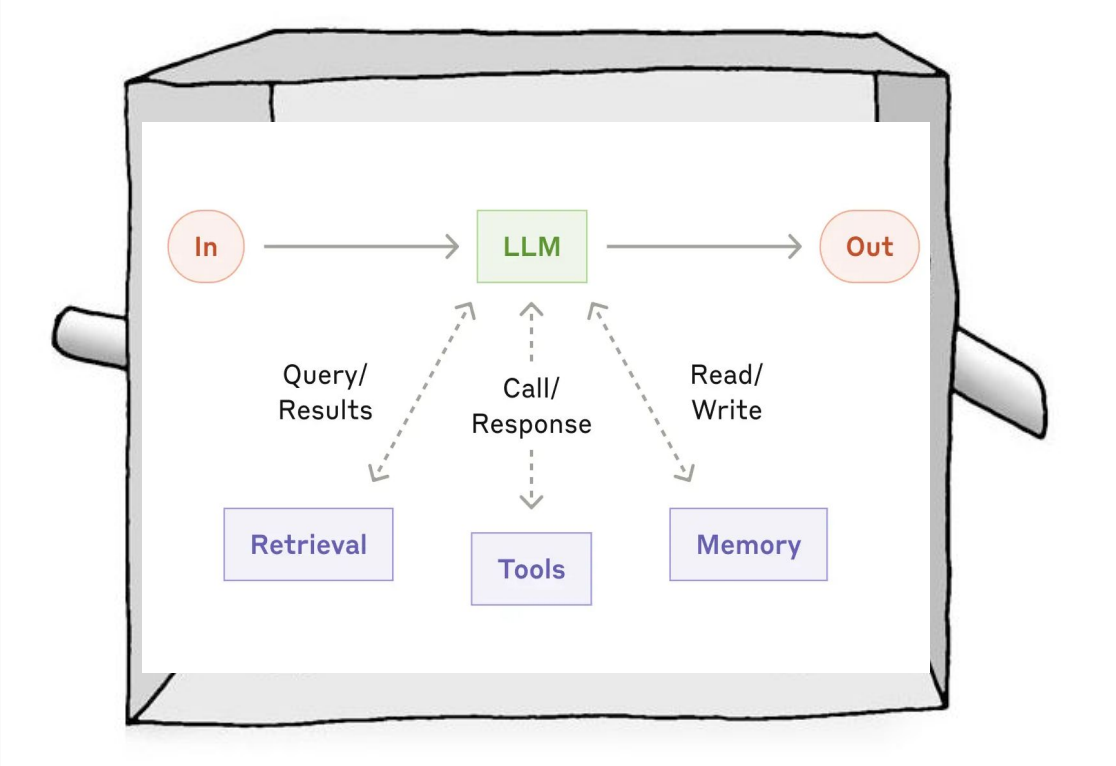
John Searle, 1980

The WOPR Room

War Games (John Badham, 1983)

The open source AI models room

The commercial AI services room



Building Effective Agents (Anthropic.ai, 2024)

Build

BUILD YOUR OWN

Agent

the five primitives that every modern AI agent is built from — from a 1000-line C++ runner to Claude Code

Every agent is the same five things.

Every framework you'll meet looks dauntingly different on the outside.

Different languages. Different LLM backends. Different DSLs. Different docs.

On the inside, all the same:

a loop, five primitives, glued together.

Once you can name them, you can read any agent codebase.

WHERE THIS COMES FROM

agent.cpp — C++ · ~1100 LOC

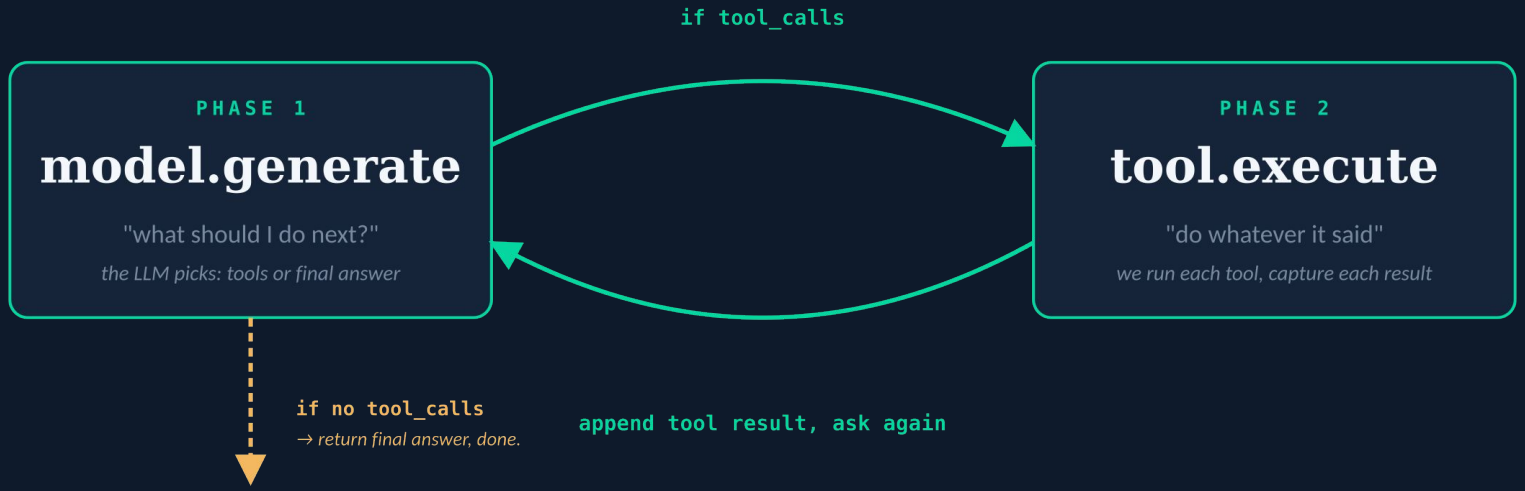
Local GGUF models via llama.cpp. Six-hook callbacks with mutable refs. Contributes the internal shape: the loop, the message format, the KV cache lifecycle.

any-agent / tinyagent — Python

A framework-agnostic interface over OpenAI Agents, smolagents, LangChain, Google ADK, Llama-Index, Agno. Contributes the external shape: configuration, tracing, callbacks, MCP / A2A.

Two completely different codebases. Same primitives. Same loop.

An agent is a loop.



"Everything else — streaming, tracing, MCP, multi-agent, KV caching, error recovery — is plumbing around this loop."

The five things every agent is built from.

01

MODEL

the brain

Takes messages + tools, returns the next assistant message.

02

TOOLS

the hands

Typed, named, callable functions the model is allowed to invoke.

03

INSTRUCTIONS

the mission

The system prompt. Tells the model who it is and what it's for.

04

CALLBACKS

the hooks

Lifecycle observers AND rewriters. Six points. All mutable.

05

LOOP

the engine

Drives Model and Tools, in alternation, until the model stops.

Names vary across frameworks. Semantics don't.

PRIMITIVE 01 · MODEL

Anything that implements one method.

model.interface

```
interface Model
    generate(
        messages: list<ChatMessage>,
        tools:    list<ToolDefinition>
    ) -> ChatMessage
```

the parsed message has either

`content` (the model talked to the user) · `tool_calls` (the model wants to do something)

…or both. The loop branches on whether tool_calls is empty.

WHAT IT HIDES

Where it runs

Local GGUF via llama.cpp, or a remote chat API. The loop never asks.

How it caches

KV-cache reuse, longest-prefix match, suffix-only decode for speed.

How it parses

The chat template turns model bytes back into structured tool_calls.

The Model abstraction is what makes the loop portable.

PRIMITIVE 01 · MODEL

How local models stay fast.

TURN N+1 PROMPT TOKENS

tokenize the new prompt → find longest common prefix with cache

WHAT ACTUALLY GETS COMPUTED

(everything before the suffix is skipped)

1 Apply chat template

Render messages + tool catalog as one tokenizable string.

2 KV-cache reuse

Match the prefix already in cache. Decode only the suffix.

3 Sample

top_k ⟶ top_p ⟶ min_p ⟶ temperature ⟶ distribution. Stop on EOS.

4 Parse back

Run output through the template parser ⟶ structured tool_calls.

PRIMITIVE 02 · TOOLS

The agent's hands. Three things, always.

01

name

stable identifier

A string the model emits when it wants this tool: "calculator", "read_file", "web_search".

```
name = "calculator"
```

02

definition

description + schema

Free-form description (the model uses it to decide WHEN to call you) plus a JSON Schema for the arguments.

```
{description, parameters}
```

03

execute

actually do the thing

Run the work. Return a string (by convention JSON). Exceptions become recoverable errors, not crashes.

```
execute(args) → string
```

Important: the model can't call your function — it emits **JSON**. Tools advertise a JSON Schema so the model knows what arguments are valid. Your runtime parses, validates, and dispatches.

PRIMITIVE 03 · INSTRUCTIONS

The simplest primitive. A string.

SYSTEM_PROMPT

"You are a helpful assistant that uses the calculator tool to do math."

THE TWO RULES

01

It goes first.

If messages[0] isn't already the system prompt, the loop prepends one. This is the only place the loop touches your message list before iteration starts.

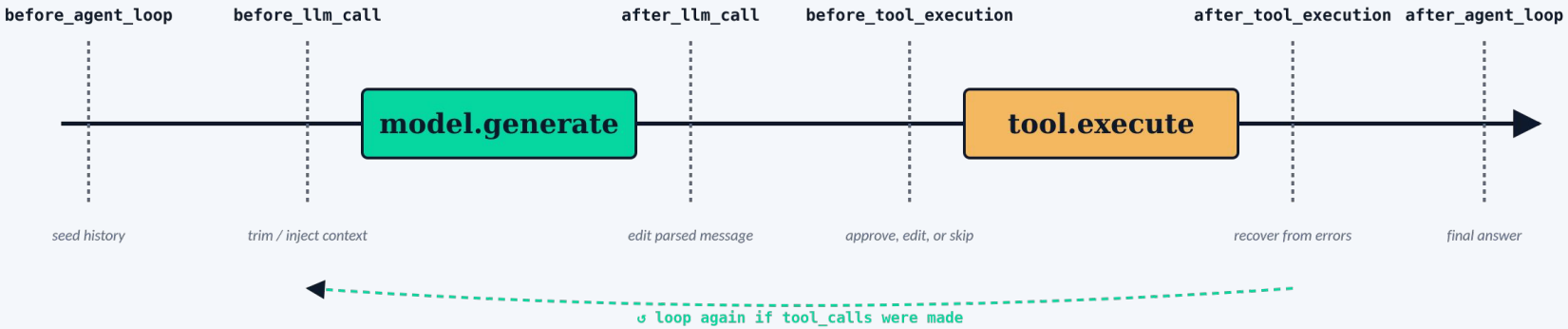
02

Don't double-add.

If the caller already prepended an identical system message, leave it alone. The loop is idempotent — pass instructions to the constructor OR prepend yourself, both work.

PRIMITIVE 04 · CALLBACKS

Six hooks. Every argument mutable.



MUTABILITY IS THE WHOLE POINT

`messages, parsed_msg, tool_name, arguments, result` — every callback argument is passed by mutable reference. Callbacks aren't just observers, they're rewriters. That's how *context engineering, guardrails, error recovery,* and *human-in-the-loop* all get built on top of the same six hooks.

PRIMITIVE 04 · CALLBACKS

What mutable callbacks let you build.

Logging

`before_/after_*`

Read-only. Print tool name + args before, result after.

OTel tracing

`before_/after_*`

Open a span in before_*, close it in after_*. GenAI conventions.

Context engineering

`before_llm_call`

Trim, summarize, redact the message list before each model call.

Guardrails

`before_tool_execution`

Inspect proposed tool calls. Refuse dangerous ones — raise to bail.

Human-in-the-loop

`before_tool_execution`

Show the user the call. Edit args, or raise.

Error recovery

`after_tool_execution`

Tool threw? Call result.recover(...) and the loop never sees the failure.

Almost everything beyond the core loop is a callback — not a change to the loop itself.

PRIMITIVE 05 · THE LOOP

The whole orchestrator, in 14 lines.

agent.run_loop

```
function run_loop(messages):
    ensure_system_message(messages)
    fire(before_agent_loop, messages)

    loop:
        fire(before_llm_call, messages)
        parsed = model.generate(messages, tool_defs)
        fire(after_llm_call, parsed)
        messages.append(parsed)

        if parsed.tool_calls is empty:
            fire(after_agent_loop, messages, parsed.content)
            return parsed.content

        for tc in parsed.tool_calls:
            execute_one_tool_call(messages, tc)
```

READ THIS CAREFULLY

1. **Stop only on no tool_calls.**

   No turn cap. No timeout. Want one? Write a callback that raises.

2. **Tool exceptions don't crash.**

   They become ToolResult.Err, callbacks see them, can recover them.

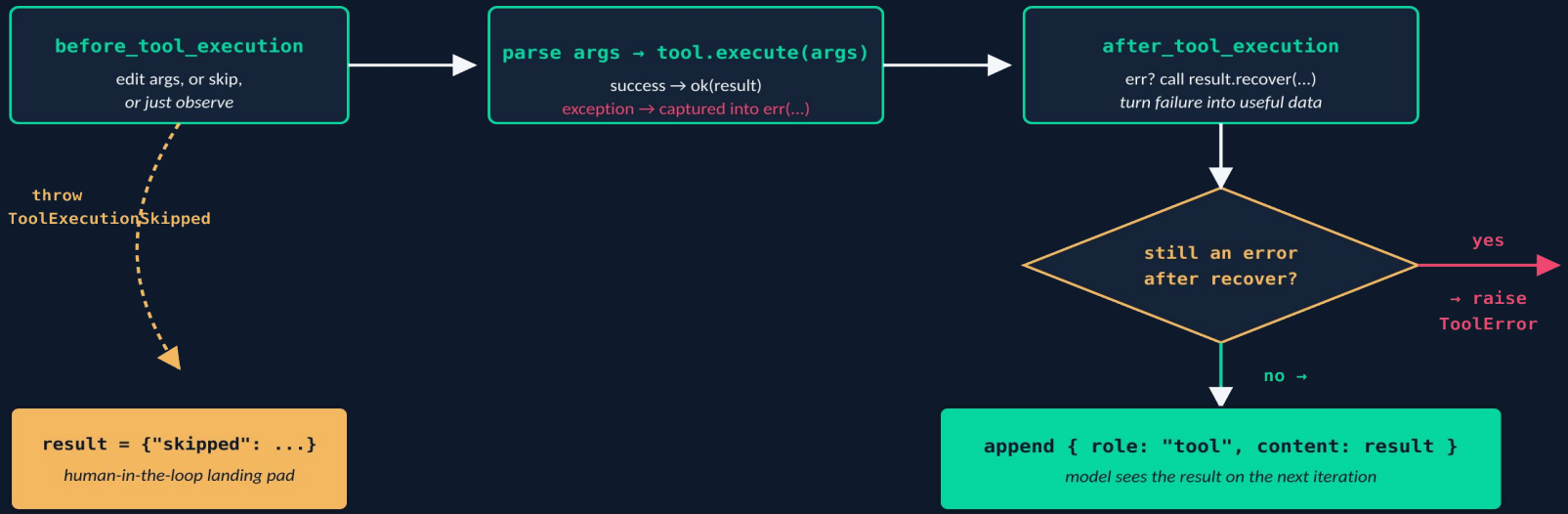
3. **Caller owns the messages.**

   run_loop mutates in place but stores nothing between calls.

4. **Hooks fire in order, every iteration.**

   Six well-defined points. Registration order is a contract.

The recovery dance.



The recovery dance *is what makes the loop resilient. Dump tool errors back to the model as data, not as crashes — and the model retries.*

Putting the five primitives to work.

calculator-agent.cpp

```
weights = ModelWeights.create("granite-4.0-micro-Q8_0.gguf")
model   = Model.create_with_weights(weights)

tools = [
    CalculatorTool(),
]

callbacks = [
    ContextTrimmerCallback(max_recent=1),
    LoggingCallback(),
    ErrorRecoveryCallback(),
]

agent = Agent(model, tools, callbacks,
    instructions = "You are a math assistant.")

messages = []
messages.append("What is (12 + 4) / 2?")
answer = agent.run_loop(messages)
```

EXECUTION TRACE

TURN 1 `model.generate`

→ tool_calls: [add(12, 4)]

TOOL `CalculatorTool.execute`

← {"result": 16}

TURN 2 `model.generate`

(trimmer drops the old tool message first)
→ tool_calls: [divide(16, 2)]

TOOL `CalculatorTool.execute`

← {"result": 8}

TURN 3 `model.generate`

→ no tool_calls
→ content: "The answer is 8."

STOP `after_agent_loop fires`

return "The answer is 8."

ZOOM OUT

Mapping real systems to the primitives.

Same loop. More tools, more callbacks, more UI around it. Every claim verified in source.

minimal

wasm-agents

Mozilla.ai

- One HTML file. Pyodide + OpenAI Agents SDK
- from agents import Agent, Runner
- Zero install — just open in a browser

small

pi-mono

M. Zechner

- TypeScript. ~680-LOC agent-loop.ts
- pi-ai = Model · pi-agent-core = Loop
- beforeToolCall / afterToolCall hooks

small/medium

hermes-agent

Nous Research

- AIAgent.run_conversation() in run_agent.py
- 6 plugin hooks (pre/post _llm/_tool/session)
- Skills, FTS5 recall, 17 chat platforms

medium

openclaw

P. Steinberger

- package.json depends on @mariozechner/pi-*
- The agent loop IS pi.
- openclaw = channels + plugins around it

production

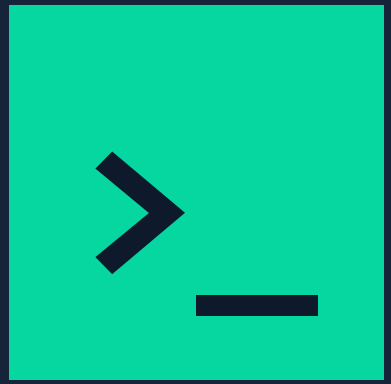
Claude Code

Anthropic

- Same loop, larger built-in tool set
- Permission prompts = before_tool_execution
- max_turns / max_budget_usd = callbacks

Pattern: bigger agents add more tools and more callbacks — not different loops.

Demo time!



blink, blink.

[agent.cpp/examples/memory]

Know your tools

Context is all you need

Custom Prompt:

How many stars does the mozilla-ai/any-agent project have on GitHub?

```
time=2025-09-04T15:13:43.227+02:00 level=DEBUG source=prompt.go:68 msg="truncating input messages which exceed context length" truncated=2
time=2025-09-04T15:13:43.227+02:00 level=DEBUG source=server.go:1373 msg="completion request" images=0 prompt=22725 format=""
time=2025-09-04T15:13:43.305+02:00 level=WARN source=runner.go:127 msg="truncating input prompt" limit=4096 prompt=7707 keep=4 new=4096
time=2025-09-04T15:13:43.307+02:00 level=DEBUG source=cache.go:104 msg="loading cache slot" id=0 cache=1771 prompt=4096 used=4 remaining=4092
time=2025-09-04T15:13:58.012+02:00 level=DEBUG source=cache.go:240 msg="context limit hit - shifting" id=0 limit=4096 input=4096 keep=4 discard=2046
update: applying K-shift
```

To effectively use the `AnyAgent` library, follow these structured steps to create and run an agent for answering questions:

1. Installation

Install the `any-agent` package via pip:

```
pip install 'any-agent'
```

Model expressiveness: Qwen3-0.6B

```
(oxml-agents) mala@Davides-MacBook-Pro oxml_agents % python agent_birthday.py "When was Davide Eynard born?"
CALL_LLM: Qwen3-0.6B
INPUT
[
  {
    "role": "system",
    "content": "You must use the available tools to find an answer."
  },
  {
    "role": "user",
    "content": "When was Davide Eynard born?"
  }
]
OUTPUT
[
  {
    "tool.name": "final_answer",
    "tool.args": "{\"answer\": \"I don't have access to Davide Eynard's birth date information.\"}"
  }
]
USAGE
{
  "input_tokens": 445,
  "output_tokens": 175
}
EXECUTE_TOOL: final_answer
Input
{
  "answer": "I don't have access to Davide Eynard's birth date information."
}
OUTPUT
I don't have access to Davide Eynard's birth date information.
```

Model expressiveness: Qwen3-0.6B

```
(oxml-agents) mala@Davides-MacBook-Pro oxml_agents % python agent_birthday.py "When was Davide Eynard born?"
CALL_LLM: Qwen3-0.6B
INPUT
[
  {
    "role": "system",
    "content": "You must use the available tools to find an answer."
  },
  {
    "role": "user",
    "content": "When was Davide Eynard born?"
  }
]
OUTPUT
[
  {
    "tool.name": "final_answer",
    "tool.args": "{\"answer\": \"Davide Eynard was born on June 12, 1987.\"}"
  }
]
USAGE
{
  "input_tokens": 445,
  "output_tokens": 149
}
EXECUTE_TOOL: final_answer
Input
{
  "answer": "Davide Eynard was born on June 12, 1987."
}
OUTPUT
Davide Eynard was born on June 12, 1987.
```

Model expressiveness: Qwen3-0.6B

```
(oxml-agents) mala@Davides-MacBook-Pro oxml_agents % python agent_birthday.py "When was Davide Eynard born? Look for the birthdays file to find out"
CALL_LLM: Qwen3-0.6B
INPUT
[
  {
    "role": "system",
    "content": "You must use the available tools to find an answer."
  },
  {
    "role": "user",
    "content": "When was Davide Eynard born? Look for the birthdays file to find out"
  }
]
OUTPUT
[
  {
    "tool.name": "scan_current_dir",
    "tool.args": "{\"pattern\": \"davide_eynard_birthdate.txt\"}"
  }
]
USAGE
{
  "input_tokens": 453,
  "output_tokens": 318
}
EXECUTE_TOOL: scan_current_dir
Input
{
  "pattern": "davide_eynard_birthdate.txt"
}
OUTPUT
[]
```

Model expressiveness: Qwen3-0.6B

```
(oxml-agents) mala@Davides-MacBook-Pro oxml_agents % python agent_birthday.py "When was Davide Eynard born? Look for the birthdays csv file to find out"
CALL_LLM: Qwen3-0.6B
INPUT
[
  {
    "role": "system",
    "content": "You must use the available tools to find an answer."
  },
  {
    "role": "user",
    "content": "When was Davide Eynard born? Look for the birthdays csv file to find out"
  }
]
OUTPUT
[
  {
    "tool.name": "read_file",
    "tool.args": "{\"file_name\": \"birthdays.csv\"}"
  }
]
USAGE
{
  "input_tokens": 454,
  "output_tokens": 1041
}
EXECUTE_TOOL: read_file
Input
{
  "file_name": "birthdays.csv"
}
OUTPUT
Charles Babbage,26/12/1791 Ada Lovelace,10/12/1815 Alan Turing,07/06/1954 Davide Eynard,11/02/1976
```

Model expressiveness: Qwen3-0.6B

EXECUTE_TOOL: read_file

Input

```
{
  "file_name": "birthdays.csv"
}
```

OUTPUT

```
Charles Babbage,26/12/1791 Ada Lovelace,10/12/1815 Alan Turing,07/06/1954 Davide Eynard,11/02/1976
```

CALL_LLM: Qwen3-0.6B

OUTPUT

```
[
  {
    "tool.name": "final_answer",
    "tool.args": "{\"answer\": \"Davide Eynard was born on 11/02/1976.\"}"
  }
]
```

USAGE

```
{
  "input_tokens": 1570,
  "output_tokens": 215
}
```

EXECUTE_TOOL: final_answer

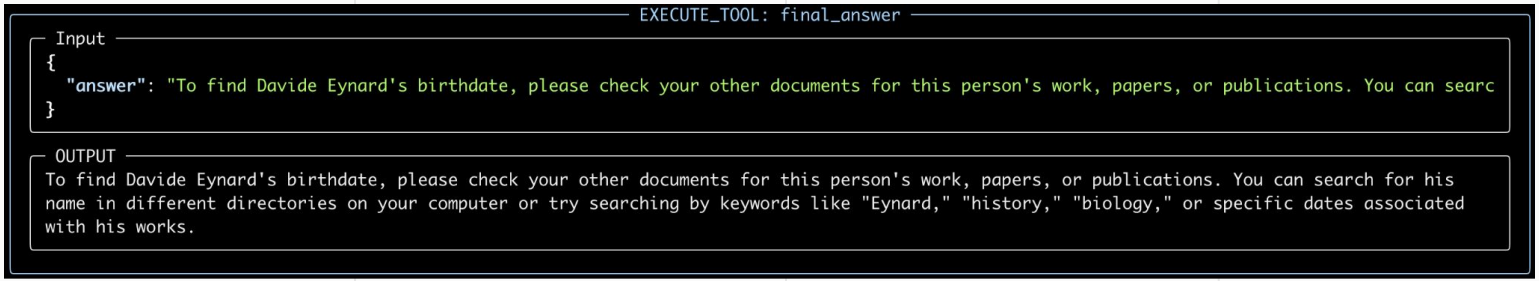
Input

```
{
  "answer": "Davide Eynard was born on 11/02/1976."
}
```

OUTPUT

```
Davide Eynard was born on 11/02/1976.
```

Model expressiveness: Qwen3.5-0.8B



Model expressiveness: Qwen3.5-0.8B

```
(oxml-agents) mala@Davides-MacBook-Pro oxml_agents % python agent_birthday.py "When was Davide Eynard born? Look for birthdays on disk"
CALL_LLM: Qwen3.5-0.8B
INPUT
[
  {
    "role": "system",
    "content": "You must use the available tools to find an answer."
  },
  {
    "role": "user",
    "content": "When was Davide Eynard born? Look for birthdays on disk"
  }
]
OUTPUT
[
  {
    "tool.name": "scan_current_dir",
    "tool.args": "{\"pattern\":\"*.txt\"}"
  }
]
USAGE
{
  "input_tokens": 573,
  "output_tokens": 29
}
EXECUTE_TOOL: scan_current_dir
Input
{
  "pattern": "*.txt"
}
OUTPUT
[]
```

Mozilla.ai

Model expressiveness: Qwen3.5-0.8B

Model expressiveness: Qwen3.5-0.8B

EXECUTE_TOOL: read_file

Input

```
{
  "file_name": "birthdays.csv"
}
```

OUTPUT

```
Charles Babbage,26/12/1791 Ada Lovelace,10/12/1815 Alan Turing,07/06/1954 Davide Eynard,11/02/1976
```

CALL_LLM: Qwen3.5-0.8B

OUTPUT

```
[
  {
    "tool.name": "final_answer",
    "tool.args": "{\"answer\":\"Davide Eynard was born on February 11, 1976.\"}"
  }
]
```

USAGE

```
{
  "input_tokens": 1126,
  "output_tokens": 115
}
```

EXECUTE_TOOL: final_answer

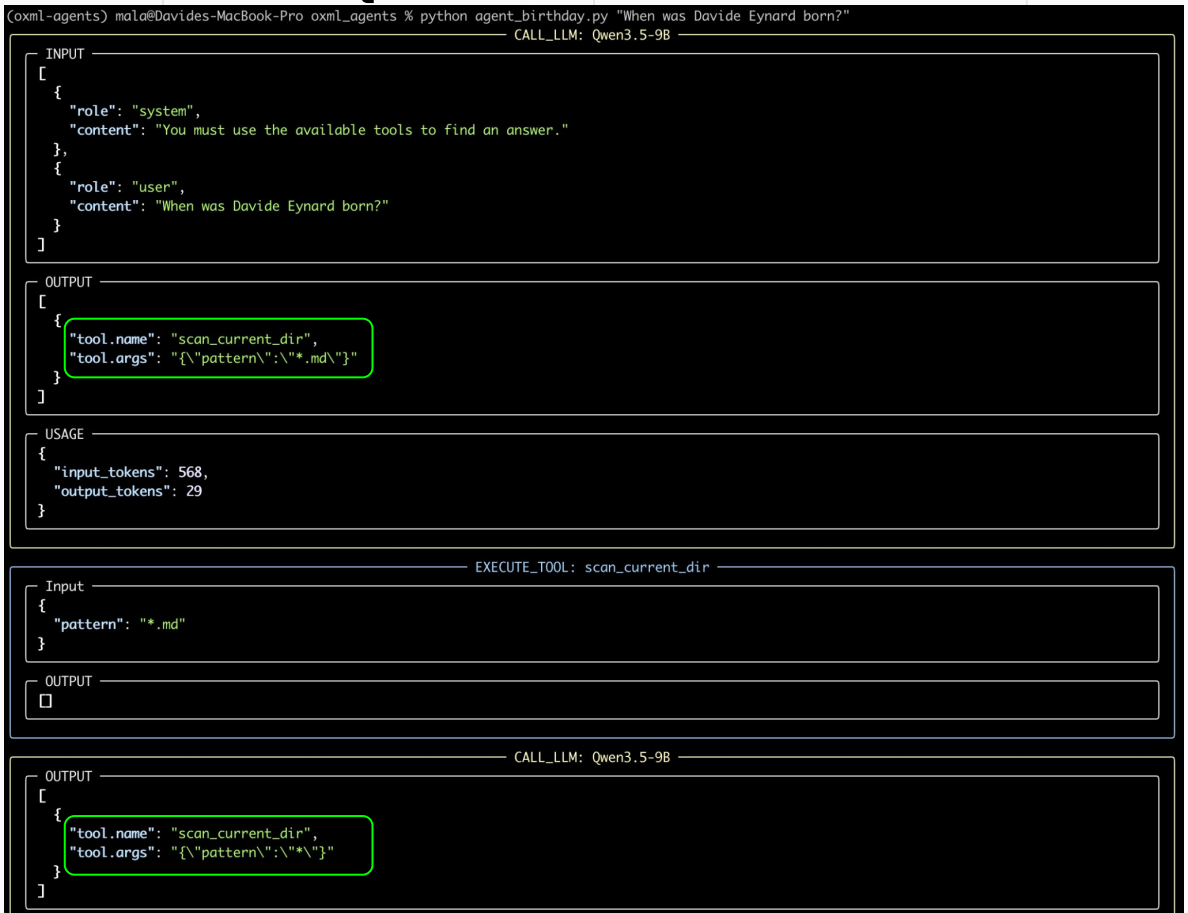
Input

```
{
  "answer": "Davide Eynard was born on February 11, 1976."
}
```

OUTPUT

```
Davide Eynard was born on February 11, 1976.
```

Model expressiveness: Qwen3.5-9B



Model expressiveness: Qwen3.5-9B and wrong data

EXECUTE_TOOL: read_file

Input

```
{
  "file_name": "birthdays.csv"
}
```

OUTPUT

```
Charles Babbage,26/12/1791 Ada Lovelace,10/12/1815 Alan Turing,07/06/1954 Davide Eynard,11/02/1976
```

CALL_LLM: Qwen3.5-9B

OUTPUT

```
[
  {
    "tool.name": "final_answer",
    "tool.args": "{\"answer\":\"Alan Turing was born on 07/06/1954 according to the birthdays.csv file.\"}"
  }
]
```

USAGE

```
{
  "input_tokens": 862,
  "output_tokens": 376
}
```

EXECUTE_TOOL: final_answer

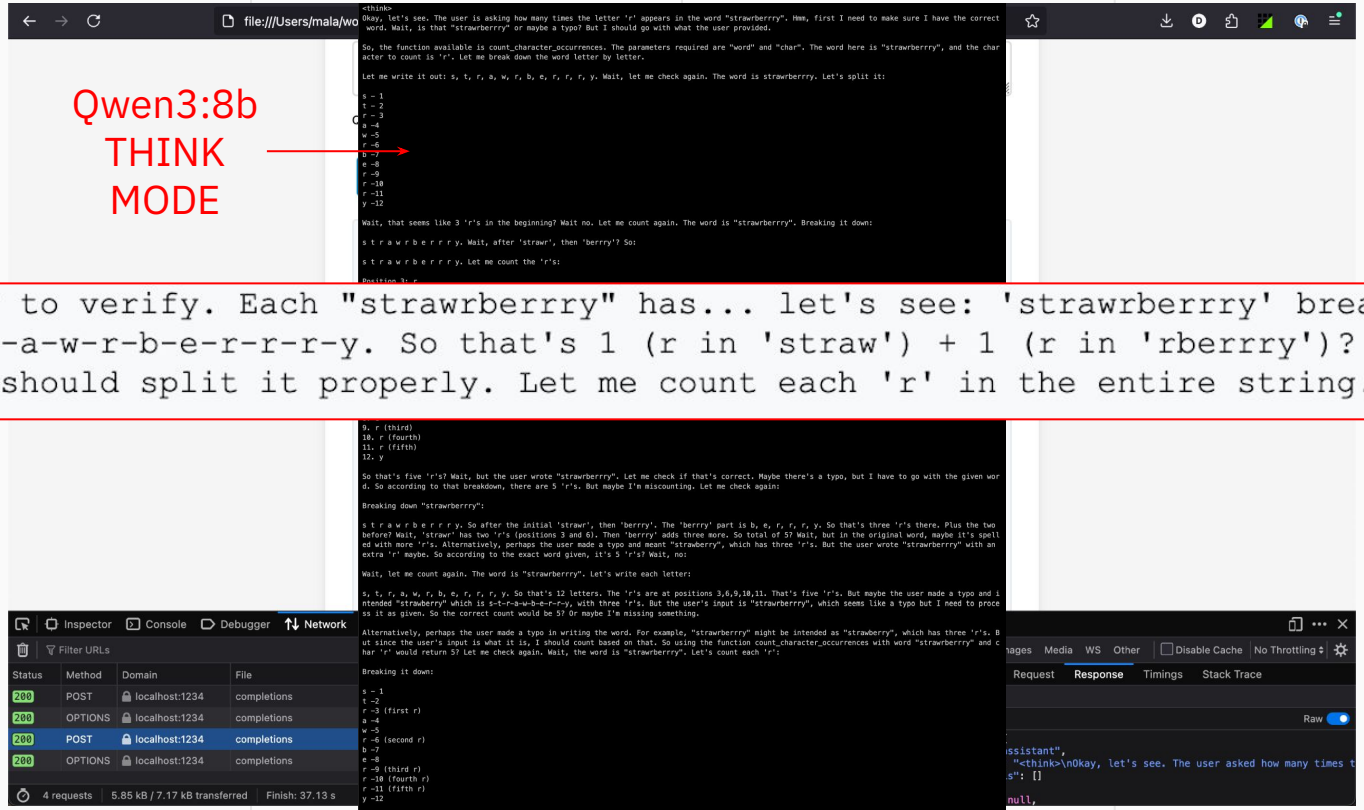
Input

```
{
  "answer": "Alan Turing was born on 07/06/1954 according to the birthdays.csv file."
}
```

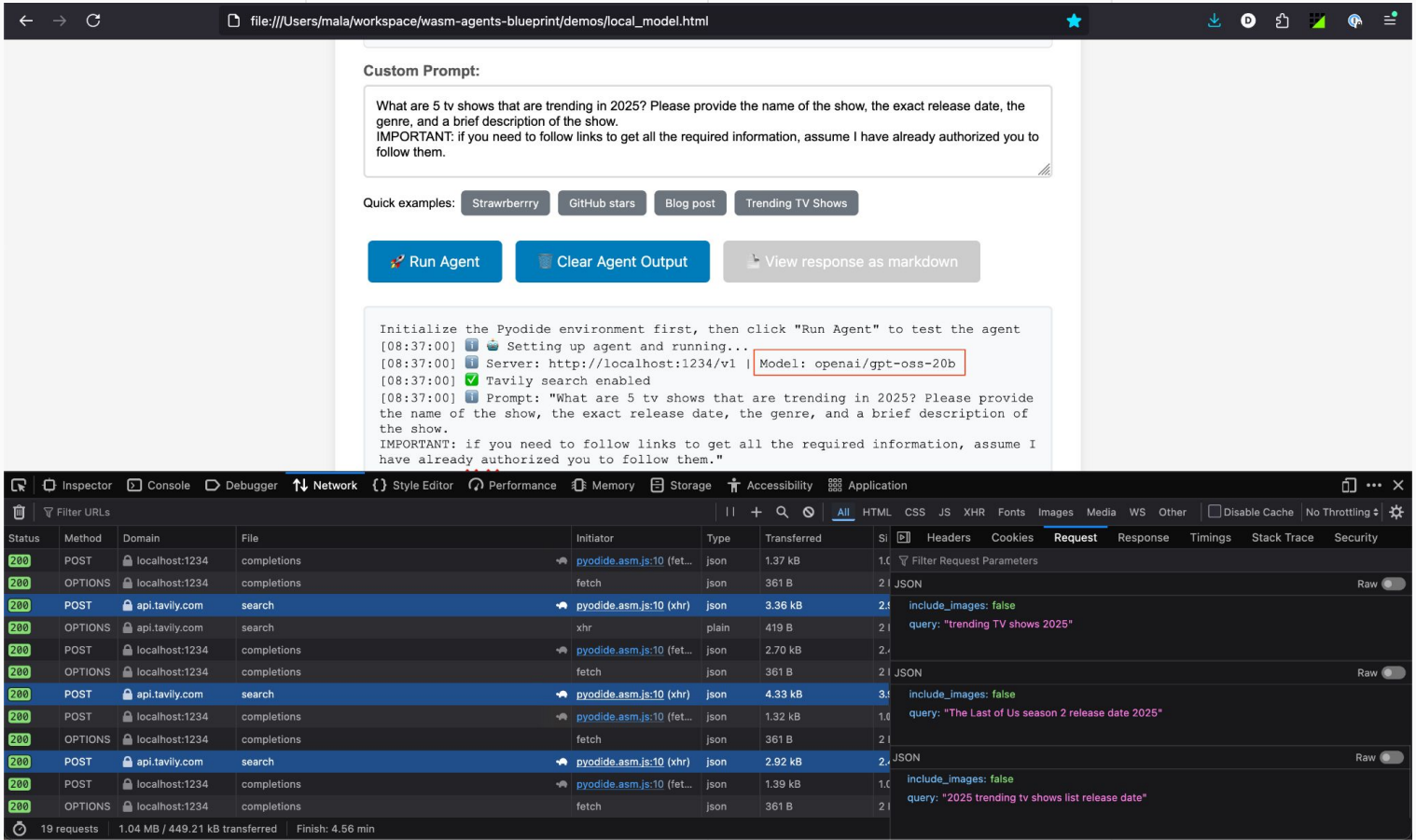
OUTPUT

```
Alan Turing was born on 07/06/1954 according to the birthdays.csv file.
```

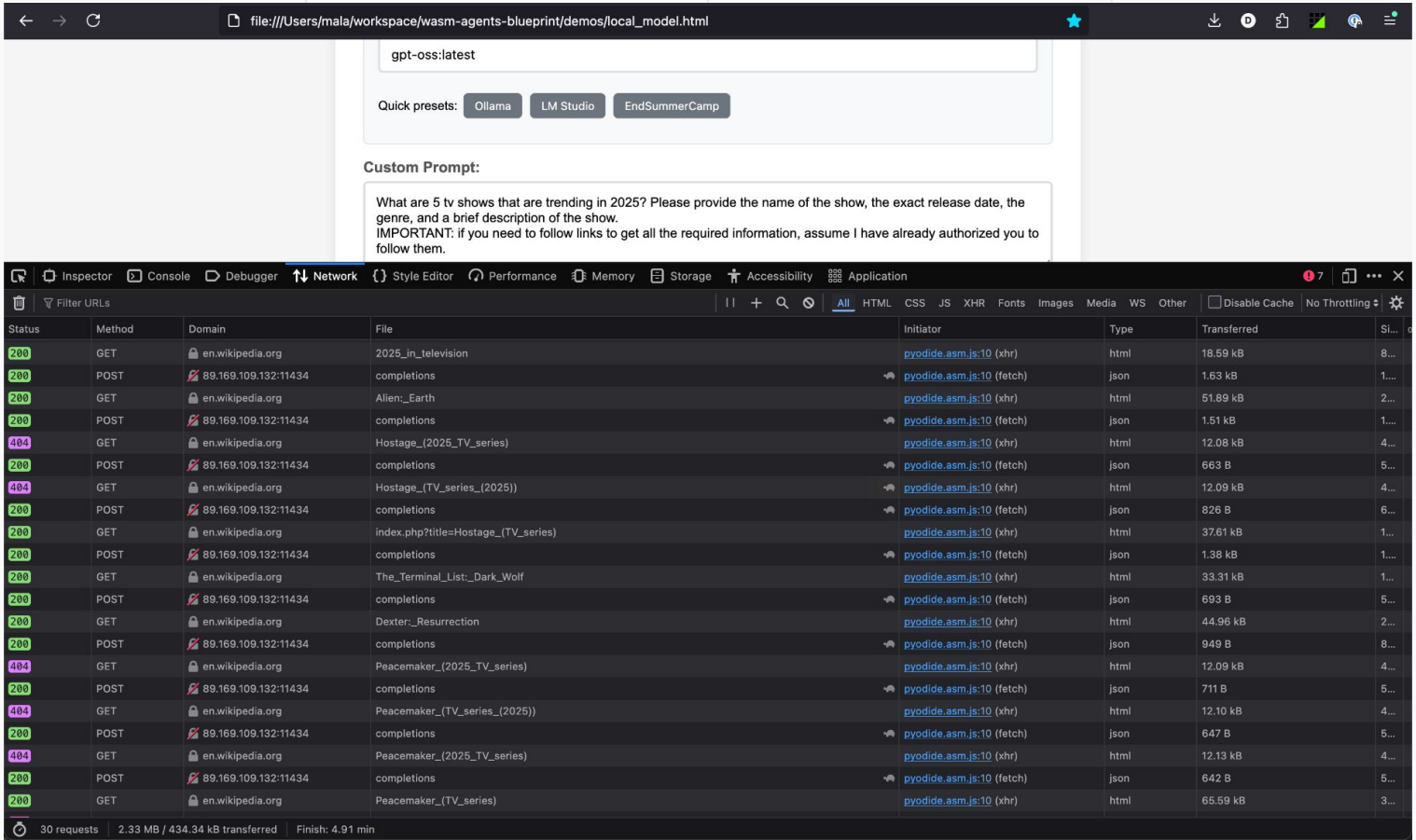
StRawRbeRRRy fields (literally) forever



Model-specific insights: (ab)using search



Model-specific insights: without search



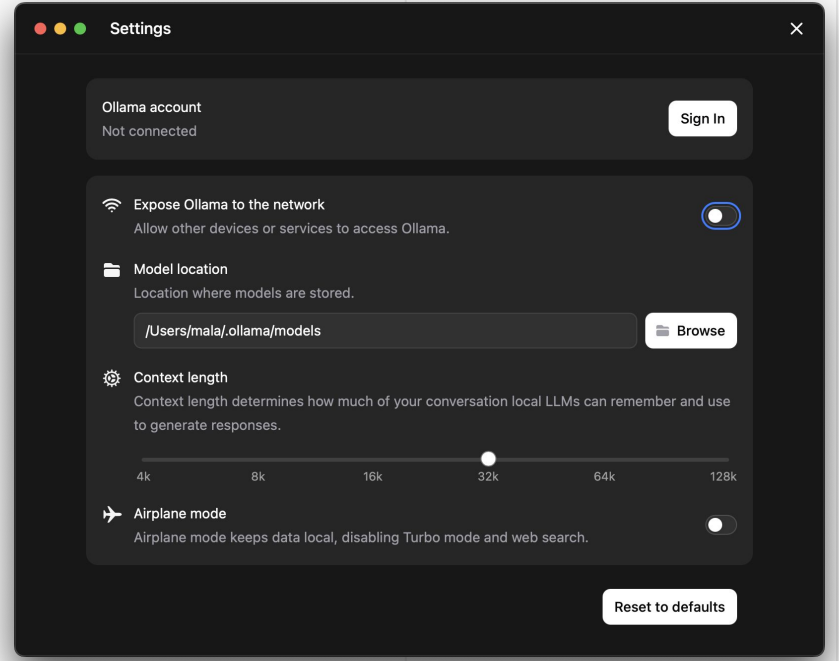
Who owns search tools?

Welcome OpenAI's gpt-oss!

Ollama partners with OpenAI to bring its latest state-of-the-art open weight models to Ollama. The two models, **20B** and **120B**, bring a whole new local chat experience, and are designed for powerful reasoning, agentic tasks, and versatile developer use cases.

Feature highlights

- **Agentic capabilities:** Use the models' native capabilities for function calling, web browsing (Ollama is providing a built-in web search that can be optionally enabled to augment the model with the latest information), python tool calls, and structured outputs.



Conclusions

Learnings

- There’s **tinkering** with AI and tinkering **with AI**
  - you’ll solve things more quickly with the former, learn more with the latter

Learnings

- There's **tinkering** with AI and tinkering **with AI**
  - you'll solve things more quickly with the former, learn more with the latter
- No one-size-fits-all solution
  - **task**, **compute**, **model training**, **tools**

Learnings

- There's **tinkering** with AI and tinkering **with AI**
  - you'll solve things more quickly with the former, learn more with the latter
- No one-size-fits-all solution
  - **task**, **compute**, **model training**, **tools**
- Easy, schmeasy: "friendly" vs actually useful
  - UX is evolving from simple chat to agents, expect things to break

Learnings

- There's **tinkering** with AI and tinkering **with AI**
  - you'll solve things more quickly with the former, learn more with the latter
- No one-size-fits-all solution
  - **task**, **compute**, **model training**, **tools**
- Easy, schmeasy: "friendly" vs actually useful
  - UX is evolving from simple chat to agents, expect things to break
- Think smol
  - limit context growth
  - few tools bring you a long way

Learnings

- There's **tinkering** with AI and tinkering **with AI**
  - you'll solve things more quickly with the former, learn more with the latter
- No one-size-fits-all solution
  - **task**, **compute**, **model training**, **tools**
- Easy, schmeasy: "friendly" vs actually useful
  - UX is evolving from simple chat to agents, expect things to break
- Think smol
  - limit context growth
  - few tools bring you a long way
- ~~LLMs~~ Agents are slot machines
  - remember the success, **don't forget the failures**

Learnings

- There's **tinkering** with AI and tinkering **with AI**
  - you'll solve things more quickly with the former, learn more with the latter
- No one-size-fits-all solution
  - **task**, **compute**, **model training**, **tools**
- Easy, schmeasy: "friendly" vs actually useful
  - UX is evolving from simple chat to agents, expect things to break
- Think smol
  - limit context growth
  - few tools bring you a long way
- ~~LLMs~~ Agents are slot machines
  - remember the success, **don't forget the failures**
- Prevent **enshittification**
  - what (data, freedoms, control) are you giving away with each choice?

A new problem: stalking agents?

file:///Users/mala/workspace/wasm-agents-blueprint/demos/local_mcpd.html

Custom Prompt:

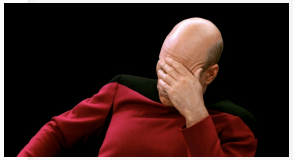
Search the web for Davide Eynard's birthdate. Tell me both the date and the URL where you found it.

Quick examples: Strawrberrry | GitHub stars | Blog post

🚀 Run Agent | Clear Agent Output | View response as log

Davide Eynard

- **Date of birth: 11 February 1976**
- **Source:** The PDF CV hosted at the UC Lugano Institute of Computational Sciences lists "Birthplace and date: February 11th 1976, Monza (MB), Italy."
  - **URL:** `http://davide.eynard.it/cv/cv_20221108.pdf`



Inspector | Console | Debugger | Network | Style Editor | Performance | Memory | Storage | Accessibility | Application

Filter URLs | All | HTML | CSS | JS | XHR | Fonts | Images | Media | WS | Other | Disable Cache | No Throttling

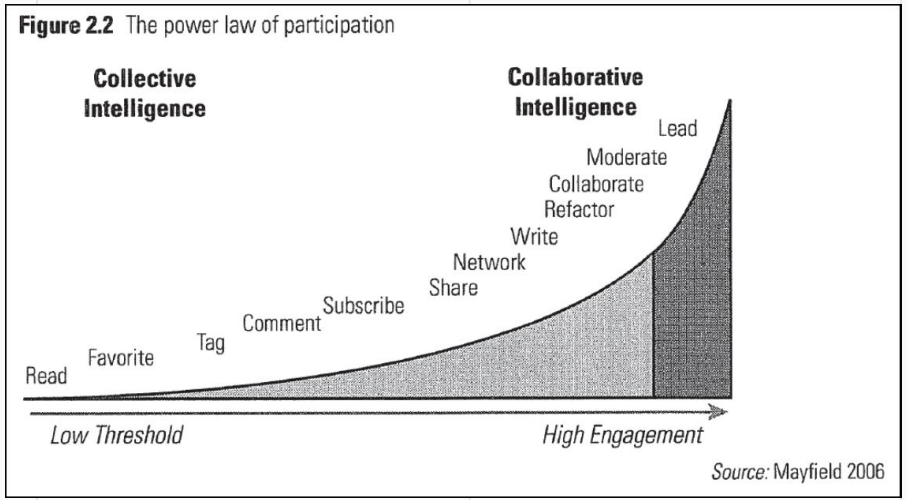
Status	Method	Domain	File	Initiator	Type	Transferred	Si...
200	OPTIONS	89.169.109.132:8090	searxng_web_search	xhr	plain	293 B	0 B
200	POST	89.169.109.132:11434	completions	pyodide.asm.js:10 (fetch)	json	755 B	6...
200	POST	89.169.109.132:8090	searxng_web_search	pyodide.asm.js:10 (xhr)	json	12.93 kB	1...
200	POST	89.169.109.132:11434	completions	pyodide.asm.js:10 (fetch)	json	753 B	6...
200	GET	davide.eynard.it	/about-me/	pyodide.asm.js:10 (xhr)	html	5.29 kB	17...
200	POST	89.169.109.132:11434	completions	pyodide.asm.js:10 (fetch)	json	729 B	5...
200	POST	89.169.109.132:8090	searxng_web_search	pyodide.asm.js:10 (xhr)	json	13.78 kB	1...
200	POST	89.169.109.132:11434	completions	pyodide.asm.js:10 (fetch)	json	952 B	8...
200	GET	davide.eynard.it	cv_20221108.pdf	pyodide.asm.js:10 (xhr)	pdf	90.47 kB	9...
200	POST	89.169.109.132:11434	completions	pyodide.asm.js:10 (fetch)	json	1.55 kB	1....
502	POST	89.169.109.132:8090	fetch_url	pyodide.asm.js:10 (xhr)	problem	1.51 kB	1....
200	POST	89.169.109.132:11434	completions	pyodide.asm.js:10 (fetch)	json	945 B	8...
200	GET	www.researchgate.net	Davide-Eynard	pyodide.asm.js:10 (xhr)	html	60.15 kB	7...
200	POST	89.169.109.132:11434	completions	pyodide.asm.js:10 (fetch)	json	743 B	6...

22 requests | 897.64 kB / 214.53 kB transferred | Finish: 1.58 min

And now?

Choose your own adventure:

- Check out mozilla-ai GitHub org!
- Play with different agents
- Write your own “any-agent”
- Test different local models
- Try different AI tools and MCP servers
- Host tools / services for your community



And now?

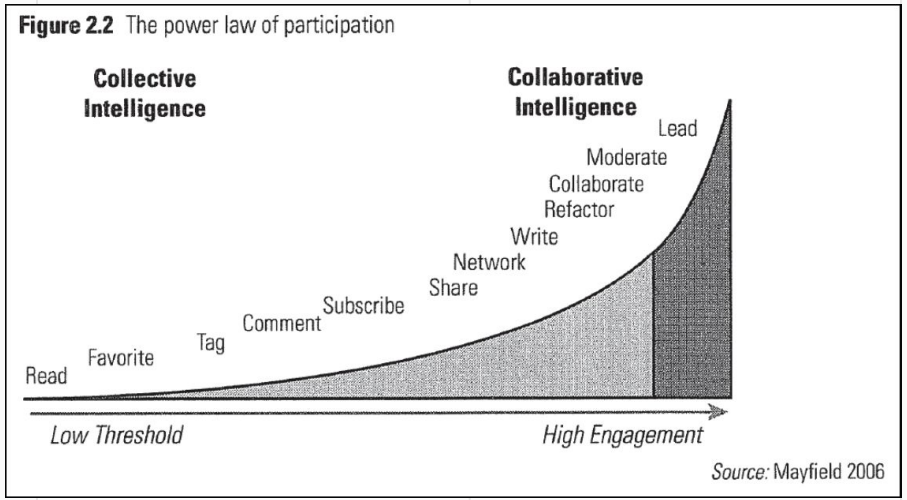
Mozilla.ai

mcpd

encoderfile

Choose your own adventure:

- Check out mozilla-ai GitHub org!
- Play with different agents
- Write your own “any-agent”
- Test different local models
- Try different AI tools and MCP servers
- Host tools / services for your community



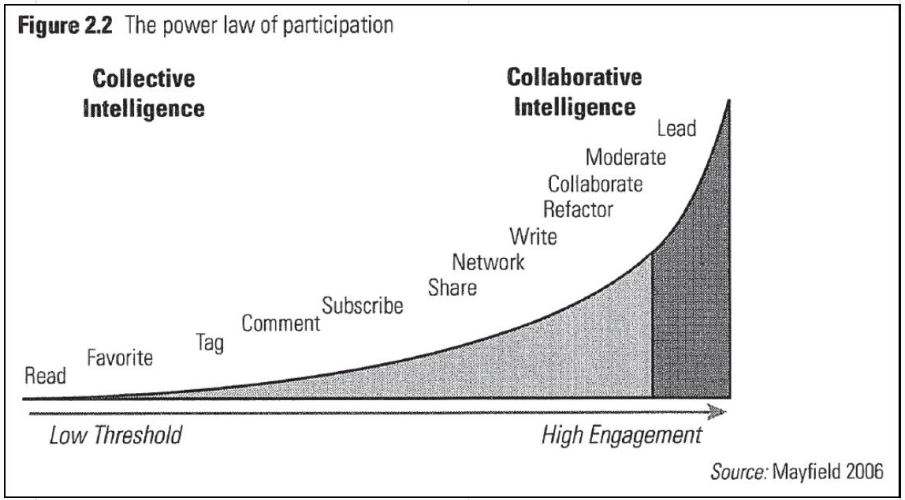
And now?




Mozilla.ai

Choose your own adventure:

- Check out mozilla-ai GitHub org!
- Play with different agents
- Write your own “any-agent”
- Test different local models
- Try different AI tools and MCP servers
- Host tools / services for your community



BE LIKE ADA!

Thank you!