DHBW Ravensburg

Disciplined Vibe Coding

The unreasonable effectiveness of language

Prof. Dr.-Ing. Mark Schutera | July 6, 2026 · *orange disciplined dino*

[1, 3, 6]

1

Vibing

When Language becomes Action

Designing Interfaces by Injection

The model only reads and writes tokens. A tool still needs an interface, so we make one the same way as before: by injecting its description into the prompt, then reading the call back out of the reply.

- The **system prompt** specifies the tool, its arguments, and the format for calling it.
- The model writes that call as ordinary text; a **parser** spots it and runs the real tool.
- The result comes back as tokens and get injected as observations, and the model continues.

System prompt

```
tool weather(city)
call <tool>...</tool>
```

LLM Tool use

```
<tool>
weather(Ravensburg)
</tool>
```

MCP and plug in outside tools

The **Model Context Protocol** is a standard way to connect the agent to outside tools and data: a database, a web service, a file store, a program, each exposed through a small server.

MCP is a protocol. The agent host connects, discovers them at runtime, and calls them. Write to the standard once, and any MCP client can use it.

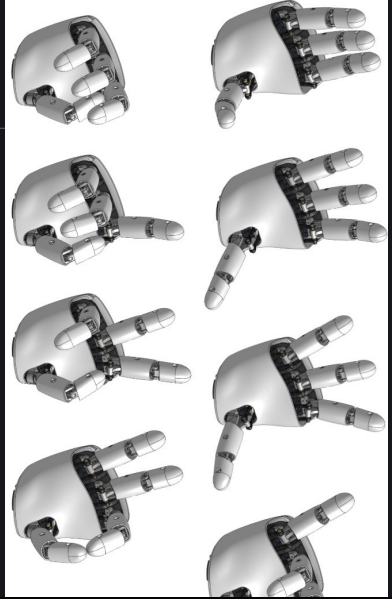
This is the structured answer to agent-ready APIs. The engineering way with proper auth and hardened.

The one Acting

Agentic, from Latin *agere*, “to act,” the root of *agent*, one who acts.

- Emits its action in a fixed structured format using tags.
- Stops at `</action> or </tool>` once emitted.
- A parser then runs the action.

An action can be many things. Communication and subagent orchestration. Information gathering through search, queries, and retrieval. It uses tools, from API calls to code execution. Or it acts on the environment, driving digital or embodied interfaces.



The Harness

The machine got an interface, now it's time for the human to get one to take control.

> The **harness** is everything wrapped around the raw model to make it reliable: the client, the system prompt, the skills, the tests, the permission gates, the loop that runs them.

`opencode` is your primary harness interface: type **/** to open the command menu and pick by name.

It has a set of **built-in tools** which can be called directly: `bash`, `read`, `write`, `edit`, `glob`, `grep`, `webfetch`.

Getting started

Run it	What it does
`/init`	Scans the project and writes an `AGENTS.md` of your stack, scripts, and conventions, so every later session starts oriented and in **context**. Run once per repo.
`/help`	Opens the in-app help: the command list, the current keybinds, and where the docs live. Your first stop when a key surprises you.
`/exit` `ctrl+x q`	Ends the session and drops you back to the shell. `ctrl+c` on an empty prompt quits too.

Steer by Context

Run it	What it does
`/compact` `ctrl+x c`	Summarises the conversation into a short brief and continues from it, reclaiming context-window space without losing the thread. Don't wait too long.
`/new` `ctrl+x n`	Starts a fresh session with an empty context. Your project stays; the conversation so far is dropped. Reach for it when the topic changes.
`/sessions` `ctrl+x l`	Lists your earlier sessions and jumps to the one you pick.
`ctrl+r`	**Rename** the current session to a memorable title, so you find it again in the list.
`esc`	**Interrupt** the agent mid-run the moment it heads the wrong way, instead of waiting for it to finish.

Gating and sandboxing

Gating keeps the human on the loop, the sanbox stops agent escapes. Latest when you auto, you want a sandbox.

Gating

opencode asks before it edits or runs: `allow`, `ask`, `deny`, set per tool in `opencode.json`. Ships in the box.

Stops surprises. It cannot stop a command you already approved.

Sandboxing

A container or VM. A project mounted, network restricted, `opencode` running inside it. You build it yourself.

Stops escapes. If something goes wrong it wrecks the box, not your machine.

Reasoning or the Inner Monologue

[1, 3, 6]

Prompted at inference time

reasoning coaxed out by the prompt, no weights changed.

Chain-of-thought prompting appends a cue like *"Let's think step by step,"* steering decoding toward a plan rather than a snap answer, so the model breaks the task into sub-tasks.

We can interleave those reasoning steps with actions and observations so thinking and tool calls advance together, called ReAct.

Trained into the weights

reasoning trained not nudged.

Models like DeepSeek-R1 and OpenAI's o1 are trained with **human feedback reinforcement learning** to reason before they answer.

> R1 emits a thinking section between the tokens `<think>` and `</think>` before its answer.

[4–6]

Interface and input

Run it	What it does
`/editor` `ctrl+x e`	Opens your `$EDITOR` to compose a long message with full editing, then sends it back to the prompt.
`ctrl+v`	**Paste** clipboard content into the prompt.
`ctrl+c`	**Clear** the prompt box in one stroke.
`/details`	Toggles a tool call's full input and output, to inspect what ran in detail.
`/thinking`	Toggles the model's reasoning `<think>` blocks, if the model has this capability.

Vibe an infinitely running banana

Now vibe an endless runner in a single `index.html`. Then ask the LLM to run it in your browser and play, then iterate.

- How easy is it to get one specific thing done?
- Watch your context and pay attention to the impact of a saturated context.
- Reflect on how productive this feels versus how productive you are.

Feel the floor rising already? Lines of code have never been cheaper.

Screenshot of a banana again.

References

1 NVIDIA et al. “GR00T N1: An Open Foundation Model for Generalist Humanoid Robots”. In: *arXiv preprint arXiv:2503.14734* (2025).

2 Mustafa Shukor et al. “SmolVLA: A Vision-Language-Action Model for Affordable and Efficient Robotics”. In: *arXiv preprint arXiv:2506.01844* (2025).

3 Tejas Kumar. *Harnesses in AI: A Deep Dive*. 2026.

4 Jason Wei et al. “Chain-of-Thought Prompting Elicits Reasoning in Large Language Models”. In: *arXiv preprint arXiv:2201.11903* (2022).

5 Shunyu Yao et al. “ReAct: Synergizing Reasoning and Acting in Language Models”. In: *arXiv preprint arXiv:2210.03629* (2022).

6 DeepSeek-AI. “DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning”. In: *arXiv preprint arXiv:2501.12948* (2025).