Pi Coding Agent Handbook

An unofficial community handbook

Pi Handbook Community

Table of contents

1. Pi Coding Agent Handbook

Agentic coding is probably the best thing that's happened since the ChatGPT moment. These tools are how individual builders ship software that used to take teams — and the more we push them, the more high-quality personal software the world gets. That matters.

I've been through the cycle — Aider, Cursor, Claude Code, Amp, OpenCode, Codex, Goose, Kiro, Kilo, Roo — the list goes on. I still use many of them daily. I'd already cut most of my Claude Code time once Amp came along, but with pi I'm feeling like I could finally stop reaching for it altogether.

Let me be honest about one thing. Claude Code is innovative. It was mindblowing for a long time and it's still pushing forward. I respect what it's done for this space. I just don't like its arrogance. That's personal, and you're welcome to disagree. Pi is the opposite — humble, minimal, and trusts you to drive. That's the energy I want to build with.

This handbook is what I wish existed when I started with pi. It covers the philosophy, the architecture, the extension system, the tradeoffs, and the honest gaps. It's opinionated because pi is opinionated, and that's the point.

I don't claim to be an expert. I'm just a builder who got hooked and wanted to write things down. If something here is wrong, outdated, or could be said better — please fix it. This is your handbook too.

1.1 What's inside

The core argument for pi, the four-tool architecture, how extensions replace features other agents bake in, the no-MCP stance (and why MCP 2.0 might change the calculus), community packages like oracle and handoff, a fair comparison with Claude Code / Amp / OpenCode / Aider, and a quick start that gets you running on macOS, Linux, or Android via Termux.

12 chapters. Code snippets. Every claim sourced.

1.2 Get involved

This handbook lives or dies by contributions. A typo fix, a better code example, a one-paragraph tip, a whole new chapter — it all counts. The Limitations page is full of gaps waiting to be filled. Every one of them is an open invitation.

If you've built something with pi — an extension, a workflow, a workaround — write it up. Someone out there is stuck on the exact problem you solved last week. We're especially interested in patterns for using pi as a general agent orchestrator — coordinating sub-agents, chaining tasks, making the build process more fun, not just more productive.

Contribute **Download PDF**

Disclaimer

This is an unofficial community handbook. It is not affiliated with or endorsed by Mario Zechner or the pi project. All quotes are attributed to their original authors and linked to their sources. Pi is MIT licensed.

1.3 Chapters

	Chapter	What you'll learn
1	Core Philosophy	Why pi exists, what it deliberately omits
2	Architecture	The minimal core: read, write, edit, bash
3	Extension System	TypeScript extensions, skills, packages
4	Session Management	Tree-structured branching, compaction
5	Multi-Model Freedom	15+ providers, mid-session switching
6	No MCP	The CLI-first philosophy and its tradeoffs
7	Community Extensions	Oracle, handoff, sub-agents, and more
8	Comparison	Pi vs Claude Code vs Amp vs others
9	Limitations	Honest assessment of current gaps
10	Quick Start	Installation, config, daily workflows
	Code Snippets	Concrete extension and config examples
	References	Official docs, community links, articles

1.4 Key references

Resource	Link
Website	shittycodingagent.ai
GitHub	badlogic/pi-mono
npm	@mariozechner/pi-coding-agent
Mario's blog	What I learned building a minimal coding agent
MCP vs CLI benchmark	MCP vs CLI: Benchmarking Tools
Armin Ronacher	Pi: The Minimal Agent Within OpenClaw

2. Core Philosophy

Pi's design is guided by a single principle: **if I don't need it, it won't be built.**

Mario Zechner built pi because existing agents (Claude Code, Cursor, Codex) kept adding features he didn't use, injecting hidden context he couldn't inspect, and changing behavior on every release. Pi is his answer: a coding agent that is aggressively minimal, fully transparent, and extensible by the user -- not the vendor.

2.1 The Minimal Core

Pi ships with exactly four tools: `read`, `write`, `edit`, `bash`. That's it. No plan mode, no sub-agents, no MCP, no permission popups, no built-in to-dos, no background bash.

From the website:

> Pi is aggressively extensible so it doesn't have to dictate your workflow. Features that other tools bake in can be built with extensions, skills, or installed from third-party pi packages. This keeps the core minimal while letting you shape pi to fit how you work.

2.2 Context Engineering as First Principle

Pi has the shortest system prompt of any coding agent. This matters because every token in the system prompt competes with your actual code and instructions for the model's attention.

From Ewald Benes (r/ClaudeCode):

> By default, tools like Claude Code inject a massive amount of hidden text before your prompt even begins. This "bloat" burns through thousands of tokens before you've typed a single word. The creators of these tools make dozens of architectural decisions for you -- whether you want them or not -- which often leads to the LLM becoming "distracted" by its own internal instructions.

Pi's approach: give the model the minimum it needs, and let the user control what else goes in via AGENTS.md, SYSTEM.md, skills, and extensions.

2.3 Primitives, Not Features

Instead of building plan mode, pi gives you extensions. Instead of building sub-agents, pi gives you extensions. Instead of building MCP support, pi gives you bash + skills.

From Armin Ronacher:

> This is not a lazy omission. This is from the philosophy of how Pi works. Pi's entire idea is that if you want the agent to do something that it doesn't do yet, you don't go and download an extension or a skill or something like this. You ask the agent to extend itself. It celebrates the idea of code writing and running code.

2.4 Software That Builds Itself

Pi ships with its own documentation and extension examples that the agent can read. This means you can literally tell pi: "Build me an extension that does X" and it will read the docs, write the extension, hot-reload it, test it, and iterate until it works.

From Armin Ronacher:

> It also ships with documentation and examples that the agent itself can use to extend itself. Even better: sessions in Pi are trees. You can branch and navigate within a session which opens up all kinds of interesting opportunities such as enabling workflows for making a side-quest to fix a broken agent tool without wasting context in the main session.

2.5 The "What We Didn't Build" List

Feature	Pi's stance	Alternative
MCP	Will not support	CLI tools + READMEs, or mcporter
Sub-agents	Not built-in	Extensions, tmux, or community packages
Plan mode	Not built-in	Just tell it "make a plan", or install an extension
Permission popups	Not built-in	Run in container, or build confirmation extension
To-do tracking	Not built-in	Use TODO.md, or build a tool extension
Background bash	Not built-in	Use tmux for full observability

2.6 YOLO by Default

Pi runs in "YOLO mode" -- it executes tools without asking for permission. This is a deliberate choice. Mario's argument: if the agent can write and run code, security theater (confirmation dialogs) doesn't actually protect you. If you want safety, use a container or build a real permission gate via extensions.

From the blog post:

> If you look at the security measures in other coding agents, they're mostly security theater. As soon as your agent can write code and run code, it's pretty much game over.

3. Architecture and Four Tools

3.1 The Monorepo

Pi is built as a monorepo (`badlogic/pi-mono`) with cleanly separated packages:

Package	**Purpose**
`pi-ai`	Unified LLM API across 4 wire protocols (OpenAI Completions, OpenAI Responses, Anthropic Messages, Google Generative AI)
`pi-agent-core`	Agent loop, tool execution, event streaming
`pi-tui`	Terminal UI framework with differential rendering, flicker-free output
`pi-coding-agent`	The CLI that wires it all together
`pi-mom`	Slack bot / autonomous agent built on pi
`pi-web-ui`	Web-based chat interface components
`pi-pods`	vLLM pod management for self-hosting

3.2 Four Tools

The agent has exactly four built-in tools:

```
read   -- Read file contents (text and images)
write  -- Create or overwrite files
edit   -- Surgical find-and-replace edits
bash   -- Execute shell commands
```

That's the entire tool surface. Everything else is built on top via extensions.

Why only four? Because these are the primitives that cover 95% of coding tasks. More tools means more token overhead in the system prompt, more confusion for the model, and more things that can break between releases.

3.3 Four Execution Modes

```
Interactive  -- Full TUI experience (default)
Print/JSON   -- pi -p "query" for scripts, --mode json for event streams
RPC          -- JSON protocol over stdin/stdout for non-Node integrations
SDK          -- Embed pi in your own apps (how OpenClaw is built)
```

3.4 Context Control Stack

Pi provides multiple layers for controlling what enters the model's context:

```
SYSTEM.md        -- Replace or append to the default system prompt (per-project)
AGENTS.md        -- Project instructions, loaded from ~/.pi/agent/, parent dirs, and cwd
Skills           -- On-demand capability packages (progressive disclosure)
Prompt templates -- Reusable prompts as markdown files (/name to expand)
Extensions       -- Dynamic context injection, RAG, message filtering, compaction
```

The key insight: skills are loaded on-demand (the agent reads the README only when relevant), which means you pay the token cost only when needed. This is "progressive disclosure" -- the opposite of MCP, which dumps all tool descriptions into context at session start.

3.5 Session Format

Sessions are JSONL files with a tree structure. Each entry has an `id` and `parentId`, enabling in-place branching without creating new files. The format supports:

- User messages, assistant messages, tool results
- Bash execution records (command, output, exit code)
- Custom messages (extension state, persisted across restarts)
- Branch summaries and compaction summaries
- Full token usage and cost tracking per message

See 04-sessions.md for details.

4. Extension System

Pi's extension system is its most powerful differentiator. Extensions are TypeScript modules that can hook into every aspect of the agent's lifecycle.

4.1 Capabilities

- Register custom tools the LLM can call
- Intercept and block/modify tool calls (permission gates)
- Inject context before each turn (RAG, memory)
- Filter and transform message history
- Customize compaction behavior
- Register slash commands (`/mycommand`)
- Register keyboard shortcuts
- Render custom TUI components (dashboards, pickers, overlays)
- Persist state into sessions (survives restarts)
- Register CLI flags

4.2 Quick Start

Create `~/.pi/agent/extensions/my-extension.ts` :

```typescript
import type { ExtensionAPI } from "@mariozechner/pi-coding-agent";
import { Type } from "@sinclair/typebox";

export default function (pi: ExtensionAPI) {
  // React to events
  pi.on("session_start", async (_event, ctx) => {
    ctx.ui.notify("Extension loaded!", "info");
  });

  // Block dangerous commands
  pi.on("tool_call", async (event, ctx) => {
    if (event.toolName === "bash" && event.input.command?.includes("rm -rf")) {
      const ok = await ctx.ui.confirm("Dangerous!", "Allow rm -rf?");
      if (!ok) return { block: true, reason: "Blocked by user" };
    }
  });

  // Register a custom tool
  pi.registerTool({
    name: "greet",
    label: "Greet",
    description: "Greet someone by name",
    parameters: Type.Object({
      name: Type.String({ description: "Name to greet" }),
    }),
    async execute(toolCallId, params, signal, onUpdate, ctx) {
      return {
        content: [{ type: "text", text: `Hello, ${params.name}!` }],
        details: {},
      };
    },
  });

  // Register a command
  pi.registerCommand("hello", {
    description: "Say hello",
    handler: async (args, ctx) => {
      ctx.ui.notify(`Hello ${args || "world"}!`, "info");
    },
  });
}
```

Test without installing:

```
pi -e ./my-extension.ts
```

Hot-reload after changes:

```
/reload
```

4.3 Extension Locations

Location	Scope
`~/.pi/agent/extensions/*.ts`	Global (all projects)
`~/.pi/agent/extensions/*/index.ts`	Global (subdirectory)
`.pi/extensions/*.ts`	Project-local
`.pi/extensions/*/index.ts`	Project-local (subdirectory)

4.4 Event Lifecycle

```
session_start
  |
  v
input --> before_agent_start --> agent_start
  |
  +-- turn_start
  |     +-- context (can modify messages)
  |     +-- tool_call (can block/modify)
  |     +-- tool_result (can transform output)
  +-- turn_end
  |
  v
agent_end --> (next user input)
```

4.5 Skills vs Extensions

	Skills	Extensions
Format	Markdown files	TypeScript modules
Loaded	On-demand by the agent	At startup
Purpose	Instructions + conventions	Tools + behavior + UI
Token cost	Only when relevant	Tool descriptions always in context
Sharing	Pi packages	Pi packages

Skills are ideal for teaching the agent how to use a CLI tool or follow a convention. Extensions are for when you need programmatic control.

4.6 Pi Packages

Bundle extensions, skills, prompts, and themes as npm or git packages:

```
# Install from npm
pi install npm:shitty-extensions

# Install from git
pi install git:github.com/badlogic/pi-doom

# Pin version
pi install npm:@foo/bar@1.2.3

# Project-local (shared with team via .pi/)
pi install -l npm:shitty-extensions

# Try without installing
pi -e npm:shitty-extensions
```

```
# Update all
pi update

# List installed
pi list
```

Packages use the `pi-package` keyword on npm for discoverability.

4.7 Self-Extending Agent

The most powerful pattern: ask pi to build its own extensions.

```
"Build me an extension that tracks how many tokens each tool call uses
and shows a summary widget above the editor."
```

Pi will read its own extension docs, write the TypeScript, hot-reload, test, and iterate. This is what Armin Ronacher calls "agents built for agents building agents."

5. Session Management

Pi's session system is tree-structured, which sets it apart from the linear chat history in most agents.

5.1 Core Concepts

Sessions are stored as JSONL files at:

```
~/.pi/agent/sessions/<project-path>/<session-id>.jsonl
```

Each entry has an `id` and `parentId`, forming a tree. All branches live in a single file -- no proliferation of session files when you explore alternatives.

5.2 Key Commands

```
# Start pi (new session)
pi

# Continue most recent session in current project
pi -c

# Browse and pick from all past sessions (across projects)
pi -r

# Jump to a specific session by ID
pi --session abc123

# Ephemeral mode (don't save)
pi --no-session
```

5.3 In-Session Navigation

Command	What it does
`/tree`	Navigate the full session tree, jump to any point, continue from there
`/fork`	Create a new session from current branch (closest to Amp's handoff)
`/compact`	Summarize older messages, keep working
`/compact <prompt>`	Directed compaction ("focus on the API refactor only")
`/name <label>`	Label the current session
`/export [file]`	Export session to HTML
`/share`	Upload as private GitHub gist with shareable URL

5.4 Branching Workflow

```
Main conversation
  |
  +-- Message 1
  +-- Message 2
  +-- Message 3  <-- /tree here, select Message 2
        |
        +-- Branch A (original continuation)
        +-- Branch B (new direction from Message 2)
```

All branches are preserved. Use `/tree` to switch between them. Filter modes:

- Default view
- No-tools (hide tool calls)
- User-only
- Labeled-only (bookmarks)
- All entries

Press `l` in tree view to label entries as bookmarks for quick navigation.

5.5 Compaction

Long sessions exhaust context windows. Compaction summarizes older messages while keeping recent ones.

- **Manual:** `/compact` or `/compact <instructions>`
- **Automatic:** Triggers on context overflow (recovers and retries) or proactively when approaching the limit
- **Customizable:** Extensions can implement topic-based compaction, code-aware summaries, or use different summarization models

5.6 Cross-Project Sessions

Sessions are organized by working directory, but `pi -r` shows sessions from ALL projects. You can jump to a session from a different project:

```
pi --session <id-from-other-project>
# Pi will ask if you want to fork it into current directory
```

5.7 Comparison with Amp's Thread Model

Feature	Amp	Pi
Thread/Session	Server-side threads with IDs	Local JSONL files with IDs
Handoff	Auto-generates focused prompt for new thread	`/fork` -- copies history, you edit the starting message
Thread mentioning	`@thread-id` from anywhere	Not native (read session file manually or build extension)
Restore to point	Hover + Restore button	`/tree` -- navigate and continue
Thread map	Visual graph of connected threads	`/tree` with filter modes

The main gap: pi doesn't have native cross-session referencing. You can't `@mention` another session from within a session. Workaround: ask pi to read the JSONL file, or build a `/recall <session-id>` extension.

6. Multi-Model Freedom

One of pi's killer features: seamless model switching mid-session across 15+ providers.

6.1 Supported Providers

Via subscription (OAuth): - Anthropic Claude Pro/Max - OpenAI ChatGPT Plus/Pro (Codex) - GitHub Copilot - Google Gemini CLI - Google Antigravity

Via API key: - Anthropic, OpenAI, Azure OpenAI, Google Gemini, Google Vertex - Amazon Bedrock, Mistral, Groq, Cerebras, xAI - OpenRouter, Vercel AI Gateway, ZAI, OpenCode Zen - Hugging Face, Kimi For Coding, MiniMax

Self-hosted: - Ollama, llama.cpp, vLLM, LM Studio (via OpenAI-compatible API)

6.2 Switching Models

```
Ctrl+L          -- Open model selector (full list)
Ctrl+P          -- Cycle through scoped/favorite models
Shift+Ctrl+P    -- Cycle backward
/model           -- Switch via command
```

6.3 Context Handoff

Pi-ai is designed from the ground up for cross-provider context handoff. When you switch from Anthropic to OpenAI mid-session:

- Thinking traces are converted to `<thinking></thinking>` content blocks
- Provider-specific signed blobs are handled transparently
- Tool call history is preserved across providers
- Token/cost tracking continues accurately

This is best-effort (providers have different capabilities), but it works well in practice.

From Ewald Benes:

Models have finally become a commodity to me. I'm currently cycling through Anthropic, z.ai, and Moonshot AI (Kimi) within the same session. I can swap the "brain" of the agent mid-stream, and the new model picks up the context seamlessly where the last one left off.

6.4 Model Aliases (via pi-model-switch extension)

Install the community extension:

```
pi install npm:pi-model-switch
```

Configure aliases in `~/.pi/agent/extensions/model-switch/aliases.json`:

```json
{
  "cheap": "google/gemini-2.5-flash",
  "fast": "google/gemini-2.5-flash",
  "coding": "anthropic/claude-opus-4-5",
  "budget": ["openai/gpt-5-mini", "google/gemini-2.5-flash"]
}
```

Then just say: "switch to cheap" or "use the coding model for this refactor."

6.5 Custom Providers

Add providers via `~/.pi/agent/models.json` if they speak a supported API:

```
import { getModel, stream } from "@mariozechner/pi-ai";

const ollamaModel = {
  id: "llama-3.1-8b",
  name: "Llama 3.1 8B (Ollama)",
  api: "openai-completions",
  provider: "ollama",
  baseUrl: "http://localhost:11434/v1",
  reasoning: false,
  input: ["text"],
  cost: { input: 0, output: 0, cacheRead: 0, cacheWrite: 0 },
  contextWindow: 128000,
  maxTokens: 32000,
};
```

6.6 Scoped Models

Configure a subset of models for quick cycling with `Ctrl+P`:

```
/scoped-models    -- Enable/disable models for cycling
```

Use `--models <patterns>` on the CLI for comma-separated patterns.

6.7 Practical Workflow

A typical multi-model session:

1. Start with Gemini Flash for quick exploration (cheap, fast, huge context)
2. Switch to Claude Sonnet for implementation (best coding)
3. Bring in GPT-5.2 via oracle extension for review (strong reasoning)
4. Switch to a cheap model for boilerplate/tests

All in one session, all context preserved.

7. No MCP: The CLI-First Philosophy

This is pi's most controversial and deliberate design decision.

"pi does not and will not support MCP." -- Mario Zechner

7.1 The Problem with MCP

Popular MCP servers dump their entire tool descriptions into your context on every session:

MCP Server	Tools	Token cost
Playwright MCP	21 tools	~13,700 tokens
Chrome DevTools MCP	26 tools	~18,000 tokens

That's 7-9% of your context window gone before you start working. Most of those tools won't be used in a given session.

Additionally, many MCP servers are thin wrappers around CLI tools that already exist:

Just like a lot of meetings could have been emails, a lot of MCPs could have been CLI invocations. For example, there's the GitHub MCP Server, which reimplements functionality that's already available in the GitHub CLI. There's little benefit of using that MCP compared to telling your coding agent to use its shell tool to run the GitHub CLI directly.

7.2 The CLI Alternative

Pi's approach: build CLI tools with README files.

1. The agent reads the README only when it needs the tool (progressive disclosure)
2. Token cost is paid only on demand
3. CLI tools are composable (pipe outputs, chain commands)
4. CLI tools are easy to extend (just add another script)
5. LLMs already know how to use CLI tools from training data

Example -- adding web search to pi via a skill:

```
# SKILL.md
name: web-search
description: Search the web using the search CLI tool

## Usage
Run `search "your query"` to search the web.
Run `search --fetch <url>` to read a page.
See the README at ~/agent-tools/search/README.md for full options.
```

Mario maintains a collection of CLI tools at `github.com/badlogic/agent-tools`.

7.3 The Benchmark

Mario ran a formal evaluation comparing MCP vs CLI for coding agents (August 2025):

Setup: terminalcp (his tmux alternative) as both MCP server and CLI, compared against tmux and screen. Three tasks, 10 runs each, using Claude Code.

Results:

Metric	terminalcp MCP	terminalcp CLI	tmux	screen
Success rate	100%	100%	100%	67%
Total time	51 min	66 min	~60 min	~70 min
Total cost	$19.45	$19.95	$22	$22+

Key findings:

- MCP vs CLI is a wash on success rates
- MCP was 23% faster due to bypassing Claude Code's security checks on bash
- Tool design and documentation quality matter far more than the protocol
- For complex tasks, well-designed tools beat standard tools by 39% on cost

Conclusion:

Maybe instead of arguing about MCP vs CLI, we should start building better tools. The protocol is just plumbing. What matters is whether your tool helps or hinders the agent's ability to complete tasks.

If you're building a tool from scratch and your users already have a shell tool available, just make a good CLI. It's simpler and more portable. Plus, the output of your CLI can be further filtered and massaged just by piping it into another CLI tool, which can increase token efficiency at the cost of additional instructions. That's not possible with MCPs.

7.4 If You Must Use MCP

Peter Steinberger's mcporter wraps MCP servers as CLI tools, giving you the best of both worlds. OpenClaw uses this approach.

7.5 The Deeper Argument

From Armin Ronacher:

If you consider how MCP works, on most model providers, tools for MCP, like any tool for the LLM, need to be loaded into the system context or the tool section thereof on session start. That makes it very hard to impossible to fully reload what tools can do without trashing the complete cache or confusing the AI about how prior invocations work differently.

Pi's skill system avoids this entirely. Skills are loaded into context only when the agent determines they're relevant, and they can be unloaded when no longer needed. This is fundamentally incompatible with how MCP tools work.

7.6 Community Perspective

From r/ClaudeAI:

Why use MCP in subagents when they can use CLI with 0 tool context overhead?

From Cobus Greyling:

What if the best interface for AI Agents is not a new protocol at all? What if it is the command line -- the same interface that has been powering software for over fifty years?

The CLI-over-MCP movement is growing, but pi is the only major agent that has made it a first-class architectural decision.

7.7 The Counter-Argument: MCP Is Evolving

Pi's no-MCP stance is well-reasoned for today's landscape, but it's worth acknowledging that MCP is not standing still.

MCP 2.0 introduces significant improvements that address some of pi's criticisms:

- **Streamable HTTP transport** -- Replacing stdio, enabling remote multi-tenant tool servers
- **OAuth 2.1 auth flows** -- Making enterprise adoption realistic
- **Elicitation** -- The server can ask the agent for more info mid-execution, enabling richer orchestration
- **Tool annotations** (`readOnlyHint`, `destructiveHint`) -- Letting agents reason about safety before calling tools, which could enable smarter tool selection and reduce the "dump all tools into context" problem

Governance shift: Anthropic has donated MCP to the Agentic AI Foundation, which will be managed by the Linux Foundation. This moves MCP from a single-vendor protocol to an open, community-governed standard -- similar to how Kubernetes moved from Google to the CNCF. With broader governance, MCP is likely to evolve faster and address more real-world pain points.

What MCP supporters would argue:

- The context pollution problem is solvable with better tool selection and lazy loading (MCP 2.0's tool annotations are a step toward this)
- CLI tools break across platforms, have version dependencies, and sometimes lack documentation -- MCP provides a structured contract
- Stateful tools (database connections, browser sessions, long-running processes) are inherently easier with MCP's persistent server model than with CLI invocations
- As MCP becomes an open standard under the Linux Foundation, the ecosystem will mature and the "badly designed wrapper" problem will diminish
- Code execution MCPs (Armin Ronacher's "ubertool" pattern) can expose a single tool that accepts code, combining MCP's statefulness with CLI's composability

The balanced view: Pi's CLI-first approach is demonstrably effective for coding agents with a bash tool. The benchmarks prove it. But MCP and CLI are solving different layers of the problem, and as MCP 2.0 matures under open governance, the gap may narrow. Pi's philosophy remains valid -- progressive disclosure and minimal context are good engineering regardless of protocol -- but declaring "no MCP ever" is a bet that the protocol's evolution won't produce something genuinely better than CLI + README for the use cases pi cares about. Time will tell.

8. Community Extensions

Pi's community has built extensions that cover the major features found in competing agents. Here are the most relevant packages.

8.1 shitty-extensions (by hjanuschka)

The most comprehensive community package. Actively maintained.

```
pi install npm:shitty-extensions
```

Extension	Description	Equivalent in
`oracle.ts`	Get second opinions from other AI models	Amp Oracle
`handoff.ts`	Transfer context to new sessions	Amp Handoff
`plan-mode.ts`	Read-only exploration mode	Claude Code plan mode
`memory-mode.ts`	Save instructions to AGENTS.md	Persistent learning
`cost-tracker.ts`	Session spending analysis	Built-in in Amp/Claude
`clipboard.ts`	Copy text to system clipboard via OSC52	--
`ultrathink.ts`	Rainbow animated "ultrathink" effect	Fun
`loop.ts`	Conditional loops (by mitsuhiko/Armin Ronacher)	--
`flicker-corp.ts`	Authentic fullscreen flicker experience	Parody of Claude Code

8.1.1 Oracle Usage

Once installed, tell the agent:

```
"Ask the oracle to review this implementation"
"Use the oracle to debug this race condition"
"Have the oracle brainstorm alternative approaches"
```

The oracle sends the current context to a second model (configurable) and returns its analysis. It's read-only -- it advises but doesn't make changes.

8.1.2 Handoff Usage

```
/handoff now implement the authentication flow
/handoff execute phase one of the plan
```

Generates a focused prompt for a new session based on the current context and your goal.

8.2 pi-model-switch (by nicobailon)

Lets the agent switch models autonomously.

```
pi install npm:pi-model-switch
```

Configure aliases:

```
{
  "cheap": "google/gemini-2.5-flash",
  "coding": "anthropic/claude-opus-4-5",
  "budget": ["openai/gpt-5-mini", "google/gemini-2.5-flash"]
}
```

Usage:

```
"Switch to a cheaper model"
"Use Opus for this refactor"
"List available models"
```

8.3 pi-subagent-enhanced (by nicobailon)

Full sub-agent support with multiple execution modes.

Mode	Description
Single	`{ agent: "worker", task: "refactor auth" }`
Chain	Sequential tasks with `{previous}` placeholder
Parallel	Multiple tasks running simultaneously
Async	Background execution with notifications

Features: - Output truncation (configurable byte/line limits) - Debug artifacts (input, output, JSONL, metadata per task) - Session-scoped notifications

8.4 @marckrenn/pi-sub-core

Shared usage tracking across providers.

```
pi install npm:@marckrenn/pi-sub-core
```

Tracks usage for: Anthropic, OpenAI Codex, GitHub Copilot, Google Gemini, Antigravity, z.ai, AWS Kiro.

8.5 pi-skills (by Mario Zechner)

Mario's official collection of skills for common development tasks.

```
pi install git:github.com/badlogic/pi-skills
```

These are curated, first-party skills that follow pi's philosophy of progressive disclosure -- the agent loads them on-demand when relevant. Check the repo for the current list of available skills and their descriptions.

https://github.com/badlogic/pi-skills

8.6 Armin Ronacher's Extensions (referenced in his blog)

Armin has built several extensions he describes in his Pi writeup:

- `/answer` -- Extracts questions from the agent's response, presents them in a structured UI, sends answers back
- Custom to-do tracker -- Agent-specific local issue tracker with a tool interface
- Various TUI widgets -- Dashboards, debugging interfaces

His philosophy: point your agent to an existing extension and say "build it like that, but with these changes."

8.7 Notable Community Integrations

- **Emacs frontend** (`dnouri/pi-coding-agent`) -- Full Emacs mode with markdown rendering, streaming, branch navigation
- **OpenClaw** -- Slack/Telegram bot built on pi's SDK
- **pi-mom** -- Mario's autonomous Slack bot

8.8 Finding More Packages

```
# Browse on npm
# Search for keyword: pi-package

# Or check the Discord community server
```

The package registry at shittycodingagent.ai/packages lists community contributions (when npm registry is reachable).

9. Comparison with Other Agents

9.1 Overview Matrix

Feature	Pi	Claude Code	Amp	OpenCode	Aider
Core tools	4	15+	~10	10+	2 (read/edit)
System prompt size	Minimal	Large	Medium	Medium	Minimal
MCP support	No (by design)	Yes	Yes	Yes	No
Sub-agents	Via extension	Built-in	Built-in (Oracle)	Built-in	No
Plan mode	Via extension	Built-in	Built-in	Built-in	No
Session branching	Tree structure	Linear	Threads + fork	Linear	No
Multi-model	15+ providers	Anthropic only	Anthropic + OpenAI	Multi-provider	Multi-provider
Mid-session switch	Yes	No	No	Yes	Yes
Extension system	TypeScript + skills	CLAUDE.md only	Toolboxes	Custom tools	No
Self-extending	Yes (hot-reload)	No	No	No	No
Context handoff	Cross-provider	N/A	N/A	Limited	Limited
YOLO mode	Default	Opt-in	Opt-in	Configurable	No
Open source	MIT	No	No	MIT	Apache 2.0
Price	Pay-per-token or subscription	Subscription	Subscription	Pay-per-token	Pay-per-token

9.2 Pi vs Claude Code

The most common migration path. Key differences:

Why people switch: - Token efficiency: "My token limits last 10x longer" (Ewald Benes) - No hidden context injection - No flickering TUI - System prompt doesn't change on every release - Multi-model support - Session branching

What you lose: - Built-in permission system - Native MCP support - Anthropic-optimized tool calling - Larger community and ecosystem - Enterprise features (SSO, audit logs)

From r/ClaudeCode:

> The selling point of pi is its simplicity. It lacks a lot of fancy features, but that means you get the smallest starting context out there, and you don't pay for things like 'plan mode' or 'todo' -- you just have to do the crazy complicated thing of telling it: `make a plan` if you want to plan something.

9.3 Pi vs Amp

Amp is the closest competitor in philosophy (focused threads, quality over features).

Amp advantages: - Oracle is deeply integrated (auto-invoked, optimized token flow) - Handoff auto-generates focused prompts - Thread mentioning (`@thread-id`) - Thread Map for visual navigation - Server-side thread management - Polished VS Code extension

Pi advantages: - Full multi-model freedom (Amp locks you to Anthropic + OpenAI) - Self-extending agent (Amp can't build its own tools) - Open source (MIT) - No vendor lock-in - Smaller context footprint - Community-driven extension ecosystem

9.4 Pi vs OpenCode

OpenCode (by SST) takes the "everything" approach.

OpenCode advantages: - Built-in MCP support - LSP integration (semantic code intelligence) - Built-in web fetch tool - Go binary (no Node.js dependency)

Pi advantages: - Smaller core, less context overhead - Extension system is far more powerful - Session branching (OpenCode is linear) - Cross-provider context handoff - Self-extending capability

9.5 Pi vs Aider

Aider is even more minimal than pi -- it only reads and edits code.

Aider advantages: - Git-native (auto-commits, diff-based editing) - Repository map for large codebases - No bash tool (can't accidentally break things)

Pi advantages: - Bash tool (can run tests, install deps, debug) - Extension system - Session branching - TUI with rich rendering - Multi-mode (interactive, print, RPC, SDK)

9.6 The Armin Ronacher Perspective

From his blog post comparing Pi and Amp:

> Pi is interesting to me because of two main reasons. First of all, it has a tiny core. It has the shortest system prompt of any agent that I'm aware of and it only has four tools. The second thing is that it makes up for its tiny core by providing an extension system that also allows extensions to persist state into sessions, which is incredibly powerful.
>
> And a little bonus: Pi itself is written like excellent software. It doesn't flicker, it doesn't consume a lot of memory, it doesn't randomly break, it is very reliable and it is written by someone who takes great care of what goes into the software.

10. Limitations and Gaps

An honest assessment of pi's current drawbacks. These are real tradeoffs, not dealbreakers -- but the team should be aware of them.

10.1 No Native Cross-Session References

You cannot `@mention` another session from within a session. Amp's thread mentioning and "Amp Now Reads Threads" feature has no equivalent. Workarounds exist (read the JSONL file, build a `/recall` extension), but it's not first-class.

10.2 No Built-in Permission System

YOLO mode means the agent executes everything without asking. For teams working on production systems, this requires discipline:

- Run in a container or sandboxed environment
- Use a dedicated user account (`useradd claude`)
- Build a permission gate extension (examples exist)
- Or install the community `plan-mode` extension for read-only exploration

From r/ClaudeCode:

> It only supports yolo mode, which is honestly fine but if you want security just use linux and `useradd claude` and use that so it can't access your own home directory.

10.3 No IDE Integration

Pi is terminal-only. There's no VS Code extension, no JetBrains plugin, no native GUI. The closest alternatives:

- Emacs mode (`dnouri/pi-coding-agent`) -- community-built, works well
- RPC mode for building custom integrations
- SDK for embedding in your own apps

If your team relies heavily on IDE features (inline diffs, gutter annotations, hover previews), pi won't replace that workflow.

10.4 Steeper Learning Curve for Non-Terminal Users

Pi assumes comfort with: - Terminal workflows - Shell commands and Unix tools - TypeScript (for writing extensions) - Managing API keys and provider configuration

There's no guided onboarding, no wizard, no "just works" setup. You need to configure providers, understand the session model, and learn the commands.

10.5 Extension Quality Varies

Community extensions are maintained by individuals. Unlike Claude Code or Amp where features are tested by a company:

- Extensions may break on pi updates
- No formal review process
- Documentation quality varies
- Some extensions are experimental

Always test extensions in a non-critical project first.

10.6 No Built-in Eval/Testing Framework

Unlike Mastra (which has `runEvals` and scorers), pi has no formal way to test agent behavior or tool quality. Testing is manual -- you use the extension and see if it works. For teams that need CI-ready verification of agent behavior, this is a gap.

10.7 Windows Support is Second-Class

From r/ClaudeCode:

> I recommend ditching windows. Saved me my sanity. Agents are constantly doing command syntax mistakes on windows.

Pi works on Windows but the experience is rougher. Linux and macOS are the primary targets.

10.8 No Server-Side Session Management

Sessions are local JSONL files. There's no cloud sync, no team-shared session history, no server-side thread management like Amp. For distributed teams, this means:

- Sessions live on individual machines
- Sharing requires `/export` to HTML or `/share` to GitHub gist
- No real-time collaboration on sessions

10.9 No Native Long-Term Memory

Pi doesn't have built-in vector search, RAG, or cross-session memory. The alternatives:

- `AGENTS.md` for project-level persistent instructions
- `memory-mode` extension to save learnings to AGENTS.md
- Build a RAG extension (the hooks exist, but you build it yourself)
- Session branching partially compensates (you can revisit old decisions)

From Ewald Benes:

> Pi's AGENTS.md for project rules, combined with its session branching, handles 90% of my "memory" needs without the overhead of a separate manager.

10.10 Small (but Growing) Ecosystem

Pi has 1.2M+ weekly downloads and an active Discord, but the extension ecosystem is still small compared to VS Code or even Claude Code's community. You may need to build extensions yourself rather than finding pre-built ones.

10.11 The "Un-Google-able" Name

Mario chose the name deliberately to avoid users and issues. It worked. Searching for "pi coding agent" returns results about Raspberry Pi, the number pi, and various unrelated projects. Always search for "pi-coding-agent" or "shittycodingagent" or "badlogic/pi-mono".

10.12 No Cloud Workspace

Pi is entirely local. There's no cloud-hosted workspace, no web UI you can open from any browser, no always-on server-side sessions. Claude Code has a web interface. Amp's threads live on their servers. Pi's sessions live on your machine.

This means: - You can't start a session on your laptop and continue it from your phone's browser - There's no team dashboard showing active sessions - No "always running" agent that works while your machine is off

Pi does run on Android via Termux (see 10-quickstart.md), and you can copy your `~/.pi/agent/` config between machines. But there's no sync layer -- it's manual file transfer. For teams that want cloud-native agent workflows, this is a real gap compared to Amp or Claude Code's web offerings.

11. Quick Start and Daily Workflows

11.1 Installation

```
npm install -g @mariozechner/pi-coding-agent
```

Or use a standalone binary (no Node.js required) -- check the GitHub releases.

11.2 Android (Termux) Setup

Pi runs on Android via Termux with zero friction:

1. Install Termux from F-Droid
2. Install Node.js: `pkg install nodejs`
3. Install pi: `npm install -g @mariozechner/pi-coding-agent`
4. Copy just your `~/.pi/agent/models.json` from your desktop
5. Run `pi`

That's it -- `models.json` is the only file you need to bring over. Extensions, skills, and packages can all be installed via CLI afterwards (`pi install npm:shitty-extensions`, etc.). This is one of pi's underrated strengths: because everything is npm packages and local files, getting productive on a new device takes under a minute.

11.3 Authentication

```
# Option 1: API key
export ANTHROPIC_API_KEY=sk-ant-...
pi

# Option 2: OAuth subscription (Claude Pro/Max, ChatGPT Plus, Copilot, Gemini)
pi
/login
```

11.4 First Session

```
cd your-project
pi
```

Pi will load any `AGENTS.md` files from `~/.pi/agent/`, parent directories, and the current directory.

11.5 Essential Commands

Command	Action
`Ctrl+L`	Switch model
`Ctrl+P`	Cycle favorite models
`Shift+Tab`	Cycle thinking level
`/tree`	Navigate session tree
`/fork`	Branch to new session
`/compact`	Summarize old context
`/name <label>`	Label current session
`/export`	Export to HTML
`/share`	Upload to GitHub gist
`/reload`	Hot-reload extensions
`Ctrl+C`	Clear editor
`Ctrl+C` x2	Quit
`Escape`	Cancel/abort
`Escape` x2	Open `/tree`
`Ctrl+O`	Collapse/expand tool output

11.6 Editor Features

Feature	How
File reference	Type `@` to fuzzy-search project files
Path completion	Tab
Multi-line	Shift+Enter
Paste images	Ctrl+V
Run bash inline	`!command` (sends output to LLM)
Run bash silent	`!!command` (runs without sending)

11.7 Message Queuing

While the agent is working:

- `Enter` -- Send steering message (interrupts after current tool)
- `Alt+Enter` -- Send follow-up (waits until agent finishes)
- `Escape` -- Abort and restore queued messages

11.8 Recommended Setup

11.8.1 1. Global AGENTS.md

Create `~/.pi/agent/AGENTS.md` with your universal preferences:

```
# Global Agent Instructions

## Style
- Use TypeScript strict mode
- Prefer functional patterns
- Keep functions under 30 lines

## Tools
- Use pnpm for package management
- Use vitest for testing
```

11.8.2 2. Project AGENTS.md

Create `AGENTS.md` in your project root:

```
# Project: My App

## Stack
- Next.js 15, TypeScript, Tailwind
- Deployed on Vercel

## Conventions
- Components in src/components/
- API routes in src/app/api/
- Use server actions for mutations
```

11.8.3 3. Install Community Extensions

```
# Oracle + handoff + plan mode + more
pi install npm:shitty-extensions

# Model switching with aliases
pi install npm:pi-model-switch
```

11.8.4 4. Configure Model Aliases

Create `~/.pi/agent/extensions/model-switch/aliases.json`:

```
{
  "cheap": "google/gemini-2.5-flash",
  "coding": "anthropic/claude-sonnet-4-20250514",
  "review": "openai/gpt-5.2"
}
```

11.9 Daily Workflow Patterns

11.9.1 Focused Task

```
pi
> "Implement the user authentication flow using NextAuth.js"
# Agent works autonomously until done
```

11.9.2 Exploration + Branch

```
pi
> "Explore two approaches for the caching layer"
# Agent explores approach A
/tree    # Go back to before approach A
> "Now try approach B using Redis instead"
# Compare both branches
```

11.9.3 Multi-Model Brainstorm

```
> "Design the API schema for the notification service"
# Claude designs it
> "Switch to review model"
> "Review the schema design, find edge cases"
# GPT-5.2 reviews
> "Switch to coding model"
> "Implement the schema based on the review feedback"
```

11.9.4 Handoff Between Sessions

```
> "Plan the migration from REST to GraphQL"
# Agent creates the plan
/handoff implement phase 1 of the migration plan
# New session starts with focused context
```

11.9.5 Resume Previous Work

```
pi -c          # Continue most recent session
pi -r          # Browse all sessions, pick one
```

12. Code Snippets Reference

Concrete code examples referenced throughout the handbook.

12.1 Extension: Permission Gate

Block dangerous bash commands with user confirmation:

```
// ~/.pi/agent/extensions/permission-gate.ts
import type { ExtensionAPI } from "@mariozechner/pi-coding-agent";

const DANGEROUS_PATTERNS = ["rm -rf", "sudo", "DROP TABLE", "format", "mkfs"];

export default function (pi: ExtensionAPI) {
  pi.on("tool_call", async (event, ctx) => {
    if (event.toolName !== "bash") return;

    const cmd = event.input.command || "";
    const match = DANGEROUS_PATTERNS.find((p) => cmd.includes(p));

    if (match) {
      const ok = await ctx.ui.confirm(
        "Dangerous Command",
        `Command contains "${match}":\n${cmd}\n\nAllow?`
      );
      if (!ok) return { block: true, reason: `Blocked: contains ${match}` };
    }
  });
}
```

12.2 Extension: Oracle (Simplified)

Consult a second model for review and brainstorming:

```
// ~/.pi/agent/extensions/oracle.ts
import type { ExtensionAPI } from "@mariozechner/pi-coding-agent";
import { Type } from "@sinclair/typebox";
import { getModel, stream } from "@mariozechner/pi-ai";

export default function (pi: ExtensionAPI) {
  pi.registerTool({
    name: "oracle",
    label: "Oracle",
    description: `Consult a second model for review, debugging, or brainstorming.
      The oracle analyzes and advises but doesn't make changes.`,
    parameters: Type.Object({
      question: Type.String({ description: "What to ask the oracle" }),
      context: Type.String({ description: "Relevant code or situation summary" }),
    }),
    async execute(toolCallId, params, signal, onUpdate, ctx) {
      const oracle = getModel("openai", "gpt-5.2");

      const response = await stream(
        oracle,
        {
          system: `You are a senior engineering oracle. You review, analyze,
            and advise. You do NOT write code directly.`,
          messages: [
            {
              role: "user",
              content: `${params.context}\n\n${params.question}`,
            },
          ],
        },
        { apiKey: process.env.OPENAI_API_KEY! }
      );

      return {
        content: [{ type: "text", text: await response.text }],
        details: { model: "gpt-5.2", role: "oracle" },
      };
    },
  });
}
```

12.3 Extension: Session Recall

Read and summarize another session (cross-session reference):

```
// ~/.pi/agent/extensions/recall.ts
import type { ExtensionAPI } from "@mariozechner/pi-coding-agent";
import { readFileSync, readdirSync } from "node:fs";
import { join } from "node:path";
import { homedir } from "node:os";

export default function (pi: ExtensionAPI) {
  pi.registerCommand("recall", {
    description: "Load summary of another session by ID",
    handler: async (sessionId, ctx) => {
      if (!sessionId) {
        ctx.ui.notify("Usage: /recall <session-id>", "error");
        return;
      }

      const sessionsDir = join(homedir(), ".pi", "agent", "sessions");
      // Search all project directories for the session
      const projects = readdirSync(sessionsDir);

      for (const project of projects) {
        const projectDir = join(sessionsDir, project);
        const files = readdirSync(projectDir).filter((f) =>
          f.startsWith(sessionId)
        );

        if (files.length > 0) {
          const content = readFileSync(join(projectDir, files[0]), "utf-8");
          const lines = content.split("\n").filter(Boolean);
          const messages = lines
            .map((l) => JSON.parse(l))
            .filter(
              (e) => e.type === "message" && e.message?.role === "assistant"
            )
            .slice(-5); // Last 5 assistant messages

          const summary = messages
            .map((m) =>
              m.message.content
                .filter((c: any) => c.type === "text")
                .map((c: any) => c.text)
                .join("")
            )
            .join("\n---\n");

          ctx.ui.notify(
            `Loaded session ${sessionId} from ${project}`,
            "success"
          );
          // Inject as context for next turn
          return { inject: summary };
        }
      }

      ctx.ui.notify(`Session ${sessionId} not found`, "error");
    },
  });
}
```

12.4 Skill: Web Search (CLI-based)

A skill file that teaches the agent to use a CLI search tool:

```
<!-- .pi/skills/web-search/SKILL.md -->
# Web Search

Search the web and fetch page content using CLI tools.

## Tools

### search
Search the web: `search "your query" --max-results 5`
Returns: title, URL, snippet for each result.

### fetch
Read a web page: `fetch <url> --format text --max-chars 5000`
Returns: page content as plain text.

## Usage Pattern
1. Search for information: `search "topic"`
2. Read promising results: `fetch <url>`
3. Synthesize findings for the user

## Notes
```

```
- Prefer specific queries over broad ones
- Fetch only the pages that look relevant from search results
- Summarize rather than dumping raw content
```

12.5 AGENTS.md Template

```
# AGENTS.md

## Project
- Name: [project name]
- Stack: [e.g., Next.js 15, TypeScript, Tailwind, Drizzle ORM]
- Deployed on: [e.g., Vercel, Cloudflare Workers]

## Conventions
- Use pnpm for package management
- Use strict TypeScript
- Components in src/components/
- Keep functions under 30 lines
- Write tests for business logic

## Model Preferences
- Simple file ops / quick questions: switch to "cheap"
- Complex refactoring / architecture: switch to "coding"
- Code review: switch to "review"
- Default to budget-friendly models unless quality is needed

## Do Not
- Commit credentials or .env files
- Modify files in node_modules/
- Run destructive commands without confirmation
```

12.6 Custom Provider Configuration

```
// ~/.pi/agent/models.json
{
  "providers": [
    {
      "id": "my-ollama",
      "name": "Local Ollama",
      "api": "openai-completions",
      "baseUrl": "http://localhost:11434/v1",
      "models": [
        {
          "id": "llama-3.1-70b",
          "name": "Llama 3.1 70B",
          "contextWindow": 128000,
          "maxTokens": 32000,
          "reasoning": false,
          "input": ["text"]
        }
      ]
    }
  ]
}
```

13. Official References

Direct links to pi's official documentation, source code, and community resources.

13.1 Core Documentation

Document	URL
README (main docs)	https://github.com/badlogic/pi-mono/blob/main/packages/coding-agent/README.md
Extensions guide	https://github.com/badlogic/pi-mono/blob/main/packages/coding-agent/docs/extensions.md
Skills guide	https://github.com/badlogic/pi-mono/blob/main/packages/coding-agent/docs/skills.md
Session format	https://github.com/badlogic/pi-mono/blob/main/packages/coding-agent/docs/session.md
Packages guide	https://github.com/badlogic/pi-mono/blob/main/packages/coding-agent/docs/packages.md
Themes guide	https://github.com/badlogic/pi-mono/blob/main/packages/coding-agent/docs/themes.md
Prompt templates	https://github.com/badlogic/pi-mono/blob/main/packages/coding-agent/docs/prompt-templates.md
Keybindings	https://github.com/badlogic/pi-mono/blob/main/packages/coding-agent/docs/keybindings.md
Custom providers	https://github.com/badlogic/pi-mono/blob/main/packages/coding-agent/docs/custom-provider.md
Adding models	https://github.com/badlogic/pi-mono/blob/main/packages/coding-agent/docs/models.md
RPC mode	https://github.com/badlogic/pi-mono/blob/main/packages/coding-agent/docs/rpc.md
SDK usage	https://github.com/badlogic/pi-mono/blob/main/packages/coding-agent/docs/sdk.md
TUI components	https://github.com/badlogic/pi-mono/blob/main/packages/coding-agent/docs/tui.md

13.2 Extension Examples (50+)

https://github.com/badlogic/pi-mono/tree/main/packages/coding-agent/examples/extensions

Notable examples in the repo: - `summarize.ts` -- Conversation summaries - `snake.ts` -- Snake game (TUI demo) - `async-subagent/` -- Sub-agent implementation - Various permission gate patterns

13.3 Source Code

Package	Path
pi-ai (LLM API)	https://github.com/badlogic/pi-mono/tree/main/packages/ai
pi-agent-core	https://github.com/badlogic/pi-mono/tree/main/packages/agent
pi-coding-agent	https://github.com/badlogic/pi-mono/tree/main/packages/coding-agent
pi-tui	https://github.com/badlogic/pi-mono/tree/main/packages/tui
pi-web-ui	https://github.com/badlogic/pi-mono/tree/main/packages/web-ui
pi-mom (Slack bot)	https://github.com/badlogic/pi-mono/tree/main/packages/mom
pi-pods (vLLM)	https://github.com/badlogic/pi-mono/tree/main/packages/pods

13.4 Community Packages

Package	npm	GitHub
shitty-extensions	`npm:shitty-extensions`	https://github.com/hjanuschka/shitty-extensions
pi-model-switch	`npm:pi-model-switch`	https://github.com/nicobailon/pi-model-switch
pi-subagent-enhanced	--	https://github.com/nicobailon/pi-subagent-enhanced
@marckrenn/pi-sub-core	`npm:@marckrenn/pi-sub-core`	https://github.com/marckrenn/pi-sub
pi-skills (official)	`git:github.com/badlogic/pi-skills`	https://github.com/badlogic/pi-skills
pi-coding-agent (Emacs)	MELPA	https://github.com/dnouri/pi-coding-agent

13.5 CLI Tools (Mario's agent-tools)

https://github.com/badlogic/agent-tools

CLI tools designed for agent use with READMEs the agent reads on demand.

13.6 Blog Posts and Articles

Title	Author	URL
What I learned building an opinionated and minimal coding agent	Mario Zechner	https://mariozechner.at/posts/2025-11-30-pi-coding-agent/
MCP vs CLI: Benchmarking Tools for Coding Agents	Mario Zechner	https://mariozechner.at/posts/2025-08-15-mcp-vs-cli/
Pi: The Minimal Agent Within OpenClaw	Armin Ronacher	https://lucumr.pocoo.org/2026/1/31/pi/
Why I Switched to Pi	Helmut Januschka	https://www.januschka.com/pi-coding-agent.html
The Only Coding Agent You'll Ever Need	Ewald Benes	https://ewaldbenes.com/en/blog/the-only-coding-agent-you-ll-ever-need
An Emacs mode for a shitty coding agent	Daniel Nouri	https://danielnouri.org/notes/2025/12/30/an-emacs-mode-for-a-shitty-coding-agent/
Your MCP Doesn't Need 30 Tools: It Needs Code	Armin Ronacher	https://lucumr.pocoo.org/2025/8/18/code-mcps/
Replace MCP With CLI	Cobus Greyling	https://cobusgreyling.substack.com/p/replace-mcp-with-cli-the-best-ai

13.7 Community Discussions

Thread	Platform
Change your coding agent to pi	https://www.reddit.com/r/ClaudeCode/comments/1qu5fa4/
Why I switched from Claude Code to Pi	https://www.reddit.com/r/ClaudeCode/comments/1r11egp/
HN discussion on Mario's blog post	https://news.ycombinator.com/item?id=46844822
pi-coding-agent in Emacs	https://www.reddit.com/r/emacs/comments/1qa8xql/

13.8 DeepWiki (AI-generated docs from source)

https://deepwiki.com/badlogic/pi-mono/4-@mariozechnerpi-coding-agent

14. Contributing

This handbook is a community project. No contribution is too small.

14.1 What we're looking for

- Typo fixes, grammar improvements, broken links
- Better code examples or updated snippets
- New sections, tips, or workflow patterns you've discovered
- Extensions or skills you've built — write them up, share the code
- Corrections to anything that's wrong or outdated
- Screenshots, terminal recordings, or diagrams
- Translations

The Limitations page documents known gaps. Building solutions for those and writing them up here is the highest-impact work you can do.

14.2 How to contribute

1. Fork sparticle9/pi-handbook
2. Edit or add markdown files in `docs/`
3. Open a PR with a short description of what you changed and why

That's it. No build step required — the site deploys automatically on merge.

14.3 Guidelines

- Write in plain, direct English. No fluff, no hype.
- Be honest about limitations. Don't oversell pi or undersell alternatives.
- All quotes must be attributed with source links.
- Keep code snippets runnable and minimal.
- Use MkDocs Material admonitions for callouts:

```
!!! tip
    This is a tip.

!!! warning
    This is a warning.
```

- New chapters go in `docs/` and must be added to `nav:` in `mkdocs.yml`.
- When comparing tools, be fair. State facts, cite sources.

14.4 Local preview

```
pip install mkdocs-material
mkdocs serve
# http://localhost:8000
```

14.5 What not to do

- Don't add vendor-specific promotional content
- Don't commit credentials or API keys
- Don't use AI-generated filler text
- Don't remove the disclaimer about unofficial status

14.6 License

Content is licensed under CC BY-SA 4.0. By contributing, you agree to license your work under the same terms.

15. Download

Get the full handbook as a single PDF for offline reading.

Download PDF PDF

The PDF is regenerated automatically on every push to `main`.