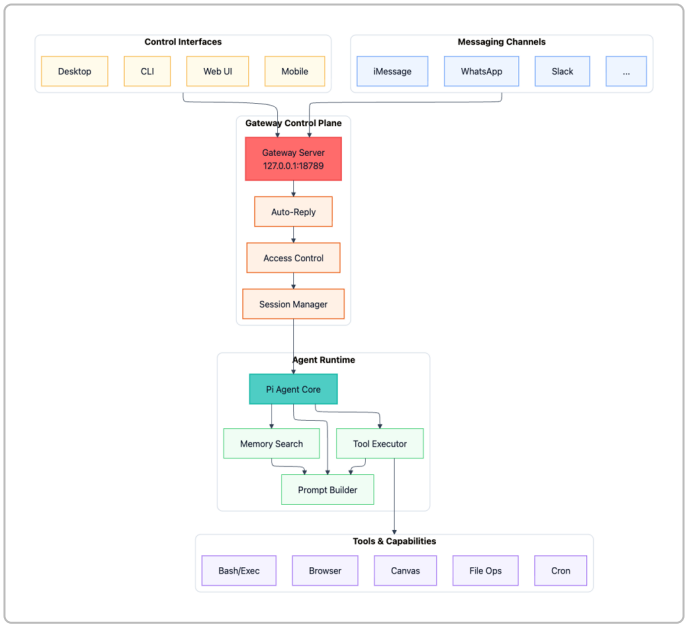
ChatGPT

OpenClaw Architecture and Code-Level Overview



OpenClaw is a **self-hosted gateway** for AI agents, connecting your chat apps (WhatsApp, Telegram, Discord, etc.) to an always-on assistant [1]. It follows a *hub-and-spoke* design centered on one **Gateway** process [2]. The Gateway (Node.js ≥22) is implemented in `src/gateway/server.ts` (using the `ws` WebSocket library) and binds by default to `127.0.0.1:18789` [3]. Inbound messages from any channel or UI are routed through the Gateway, which performs access control, dispatches to the correct session, and broadcasts events to clients [4] [2]. The **Agent Runtime** (in `src/agents/piembeddedrunner.ts`) uses the @mariozechner/pi-agent-core library. On each turn it resolves the session (e.g. DM vs. group), builds the system prompt (combining static files and recent history), streams tokens from the LLM, watches for tool-invocation requests, executes tools (e.g. bash, browser, file I/O, Canvas) in context, and persists state and memory [5] [2]. By separating interfaces from the agent loop, one assistant instance stays alive on your hardware and serves all channels.

- **Code Components:** OpenClaw is written in TypeScript/Node. The CLI entrypoint is `openclaw.mjs` (bootstrapped with Commander.js) and primary commands live in `src/cli/program.ts` [6]. For example, `openclaw gateway` starts the server, `openclaw agent` queries the agent directly, `openclaw channels login` pairs devices, and `openclaw message send` posts messages to channels [6] [7]. Internally, the Gateway is in `src/gateway/`, the Agent loop in `src/agents/piembeddedrunner.ts`, and built-in tools in `src/agents/pi-tools.ts` / `openclaw-tools.ts` [5] [8]. Communication adapters for each channel live under `src/telegram/`, `src/discord/`, `src/slack/`, `src/imessage/`, etc. Each adapter (using libraries like Baileys for WhatsApp, grammY for Telegram, discord.js, etc.) implements a common interface for auth, message parsing, access control, and sending responses [9].

- **Extensibility:** OpenClaw's architecture is plugin-driven [10]. In `extensions/` you can add new channel plugins (e.g. Mattermost), memory backends (vector stores vs. default SQLite), custom tools, or new model providers [10]. A plugin is declared in `package.json` under an

`openclaw.extensions` field and is loaded by `src/plugins/loader.ts`. For example, you can add a provider plugin so the Gateway calls a local or third-party LLM (Claude, OpenAI, etc.) instead of the bundled Pi model [10]. Internally all WebSocket messages are type-checked against JSON schemas (via TypeBox) and key operations require idempotency tokens, making the system robust and secure [11].

PicoClaw Architecture and Code-Level Overview

PicoClaw is an **ultra-lightweight AI agent** implemented in Go [12]. It was bootstrapped by an AI-driven rewrite of a Python prototype, resulting in a single **self-contained binary** (RISC-V/ARM/x86) with a memory footprint under 10 MB [13] [12]. It is optimized for microcontroller-class hardware ($10 dev boards): startup is ~1 s on a 0.6 GHz core [12] [13]. Despite its small size, PicoClaw offers "full assistant" capabilities: planning, logging (cron jobs), web search, automation, and even integrations with Telegram or Discord [14] [15]. The Go code orchestrates everything locally: it handles sessions, calls the configured LLM APIs (via HTTP), executes any built-in tools (web search, shell commands, etc.), and manages persistent memory.

- **Single-Binary Go Implementation:** All code lives under `cmd/picoclaw` and `pkg/`. Building for your platform just requires `make deps` (to fetch Go modules) and `make build` [16]. The result is one executable (no external server). Common agent logic (LLM calls, memory management, tool interfaces) is in Go packages under `pkg/`. PicoClaw embeds a minimal LLM loop (using HTTP to OpenRouter, Zhipu, Anthropic, etc.) and tool sets for search and cron.

- **Config & Workspace:** PicoClaw uses a JSON config in `~/.picoclaw/config.json` (copied from `config.example.json`). This defines your default model (e.g. `"glm-4.7"` or another provider) and provider credentials [17]. After running `picoclaw onboard`, it creates a workspace at `~/.picoclaw/workspace/` [18]. This workspace contains subfolders like `sessions/` (chat histories), `memory/` (long-term notes), `cron/` (scheduled jobs), and `skills/` (custom skill scripts), along with files `AGENTS.md` and `IDENTITY.md` that describe the agent's rules and persona [19] [20]. By default, each message thread or chat session is isolated (with its own session file) for continuity.

- **CLI Commands:** Interaction is via a CLI similar to OpenClaw's. Example: `picoclaw onboard` initializes your directory; `picoclaw gateway` starts listening on channels; and `picoclaw agent -m "text"` sends a message directly to the agent [21] [22]. After configuring channels (e.g. setting a Telegram bot token under `channels.telegram.token`), `picoclaw gateway` will proxy inbound messages to the agent loop and deliver replies back over that channel. Internal logs and session files update automatically.

Adapting to Moonshot's "QMK 2.5" (Kimi K2.5) Model

Both OpenClaw and PicoClaw support pluggable LLM backends. OpenClaw allows **custom provider plugins** [10]. In practice, you would configure it to use the Moonshot/Kimi K2.5 model instead of, say, Anthropic. For example, if Kimi is exposed via an API (Moonshot's platform or Hugging Face inference), you can add a provider entry in OpenClaw's config and supply your API key. PicoClaw's `config.json` similarly lets you set `"model": "<provider/K2.5>"` and add your API key under `"providers"`. (For example, using OpenRouter or a Moonshot gateway that supports Kimi.) In both cases, once the provider is added, the agent loops will call Kimi's API for completions. The underlying system doesn't change – it will stream tokens back into the agent runtime as usual. Key points for Claude to consider in

coding this integration:
- **OpenClaw:** Use the plugin mechanism or built-in providers. In `~/.openclaw/openclaw.json`, add a provider (e.g. `"moonshot": {...}`) and set the default model name to the K2.5 endpoint. If needed, write a small extension under `extensions/` that wraps the Kimi API. This makes OpenClaw direct its LLM calls to Kimi instead of Claude/GPT.
- **PicoClaw:** In `~/.picoclaw/config.json`, under `"providers"` include the Kimi API key (similarly to how the OpenRouter key is shown [17]). Set `"model": "kimi-k2-5"` (or the exact identifier) in the defaults. PicoClaw will then POST prompts to Kimi and use the JSON response. Because PicoClaw already supports long contexts (8k tokens+) and streaming, it should work with Kimi's output.
- **Compatibility:** Ensure any differences in tokenization or streaming protocols are handled. (If Kimi uses a different API format, adjust the code that reads tokens.) In summary, you *can* reuse most of OpenClaw/PicoClaw's code: just inject Kimi as the LLM provider.

Build & Run Instructions on macOS

- **Install runtimes:** On your Mac, install **Node.js ≥22** (e.g. via Homebrew: `brew install node`) and **Go (1.20+)**. Also install a package manager like `pnpm` or use `npm`.
- **OpenClaw setup:** Run `npm install -g openclaw@latest` [23]. Then execute `openclaw onboard --install-daemon` to create the gateway service. This sets up `~/.openclaw/openclaw.json`. Next, use `openclaw gateway` to start the server [7]. You may also run pairing commands (e.g. `openclaw channels login`) as needed.
- **PicoClaw build:** Clone the PicoClaw repo (`git clone https://github.com/sipeed/picoclaw.git`) and `cd picoclaw`. Run `make deps` then `make build` [16]. This produces a `picoclaw` binary. Optionally `make install` to put it in your PATH.
- **Configure providers:** Copy the example configs and insert your API keys. For OpenClaw, edit `~/.openclaw/openclaw.json` adding your Moonshot/Kimi key or setting a provider plugin. For PicoClaw, copy `config.example.json` to `~/.picoclaw/config.json` and fill in `"providers": { ... }` with your LLM API and (if used) a search API. (The medium walkthrough [15] and example config [17] show the JSON structure.)
- **Run the assistants:** Start the agents and gateways. For example:
- *OpenClaw:* `openclaw gateway --port 18789` to launch the control UI (see [11] and [7] for details).
- *PicoClaw:* Use `picoclaw onboard` once, then `picoclaw gateway` to listen on configured channels [21]. You can also test on the CLI: e.g. `picoclaw agent -m "Hello"` [22].
- **Dependencies for QMK:** If running Kimi K2.5 locally (instead of via an API), ensure your machine has enough resources and any needed libraries (e.g. OpenBLAS for Hugging Face inference). Otherwise, rely on Moonshot's free API access to K2.5 as indicated by the provider documentation.

With this setup, Claude can reuse much of the existing OpenClaw/PicoClaw code. The architecture diagrams, folder layouts, and CLI behaviors are documented and cited above. By plugging in the QMK (Kimi 2.5) model as shown, the system should run end-to-end with your local model environment.

Sources: Official OpenClaw docs and architecture guide [1] [2] [6]; OpenClaw code (gateway, CLI, agent) via Paolo's overview [3] [5]; PicoClaw repo and press materials [13] [12]; PicoClaw configuration examples [17] [20]. Each section above cites the relevant documentation.

---

[1] [23] OpenClaw - OpenClaw
https://docs.openclaw.ai

2 3 4 5 6 8 9 10 11 OpenClaw Architecture, Explained - by Paolo Perazzo
https://ppaolo.substack.com/p/openclaw-system-architecture-overview

7 GitHub - openclaw/openclaw: Your own personal AI assistant. Any OS. Any Platform. The lobster way.
https://github.com/openclaw/openclaw

12 14 Forget the Mac Mini: Run This OpenClaw Alternative for Just $10 - Hackster.io
https://www.hackster.io/news/forget-the-mac-mini-run-this-openclaw-alternative-for-just-10-da23b2819d25

13 16 17 18 19 20 21 22 GitHub - sipeed/picoclaw: picoclaw
https://github.com/sipeed/picoclaw

15 PicoClaw: The $10 AI Agent That Changed My Edge Computing Game | by Ishank choudhary | Feb, 2026 | Medium
https://medium.com/@ishank.iandroid/picoclaw-the-10-ai-agent-that-changed-my-edge-computing-game-5c2c0c6badfb