AI CODING AGENT & PERSONAL ASSISTANT

Build Your Own AI Coding Agent - 2-page cheat sheet

Course companion - architecture, tool contracts, memory policy, sandboxing, release gates.

Sovereignty is an architectural property

- **Threat model:** The LLM is an UNTRUSTED planner. Tools are PRIVILEGED services. Everything from repos, docs, inboxes, the web, and user pastes is UNTRUSTED input.

- **Prompt is a request; gate is the law:** You can write 'never access the network' in the system prompt - and should. But the guarantee comes from authorize() in policy code denying the tool regardless of what the model was told or tricked into.

- **Quiz-grade facts:** Missing when a local app can't export memory in a documented format: Portability. Open-weight != unrestricted commercial use - the license controls that.

- **Offline smoke test:** Most useful offline test: run a representative task with the network disconnected after dependencies are cached.

Inference lanes (1.3)

- **Lanes + contract:** Ollama (default), vLLM, llama.cpp server mode - interchangeable behind one interface. The app only ever talks to ModelClient: chat / stream / embed / health, OpenAI-compatible schema. Never a vendor API.

- **Adapter + templates:** Why the adapter: stable application contract, testable replacement. Apply each model's chat template exactly (<|im_start|>...); budget tokens = prompt + tool schemas + transcript + answer headroom.

ReAct as a bounded state machine (2.3)

```
load state -> build context -> model decision
  |- final answer -> verify -> finish
  |- tool request -> validate -> authorize
       -> execute -> record observation
  repeat action? STOP (partial + evidence)
  budget exhausted? STOP (partial + evidence)
```

- **Decision record:** What the model asked, what you allowed, what happened, why the loop continued or stopped. The trace is the product. Never depend on the model's chain-of-thought.

Tool calling = structured prediction (2.2)

- **Mental model:** Model emits JSON like {"name": "read_file", "arguments": {"path": "notes.md"}}. Your code validates against JSON Schema, checks policy, runs with timeout, bounds the result, returns a typed envelope.

- **Validate first:** Schema catches the most common model failure - plausible-looking garbage - before anything executes. Cheapest defense you have.

```
class ToolResult(BaseModel):
    ok: bool
    summary: str   # becomes the observation
    data: dict = {}
    error_code: str | None = None
```

The model proposes; the application disposes.

Prompts for agents, not chat (2.4)

- **Separate policy from task:** Invariant POLICY (workspace, tools, evidence rules, stop conditions) vs per-run TASK instructions. Never mix - downstream content is untrusted and the model can't tell operator instructions from instruction-shaped data.

- **Mechanisms:** Define tools with use-conditions; pair every prompt rule with an enforcing mechanism.

Coding agent tools (Section 3)

- **list_tree:** Bounded listing: depth + entry caps, skip .git / __pycache__ / node_modules / .venv.

- **read_file:** Line ranges, not whole files. Returns the slice + SHA-256 of the full file - the hash is the precondition that lets writes reject stale edits.

- **search_text:** Regex with match windows. Windows over wholes - long contexts make models dumber and poorer.

- **Safe writes:** propose patch -> policy check (extensions, max lines) -> apply -> show diff -> run tests. Stale-hash rejection on mismatch.

Planning & self-correction (Section 4)

- **Earned planning:** 'First, I'll make a plan' is the most expensive sentence in agent engineering. Plan only when the task earns it; a plan is an executable contract (steps, preconditions, evidence, stop rules), not optimism.

- **Failure taxonomy:** Classify from EVIDENCE (error_code, exit codes, stderr), then apply the rule:

- **1 tool error:** fix invocation, same intent.

- **2 environment:** retry 1-2x; the action wasn't wrong, the world was.

- **3 wrong hypothesis:** new hypothesis from the new evidence.

- **4 invalid patch:** regenerate against fresh file state.

- **5 test failure:** read the output, update hypothesis, THEN act.

- **6 policy denial:** stop, replan within policy or escalate. Never retry harder.

Repo understanding = retrieval + compression (Sec 5)

- **Map + chunks:** RepoMap: hash, language, size, symbols, imports, last-indexed commit. Symbol-aware chunking; persist chunks with full metadata.

- **Local embeddings:** sentence-transformers locally (default all-MiniLM-L6-v2, 384 dims, CPU-fast). Batch 32-64. Record the model ID per collection. Cache weights offline. Prove no request leaves the machine.

Never ship your codebase to an embeddings API. Non-negotiable.

Memory: explicit & reversible (Section 6)

- **DB leads:** PostgreSQL = source of truth; everything else is a cache. Persist the state transition BEFORE the side effect - the DB survives the crash, so the DB leads. Transactional outbox for queue messages.

- **Write policy (1/2):** explicit_request -> PERSIST (0.95+); stable_preference -> PERSIST if clear else ASK; task_commitment -> PERSIST with due date.

- **Write policy (2/2):** transient ('near Denver TODAY') -> NEVER (read temporal scope!); sensitive -> ASK FIRST; inferred -> ASK/CONFIRM, never silent-persist. Inferred medical facts: NEVER store.

Voice, product, sandbox, production

- **Voice (Sec 7):** Voice is a latency budget: capture -> VAD -> STT -> decide -> TTS -> play; time each stage. faster-whisper + Piper (Coqui alt). No cloud speech APIs. Every proactive act visible, bounded, cancellable, auditable.

- **Product (Sec 8):** FastAPI + HTMX control plane: HTTP at the edge, a RUN as the unit of long work, one correlation ID per request. MinIO for artifacts, not DB blobs.

- **Sandbox (Sec 9):** Non-root, read-only fs, dropped caps, seccomp, cgroup/PID limits, NO network, NO docker socket, NO host secrets. Prove each with adversarial tests (e.g. code curling 169.254.169.254).

- **Production (Sec 10):** Telemetry: identifiers/timings/sizes/error codes only - never raw prompts or audio. Quantization: benchmark on TASK evals (4-bit fine for summarization, cliff on tool-use). Release gate: unit -> tool-contract -> coding evals -> RAG tests -> canary.

- **Capstone (Sec 11):** Freeze scope before building. Demo on a schedule. Package the portfolio: architecture review, demo, honest write-up of what you'd do differently.