03 — Context Delivery

🕒 state of the art 2026-07 · revised 2026-07-25 · 📖 ~11 min read

Learning objectives

By the end of this chapter, you should be able to:

1. **Explain** why context is a budget managed at runtime, not a warehouse (and what *context rot* is);
2. **Compose** a system prompt in layers ordered by volatility (cache-aware);
3. **Design** a cascade of context files (global → project → package → personal) with declared precedence;
4. **Implement** harness-zero's context assembler (step 3) with a project rules file;
5. **Evaluate** a real AGENTS.md file against the authoring practices (lean, executable commands, grown by evidence of failure).

The problem

The model only knows what the harness shows it. "Context delivery" is the engineering of deciding **what** goes into each call — system prompt, project rules, environment state, memories, instructions from external servers — **in what order**, and **how that changes** mid-conversation without breaking the provider's cache or confusing the model.

The classic sub-problems: where project rules live and how they are discovered; whether the system prompt should vary by model; how to communicate state changes mid-conversation without invalidating the cached prefix.

Scientific foundations

- **Context degrades with position and with volume** — *Lost in the Middle* (arXiv 2307.03172): information in the middle of long contexts is poorly used. Design consequence: what matters goes to the edges (system prompt at the start; the current task at the end), and "send everything" is an anti-pattern with empirical backing.
- **Context engineering as a discipline** — the survey arXiv 2507.13334 systematizes the area (RAG, memory, tool-integrated reasoning) and legitimizes the term the industry adopted.
- **Less context, better agents** — arXiv 2606.10209 measures in long-running agents what Anthropic calls context rot: aggressive curation beats full windows.

(Full bibliography: `livro/bibliografia.md`.)

Industry sources

- **Effective context engineering for AI agents** (Anthropic Engineering): names the successor to prompt engineering — the job is to **curate the optimal set of tokens at inference time**; names *context rot* as an engineering fact. Decision: the window is a budget, and the goal is the smallest set of high-signal tokens.
- **Prompt caching** (official docs) + **Lessons from building Claude Code: prompt caching is everything**: caching is **by prefix** — the context assembly order is a cost decision. The Claude Code account lists the classic invalidators (a timestamp at the top, a request ID in the tool list, history reserialization) and treats **cache hit rate as a first-class harness metric** (~59% reduction in billable input).
- **AGENTS.md** + **Agentic AI Foundation**: the "README for agents" was **donated to the Linux Foundation (Dec 2025)** with OpenAI, Anthropic and Block as co-founders; 60k+ projects. Decision: per-repository file context has become neutral, portable infrastructure — investing in that pipeline is safe.
- **How Claude remembers your project** (docs): formalizes the global → project → local **cascade**, with the closest file winning and the personal one kept out of version control.
- **AGENTS.md Field Guide 2026** (practitioner): authoring — start with ~30 lines, cap at ~150–200 at the root, exact commands before prose, nest per package in a monorepo, and **grow only on evidence of the agent's recurring failure** (the common mistake is treating it as documentation).
- **See also**: the living collection Awesome Harness Engineering — Context Delivery & Compaction gathers more resources for this dimension (patterns, articles and implementations), curated by problem.

The state of the art

1. Context is a managed budget — and retrieval went just-in-time

The modern consensus inverted the "the more context, the better" instinct: the harness actively manages the window (rule-based pruning, awareness of how much remains, on-demand retrieval). The benchmark's two most advanced materializations: Aider's **repo-map** (the model "sees" the structure of an entire repository within a ~1k-token budget, via tree-sitter + personalized PageRank — static just-in-time retrieval with no explorer agent at all) and Goose's **incremental per-subdirectory hints** (rules loaded as the agent navigates, not all upfront).

2. Prefix stability became an architectural requirement

Cache-awareness stopped being an optimization and reorganized context assembly: layers ordered by volatility, deterministic serialization, zero volatile content at the top. The most rigorous formalizations measured: opencode's **Context Epochs** (the prefix as an immutable cache baseline, with state changes delivered only at safe turn boundaries) and Hermes's **explicit three-layer prompt** (`stable` → `context` → `volatile`, declaredly designed to maximize prefix-cache — including in the skill-curation fork, which inherits the parent's prefix to save ~26%).

3. The rules file standardized — and became a cascade

The AGENTS/CLAUDE/GEMINI.md fragmentation of the discipline's early days is resolved by neutral governance (Linux Foundation): AGENTS.md is the portable format, read natively by Codex, Goose, opencode, OpenClaw, Hermes, Aider and dozens of others, with the proprietary names becoming aliases. The mature pattern is the **cascade with declared precedence** (global → project → package → personal; the closest wins; the personal one gitignored), `@imports` for composition (gemini-cli) and — the authoring practice that separates useful files from dead documentation — growing **on evidence of failure**, like code.

4. The new frontiers

Three recent moves that have not yet become consensus: **per-model-family prompts** (opencode with ~10 variants; Codex taking it to the extreme with **server-driven** instructions — the backend delivers the per-model base prompt, with even a configurable "personality"); **persona × rules separation** (the personal-agent category's contribution: `SOUL.md` for voice/identity separate from the operational `AGENTS.md` — OpenClaw, Hermes, ohmo); and **trust-classed context** (IronClaw: personal/injected content travels in "prompt envelopes" with the trust class preserved — context delivery meeting the security of ch. 07).

> **The counterpoint: the minimal harness (Pi)** — *addendum from round ext-1, 2026-07-31*. While this chapter describes ever-richer context assemblers, Pi (Earendil/Zechner, ~54k stars) bets in the opposite direction: a base system prompt **measured at ~460 tokens**, derived from the tool set (each tool contributes its snippet; guidelines enter only if the corresponding tool is active), and skills announced **by name+description only** — the body is loaded by the model itself via `read` when the task calls for it (progressive disclosure taken to the limit: there is not even a skill tool). Editorial honesty demands the two caveats the code reading revealed: (1) the same assembler concatenates the cascade's `AGENTS.md` files **with no budget** — in Pi's own repo this adds ~2,700 tokens, six times the slogan; the minimality is the harness's, not the context's; (2) minimalism is not absence of engineering — Pi's compaction is the most complete in the corpus (see the evaluation, in Portuguese). The underlying bet is falsifiable and worth tracking: **better models would need less harness** — if true, part of this chapter expires; if the window stays expensive, the missing budget charges interest. It is the control experiment the corpus was missing.

EXECUTIVE SUMMARY

What is most modern: budget + just-in-time (not volume), stable prefix as a requirement (with cache hit rate as an SLI), cascading AGENTS.md under neutral governance, and the three frontiers (per-model/server-driven prompts, separate persona, trust class). The minimalist counterpoint (Pi, round ext-1) shows the other end of the spectrum: a ~460-token prompt derived from the tool set — and proves the budget×richness tension remains open. **What to steal:** the repo-map as a cheap alternative to exploration; Hermes's 3 layers by volatility; the "grows on recurring failure" discipline in AGENTS.md authoring; from Pi, the prompt snippet coupled to the tool definition (prompt and tool set never desynchronize).

Hands-on — harness-zero, step 3

In step 3 you build harness-zero's context assembler: a system prompt in layers ordered by volatility (identity → environment → project rules → memory → task), discovery of an `AGENTS.md` at the target project's root, and a test that proves **prefix stability** across two consecutive turns (same bytes up to the last message). Completion exercise: the cascade discovery function comes skeletonized; you implement the precedence.

Check your understanding

1. Why is a timestamp at the top of the system prompt expensive — and where should it live? (Prefix caching + mid-conversation updates.)
2. Your agent ignores a project convention repeatedly. What is the right response according to modern authoring practice — and what is the wrong one? (Adding the rule to AGENTS.md on evidence × dumping documentation.)
3. A harness wants to tell the model the date has changed in the middle of a long conversation. Describe two strategies with different cache costs. (Epochs/turn boundaries × rewriting the prefix.)

Appendix A — How each repository handles context delivery

> Per-harness evidence, with paths — online supplement, expanded with each benchmark round.

opencode (round 1) — typed algebra and Context Epochs

`packages/opencode/src/session/system.ts` assembles environment + skills + MCP (Model Context Protocol) instructions; **~10 prompts per model family** in `session/prompt/*.txt` (anthropic, gpt, codex, gemini, kimi, beast...), selected by model-id substring; global/ancestor `AGENTS.md` aggregated by `session/instruction.ts`. V2 (`CONTEXT.md`) formalizes context as an algebra of "Context Sources" with snapshots, **Context Epochs** (cache baseline) and mid-conversation system messages only at safe boundaries.

gemini-cli (round 1) — hierarchy with @imports

`prompts/promptProvider.ts` assembles by mode/tools/model (modern × legacy snippets); hierarchical `GEMINI.md` (`memoryDiscovery.ts`: global → parents → subfolders) with `@imports` (`memoryImportProcessor.ts`) and `flattenMemory`; full override via `GEMINI_SYSTEM_MD`; just-in-time injection (`tools/jit-context.ts`).

OpenHarness (round 1) — aggregation with relevant memory

`src/openharness/prompts/context.py`: base + environment + `CLAUDE.md` + **memories selected by relevance** (`memory/relevance.py`, with `usage.py` tracking usage) + skills + active repo context; `-s/--append-system-prompt` on the CLI.

Codex CLI (round 2) — central AGENTS.md + server-driven prompts

`core/src/agents_md.rs`: hierarchical discovery with merge from project-root to cwd; the system prompt **varies by model and comes from the backend** (`ModelInfo.base_instructions` via `models-manager`, with a template and `Personality::Friendly/Pragmatic`); environmental context via `WorldState`.

Goose (round 2) — incremental hints and hardening

`SystemPromptBuilder` with override + extras; multi-file hints (**`.goosehints` AND `AGENTS.md`**, `CLAUDE.md` via config) respecting `.gitignore`; **`SubdirectoryHintTracker`** loads subdirectory hints as the agent navigates; anti prompt-injection sanitization of Unicode tags; per-turn "top of mind".

Aider (round 2) — the repo-map ⭐

`aider/repomap.py`: definition/reference tags via tree-sitter (per-language `.scm` queries) → file→file graph → **personalized PageRank** (chat files and mentioned idents bias the ranking; ×10/×50/×0.1 multipliers) → rendering under budget with binary search (~1024 tokens; `map_mul_no_files=8` with no files in the chat) → mtime-keyed cache. The entire context-first path in one file.

OpenHands/Canvas (round 2) — organizational skills

`app_conversation/skill_loader.py`: skills auto-discovered from the conventional repositories **`owner/.openhands` and `owner/.agents`** across all the user's organizations (GitHub/GitLab/Azure), with KeywordTrigger/TaskTrigger and a marketplace — team context versioned and loaded for all members.

OpenClaw (round 2) — identity workspace with budgets

`buildAgentSystemPrompt` injects `SOUL.md` (persona), `AGENTS.md` (rules), `USER.md`, `IDENTITY.md`, `TOOLS.md`, `MEMORY.md`, `HEARTBEAT.md`, `BOOTSTRAP.md` — with budgets (20k chars/file, 60k total) and marked truncation; provider-aware contributions **above/below the cache boundary**.

Hermes (round 2) — three layers by volatility ⭐

`agent/system_prompt.py` + `prompt_builder.py`: `stable` (identity/SOUL.md + guidance + skill index) → `context` (the project's AGENTS.md/.cursorrules) → `volatile` (memory, USER.md, timestamp) — explicit design for prefix-cache; persona migratable from OpenClaw.

IronClaw (round 2) — context as a policy decision

`LoopPromptPort` (crates/ironclaw_loop_host): resolves identity, personal context (**opt-in per run profile, not per channel**), skills and security; injected/personal content travels in **prompt envelopes** with an unforgeable trust class — separating what the loop requests from what the host allows it to see.

ohmo (round 2.5) — the minimal correct version

`ohmo/prompts.py`: ordered concatenation base → soul → identity → user → BOOTSTRAP → workspace → memory; the rigorous decision `include_project_memory=False` (the personal agent does not read a project's CLAUDE.md — tested).

Pi (round ext-1) — the prompt derived from the tool set ⭐

`core/system-prompt.ts`: base **measured at ~460 tokens**, assembled from the tool definitions' own `promptSnippet`s with dedup and guidelines conditional on the active set (deactivate the tool, the prompt shrinks); skills announced only as `<name/description/location>` and loaded by the model via `read` (block omitted if `read` is not active); `AGENTS.md`/`CLAUDE.md` cascade global → root → cwd with dedup of nested worktrees (`resource-loader.ts`) — yet concatenated **with no budget** (see the box in the chapter body); full override via `.pi/SYSTEM.md`.

n8n (round 2) — the embedded minimum

`ToolsAgent/common.ts`: `ChatPromptTemplate` with a free-form system message + history + rich binaries (images/PDF); no rules file and no hierarchy — the context comes mapped from the workflow by its author.

Frameworks (frameworks round) — open by design

LangGraph and the Agents SDK (Software Development Kit) leave assembly to the dev (static or callable instructions); CrewAI imposes role/goal/backstory as structural context; the software-agent-sdk provides a Jinja preset with a documented escape hatch (`prompt_dir` + `_prompt_preset() -> None`).