03 — Entrega de Contexto

🕒 estado da arte 2026-07 · revisão 2026-07-25 · 📖 ~11 min de leitura

Objetivos de aprendizagem

Ao final deste capítulo, você deve ser capaz de:

1. **Explicar** por que contexto é um orçamento gerenciado em runtime, não um depósito (e o que é *context rot*);
2. **Compor** um system prompt em camadas ordenadas por volatilidade (cache-aware);
3. **Projetar** uma cascata de arquivos de contexto (global → projeto → pacote → pessoal) com precedência declarada;
4. **Implementar** o montador de contexto do harness-zero (etapa 3) com um arquivo de regras de projeto;
5. **Avaliar** um arquivo AGENTS.md real contra as práticas de autoria (enxuto, comandos executáveis, crescido por evidência de falha).

O problema

O modelo só sabe o que o harness mostra. "Entrega de contexto" é a engenharia de decidir **o que** entra em cada chamada — system prompt, regras do projeto, estado do ambiente, memórias, instruções de servidores externos — **em que ordem**, e **como isso muda** no meio de uma conversa sem quebrar o cache do provedor nem confundir o modelo.

Sub-problemas clássicos: onde vivem as regras do projeto e como são descobertas; se o prompt de sistema deve variar por modelo; como informar mudanças de estado mid-conversation sem invalidar o prefixo cacheado.

Fundamentos científicos

- **Contexto degrada com posição e com volume** — *Lost in the Middle* (arXiv 2307.03172): a informação no meio de contextos longos é mal utilizada. Consequência de projeto: o que importa vai para as bordas (system prompt no início; a tarefa atual no fim), e "mandar tudo" é anti-padrão com base empírica.
- **Context engineering como disciplina** — o survey arXiv 2507.13334 sistematiza a área (RAG, memória, tool-integrated reasoning) e legitima o termo que a indústria adotou.
- **Menos contexto, agentes melhores** — arXiv 2606.10209 mede em agentes de longa duração o que a Anthropic chama de context rot: curadoria agressiva supera janelas cheias.

(Bibliografia completa: `livro/bibliografia.md`.)

Fontes da indústria

- **Effective context engineering for AI agents** (Anthropic Engineering): batiza a sucessão da prompt engineering — o trabalho é **curar o conjunto ótimo de tokens em tempo de inferência**; nomeia *context rot* como fato de engenharia. Decisão: a janela é orçamento, e a meta é o menor conjunto de tokens de alto sinal.
- **Prompt caching** (docs oficiais) + **Lessons from building Claude Code: prompt caching is everything**: cache é **por prefixo** — a ordem de montagem do contexto é decisão de custo. O relato do Claude Code lista os invalidadores clássicos (timestamp no topo, request ID na lista de tools, reserialização do histórico) e trata **cache hit rate como métrica de primeira classe do harness** (~59% de redução de input billable).
- **AGENTS.md** + **Agentic AI Foundation**: o "README para agentes" foi **doado à Linux Foundation (dez/2025)** com OpenAI, Anthropic e Block como co-fundadores; 60k+ projetos. Decisão: contexto por arquivo de repositório virou infraestrutura neutra e portável — investir nesse pipeline é seguro.
- **How Claude remembers your project** (docs): formaliza a **cascata** global → projeto → local, com o arquivo mais próximo vencendo e o pessoal fora do versionamento.
- **AGENTS.md Field Guide 2026** (praticante): autoria — começar com ~30 linhas, teto ~150–200 na raiz, comandos exatos antes de prosa, aninhar por pacote em monorepo, e **crescer só por evidência de falha reincidente do agente** (o erro comum é tratá-lo como documentação).
- **Consulte também**: a coleção viva Awesome Harness Engineering — Context Delivery & Compaction reúne mais recursos consultáveis desta dimensão (padrões, artigos e implementações), curados por problema.

O estado da arte

1. Contexto é orçamento gerenciado — e a recuperação virou just-in-time

O consenso moderno inverteu o instinto de "quanto mais contexto, melhor": o harness administra ativamente a janela (poda por regra, awareness de quanto resta, recuperação sob demanda). As duas materializações mais avançadas do benchmark: o **repo-map** do Aider (o modelo "enxerga" a estrutura de um repositório inteiro num orçamento de ~1k tokens, via tree-sitter + PageRank personalizado — recuperação estática just-in-time sem nenhum agente explorador) e os **hints incrementais por subdiretório** do Goose (regras carregadas conforme o agente navega, não tudo de antemão).

2. Estabilidade de prefixo virou requisito arquitetural

Cache-awareness deixou de ser otimização e reorganizou a montagem do contexto: camadas ordenadas por volatilidade, serialização determinística, zero conteúdo volátil no topo. As formalizações mais rigorosas medidas: os **Context Epochs** do opencode (o prefixo como baseline imutável de cache, com mudanças de estado entregues só em fronteiras seguras de turno) e o **prompt em três camadas explícitas** do Hermes (`stable` → `context` → `volatile`, desenhado declaradamente para maximizar prefix-cache — inclusive no fork de curadoria de skills, que herda o prefixo do pai para economizar ~26%).

3. O arquivo de regras padronizou — e virou cascata

A fragmentação AGENTS/CLAUDE/GEMINI.md do início da disciplina está resolvida por governança neutra (Linux Foundation): AGENTS.md é o formato portável, lido nativamente por Codex, Goose, opencode, OpenClaw, Hermes, Aider e dezenas de outros, com os nomes proprietários virando alias. O padrão maduro é a **cascata com precedência declarada** (global → projeto → pacote → pessoal; o mais próximo vence; o pessoal gitignored), `@imports` para composição (gemini-cli) e — a prática de autoria que separa arquivos úteis de documentação morta — crescer **por evidência de falha**, como código.

4. As fronteiras novas

Três movimentos recentes que ainda não viraram consenso: **prompt por família de modelo** (opencode com ~10 variantes; Codex levando ao extremo com instruções **server-driven** — o backend entrega o prompt-base por modelo, com até "personalidade" configurável); **separação persona × regras** (a contribuição da categoria de agentes pessoais: `SOUL.md` para voz/identidade separado do `AGENTS.md` operacional — OpenClaw, Hermes, ohmo); e **contexto com classe de confiança** (IronClaw: conteúdo pessoal/injetado viaja em "prompt envelopes" com trust class preservada — a entrega de contexto encontrando a segurança do cap. 07).

> **O contraponto: o harness mínimo (Pi)** — *adendo da rodada ext-1, 2026-07-31.* Enquanto este capítulo descreve montadores de contexto cada vez mais ricos, o Pi (Earendil/Zechner, ~54k estrelas) aposta na direção oposta: system prompt base **medido em ~460 tokens**, derivado do tool set (cada ferramenta contribui seu snippet; guidelines entram só se a ferramenta correspondente está ativa), e skills anunciadas **só por nome+descrição** — o corpo é carregado pelo próprio modelo via `read` quando a tarefa pede (a divulgação progressiva levada ao limite: nem tool de skill existe). A honestidade editorial exige as duas ressalvas que a leitura de código revelou: (1) o mesmo montador concatena os `AGENTS.md` da cascata **sem orçamento** — no próprio repo do Pi isso adiciona ~2.700 tokens, seis vezes o slogan; a minimalidade é do harness, não do contexto; (2) o minimalismo não é ausência de engenharia — a compactação do Pi é a mais completa do corpus (ver avaliação). A aposta subjacente é falsificável e vale acompanhar: **modelos melhores precisariam de menos harness** — se for verdade, parte deste capítulo expira; se a janela continuar cara, a falta de orçamento cobra juros. É o experimento de controle que faltava ao corpus.

LEITURA EXECUTIVA

O que está mais moderno: orçamento + just-in-time (não volume), prefixo estável como requisito (com cache hit rate como SLI), AGENTS.md em cascata sob governança neutra, e as três fronteiras (prompt por modelo/server-driven, persona separada, trust class). O contraponto minimalista (Pi, rodada ext-1) mostra o outro extremo do espectro: prompt de ~460 tokens derivado do tool set — e prova que a tensão orçamento×riqueza segue aberta. **O que roubar:** repo-map como alternativa barata à exploração; as 3 camadas por volatilidade do Hermes; a disciplina "cresce por falha reincidente" na autoria de AGENTS.md; do Pi, o snippet de prompt acoplado à definição da ferramenta (prompt e tool set nunca dessincronizam).

Mão na massa — harness-zero, etapa 3

Na etapa 3 você constrói o montador de contexto do harness-zero: system prompt em camadas ordenadas por volatilidade (identidade → ambiente → regras do projeto → memória → tarefa), descoberta de um `AGENTS.md` na raiz do projeto-alvo, e um teste que prova a **estabilidade do prefixo** entre dois turnos consecutivos (mesmos bytes até a última mensagem). Exercício de completude: a função de descoberta em cascata vem esqueletada; você implementa a precedência.

Verificação

1. Por que um timestamp no topo do system prompt é caro — e onde ele deveria ficar? (Cache por prefixo + mid-conversation updates.)
2. Seu agente ignora uma convenção do projeto de forma reincidente. Qual é a resposta certa segundo a prática de autoria moderna — e qual é a errada? (Adicionar a regra ao AGENTS.md por evidência × despejar documentação.)
3. Um harness quer informar ao modelo que a data mudou no meio de uma conversa longa. Descreva duas estratégias com custos de cache diferentes. (Epochs/fronteiras de turno × reescrever o prefixo.)

Apêndice A — Como cada repositório trata a entrega de contexto

> Evidência por harness, com paths — complementação online, expandida a cada rodada do benchmark.

opencode (rodada 1) — álgebra tipada e Context Epochs

`packages/opencode/src/session/system.ts` monta environment + skills + instruções MCP (Model Context Protocol); **~10 prompts por família de modelo** em `session/prompt/*.txt` (anthropic, gpt, codex, gemini, kimi, beast...), selecionados por substring do model id; `AGENTS.md` globais/ascendentes agregados por `session/instruction.ts`. A V2 (`CONTEXT.md`) formaliza o contexto como álgebra de "Context Sources" com snapshots, **Context Epochs** (baseline de cache) e mensagens de sistema mid-conversation só em fronteiras seguras.

gemini-cli (rodada 1) — hierarquia com @imports

`prompts/promptProvider.ts` monta por modo/tools/modelo (snippets modernos × legados); `GEMINI.md` hierárquico (`memoryDiscovery.ts`: global → pais → subpastas) com `@imports` (`memoryImportProcessor.ts`) e `flattenMemory`; override total via `GEMINI_SYSTEM_MD`; injeção just-in-time (`tools/jit-context.ts`).

OpenHarness (rodada 1) — agregação com memória relevante

`src/openharness/prompts/context.py`: base + ambiente + `CLAUDE.md` + **memórias selecionadas por relevância** (`memory/relevance.py`, com `usage.py` rastreando uso) + skills + contexto de repo ativo; `-s/--append-system-prompt` na CLI.

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

`core/src/agents_md.rs`: descoberta hierárquica com merge do project-root ao cwd; system prompt **varia por modelo e vem do backend** (`ModelInfo.base_instructions` via `models-manager`, com template e `Personality::Friendly/Pragmatic`); contexto ambiental via `WorldState`.

Goose (rodada 2) — hints incrementais e hardening

`SystemPromptBuilder` com override + extras; hints multi-arquivo (**`.goosehints` E `AGENTS.md`**, `CLAUDE.md` via config) respeitando `.gitignore`; **`SubdirectoryHintTracker`** carrega hints de subdiretório conforme o agente navega; sanitização anti prompt-injection de tags Unicode; "top of mind" por turno.

Aider (rodada 2) — o repo-map ⭐

`aider/repomap.py`: tags de definição/referência via tree-sitter (queries `.scm` por linguagem) → grafo arquivo→arquivo → **PageRank personalizado** (chat files e idents mencionados enviesam o ranking; multiplicadores ×10/×50/×0.1) → renderização sob orçamento com busca binária (~1024 tokens; `map_mul_no_files=8` sem arquivos no chat) → cache por mtime. O caminho context-first inteiro em um arquivo.

OpenHands/Canvas (rodada 2) — skills organizacionais

`app_conversation/skill_loader.py`: skills auto-descobertas de repositórios convencionais **`owner/.openhands`** e **`owner/.agents`** em todas as organizações do usuário (GitHub/GitLab/Azure), com KeywordTrigger/TaskTrigger e marketplace — contexto de time versionado e carregado para todos os membros.

OpenClaw (rodada 2) — workspace de identidade com orçamentos

`buildAgentSystemPrompt` injeta `SOUL.md` (persona), `AGENTS.md` (regras), `USER.md`, `IDENTITY.md`, `TOOLS.md`, `MEMORY.md`, `HEARTBEAT.md`, `BOOTSTRAP.md` — com orçamentos (20k chars/arquivo, 60k total) e truncamento marcado; contribuições provider-aware **acima/abaixo do cache boundary**.

Hermes (rodada 2) — três camadas por volatilidade ⭐

`agent/system_prompt.py` + `prompt_builder.py`: `stable` (identidade/SOUL.md + guidance + índice de skills) → `context` (AGENTS.md/.cursorrules do projeto) → `volatile` (memória, USER.md, timestamp) — desenho explícito para prefix-cache; persona migrável do OpenClaw.

IronClaw (rodada 2) — contexto como decisão de política

`LoopPromptPort` (crates/ironclaw_loop_host): resolve identidade, contexto pessoal (**opt-in por run profile, não por canal**), skills e segurança; conteúdo injetado/pessoal viaja em **prompt envelopes** com trust class inforjável — separação entre o que o loop pede e o que o host permite ver.

ohmo (rodada 2.5) — a versão mínima correta

`ohmo/prompts.py`: concatenação ordenada base → soul → identity → user → BOOTSTRAP → workspace → memória; decisão rigorosa `include_project_memory=False` (o agente pessoal não lê CLAUDE.md de projeto — testado).

Pi (rodada ext-1) — o prompt derivado do tool set ⭐

`core/system-prompt.ts`: base **medida em ~460 tokens**, montada dos `promptSnippet` das próprias tool definitions com dedup e guidelines condicionais ao conjunto ativo (desativou a tool, o prompt encolhe); skills anunciadas só como `<name/description/location>` e carregadas pelo modelo via `read` (bloco omitido se `read` não está ativa); cascata `AGENTS.md`/`CLAUDE.md` global→raiz→cwd com dedup de worktrees aninhadas (`resource-loader.ts`) — porém concatenada **sem orçamento** (ver caixa no corpo do capítulo); override total via `.pi/SYSTEM.md`.

n8n (rodada 2) — o mínimo do embutido

`ToolsAgent/common.ts`: `ChatPromptTemplate` com system message livre + histórico + binários ricos (imagens/PDF); sem arquivo de regras nem hierarquia — o contexto vem mapeado do workflow pelo autor.

Frameworks (rodada frameworks) — aberto por design

LangGraph e Agents SDK (Software Development Kit) deixam a montagem por conta do dev (instructions estáticas ou callable); CrewAI impõe role/goal/backstory como contexto estrutural; o software-agent-sdk dá preset Jinja com escape hatch documentado (`prompt_dir` + `_prompt_preset() -> None`).