16 — Aprendizado e Auto-melhoria

🕒 estado da arte 2026-07 · revisão 2026-07-28 · 📖 ~8 min de leitura

Objetivos de aprendizagem

Ao final deste capítulo, você deve ser capaz de:

1. **Explicar** por que o aprendizado auto-evolutivo quebra o pressuposto do *scaffolding* estático — e como ele inverte a cláusula de expiração do livro;
2. **Descrever** as etapas do ciclo fechado de captura de skills (gatilho, curadoria, isolamento, formato portável, reencontro indexado, manutenção contra a entropia);
3. **Comparar** os dois designs concorrentes de aplicação do aprendizado — autônoma × promoção humana — e **localizar** um harness real na escada de maturidade da dimensão;
4. **Avaliar** os riscos da dimensão (superstição, entropia, contaminação, prompt injection como aprendizado permanente) e as engenharias que os previnem.

O problema

As doze dimensões dos capítulos 02–13 descrevem *scaffolding* (andaime) *estático*: alguém — o autor do harness, o usuário, um plugin — escreve as instruções, tools e políticas, e o agente as consome. Este capítulo documenta a dimensão emergente que quebra esse pressuposto: o agente que **escreve o próprio scaffolding** — capturando procedimentos aprendidos como skills reutilizáveis.

A dimensão foi promovida a suplementar do template do benchmark (dimensão 13) por força de uma evidência: o **Hermes Agent** (Nous Research) implementa o ciclo completo, e a leitura do código confirma cada etapa (Apêndice A).

O estado da arte

O ciclo fechado: as seis etapas

O mecanismo de referência, verificado no código do Hermes (evidência detalhada no Apêndice A), fecha o ciclo em seis etapas:

1. **Gatilho autônomo** — a revisão de aprendizado dispara sozinha, em background, sem o usuário pedir (com gatilho manual como complemento).
2. **Curadoria por um fork isolado** — um clone do agente, com um prompt curatorial que define o que capturar e — o mais importante — **anti-padrões do que NÃO aprender**. Sem essa lista, o sistema degeneraria em superstição acumulada.
3. **Isolamento do meta-trabalho** — o fork curador tem tools restritas e persistência desligada, para não contaminar a sessão real.

4. **Escrita em formato portável** — a skill vira um `SKILL.md` sob standards rígidos, com a restrição de contexto moldando o formato do conhecimento.
5. **Reencontro barato** — índice compacto sempre no system prompt; conteúdo integral só entra no contexto sob demanda. Aprendizado indexado, não despejado.
6. **Manutenção contra a entropia** — um curador periódico consolida, arquiva por inatividade e protege o que está fixado. Memória que só cresce vira ruído; o curador é o coletor de lixo do conhecimento.

A escada de maturidade na coorte avaliada

Harness	Nota 13	O que tem
Hermes	**3**	O ciclo fechado completo (Apêndice A), com aplicação autônoma
gemini-cli	**3** (retro)	Auto Memory: agente extrator com gates anti-ruído ("Default to NO SKILL", 5 perguntas de bloqueio) produzindo SKILL.md + patches de memória — mas com **promoção humana via inbox** (`/memory inbox`); dedupe, sandbox de escrita, evals dedicados
IronClaw	2	Extração automática de skills (`learning.rs`) com métricas de uso/confiança e versionamento
OpenClaw	1	Dreaming (consolidação autônoma de memória); Skill Workshop com fila de propostas
OpenHarness	1 (retro)	Auto-extração de fatos por turno, com staleness por uso (60 dias) — fatos, não procedimentos
Codex CLI	1	Memórias automáticas com pruning (fatos, não procedimentos)
Goose	1	chatrecall (recall semântico de conversas passadas)
opencode, demais	0 (retro)	Skills são consumo/distribuição; nada é escrito pela experiência

A escada é nítida: **memória de fatos** (nível 1) → **extração de procedimentos** (nível 2) → **ciclo curado com anti-padrões e manutenção** (nível 3). O que separa o nível 3 não é capturar mais — é a engenharia de *não* capturar errado e de podar o que envelheceu.

Os dois designs concorrentes do nível 3

O nível 3 já tem **dois designs concorrentes**, com a divergência exatamente onde importa: *quem aplica o que foi aprendido*. O Hermes aplica autonomamente (com o curador limpando depois); o gemini-cli exige promoção humana (inbox — nada entra no contexto sem `/memory inbox`). É o trade-off clássico autonomia × controle do capítulo 07, reaparecendo na dimensão mais nova: o Hermes aposta que anti-padrões bastam para prevenir aprendizado ruim; o gemini-cli aposta que não. As próximas rodadas dirão qual escala melhor.

Por que isso muda a tese do livro

A cláusula de expiração (cap. 01, 14) diz: todo componente de harness é uma prótese para uma limitação atual do modelo, e expira quando o modelo melhora. O aprendizado auto-evolutivo **inverte a cláusula**: em vez de esperar o modelo dispensar o scaffolding, o par modelo+harness *escreve scaffolding novo para si mesmo*. Cada

skill aprendida é um pedaço de harness gerado em runtime, específico ao usuário e ao ambiente — algo que nenhum autor de harness poderia ter escrito de fábrica.

Isso cria uma terceira via na taxonomia:

1. **Scaffolding de fábrica** — escrito pelo autor do harness; expira com a evolução dos modelos.
2. **Scaffolding de fronteira** — sandbox, permissões, interfaces; não expira (é sobre o mundo).
3. **Scaffolding auto-gerado** — skills escritas pelo agente; *cresce* com o uso, e sua qualidade depende da engenharia de curadoria, não da capacidade bruta do modelo.

Os riscos: o espelho das promessas

Os riscos são o espelho das promessas: sem anti-padrões, superstição; sem curadoria, entropia; sem isolamento do meta-trabalho, contaminação; e — apontado pela avaliação do IronClaw (prompt-write safety; cf. cap. 07) — sem fronteira de escrita protegida, **prompt injection vira aprendizado permanente**: um atacante que convence o agente a "aprender" uma skill maliciosa persiste na memória procedural. A dimensão 13 madura exigirá a dimensão 6 madura.

LEITURA EXECUTIVA

A dimensão é a mais nova do template e a menos convergida: dois harnesses no nível 3 com designs opostos sobre quem aplica o aprendizado, e o resto da coorte entre memória de fatos e nada. O que já é consenso de engenharia entre os que chegaram lá: a peça central não é o mecanismo de captura, e sim os **anti-padrões do que não aprender** e a **manutenção** (consolidar, arquivar, nunca deletar). **O que roubar** hoje: lista de anti-padrões no prompt curatorial; isolamento do meta-trabalho em fork sem persistência; índice compacto com conteúdo sob demanda; curador periódico como coletor de lixo; fronteira de escrita protegida contra prompt injection.

Reavaliação retroativa da coorte de código pendente; a dimensão sai de "suplementar" quando ≥3 harnesses atingirem nível 2+.

> **Consulte também**: a coleção viva Awesome Harness Engineering — Skills & MCP reúne mais recursos consultáveis desta dimensão, curados por problema.

Verificação

1. Por que a lista de **anti-padrões** ("o que NÃO aprender") é descrita como a peça central da engenharia curatorial, e não o mecanismo de captura em si? O que acontece com um sistema que captura sem ela?
2. Localize na escada de maturidade um harness que extrai fatos automaticamente com staleness por uso, mas não captura procedimentos. Que nota ele recebe, e o que faltaria para subir um nível?

3. Hermes e gemini-cli estão ambos no nível 3, mas divergem em *quem aplica* o que foi aprendido. Reconstrua o trade-off autonomia × controle nesse contexto: qual é a aposta de cada design?
4. Explique a frase "a dimensão 13 madura exigirá a dimensão 6 madura": por que prompt injection é qualitativamente mais grave num harness que aprende do que num harness estático?

Apêndice A — Hermes Agent

> Evidência por repositório, com paths — material de complementação (versão online), expandido a cada rodada do benchmark. Avaliação completa: `../../benchmark/avaliacoes/hermes-agent.md`.

O ciclo fechado do Hermes (evidência: `agent/background_review.py` e afins)

O mecanismo, verificado no código do fork avaliado:

1. Gatilho autônomo. A cada ~10 iterações de tool-calling (`skill_nudge_interval`, em `agent/turn_finalizer.py`), o harness dispara uma revisão em background — sem o usuário pedir. Há também o gatilho manual `/learn`.

2. Curadoria por um fork isolado. Um clone do agente roda em thread separada com o snapshot da conversa e um prompt curatorial (`_SKILL_REVIEW_PROMPT`) que é a peça central da engenharia. Ele instrui o curador a ser ativo ("um passe que não faz nada é aprendizado perdido"), define ordem de preferência (atualizar skill existente > criar nova; skills novas só class-level, nunca "fix-bug-1234") e — o mais importante — lista **anti-padrões do que NÃO aprender**: falhas dependentes de ambiente, claims negativos sobre tools ("o browser não funciona"), erros transitórios, narrativas one-off. Sem essa lista, o sistema degeneraria em superstição acumulada.

3. Isolamento do meta-trabalho. O fork tem whitelist de tools restrita (`memory` + `skills`), memória e persistência desligadas — para a curadoria não contaminar a sessão real — e herda o prefixo de prompt cacheado do pai (redução de ~26% no custo da revisão).

4. Escrita em formato portável. A skill vira um `SKILL.md` compatível com **agentskills.io** em `~/.hermes/skills/<categoria>/<nome>/` (com `references/`, `templates/`, `scripts/`), sob standards rígidos — descrição ≤60 caracteres *porque o índice no system prompt trunca em 60*: a restrição de contexto moldando o formato do conhecimento.

5. Reencontro barato. O índice compacto (nome + descrição) está sempre no system prompt; o conteúdo integral só entra no contexto quando o agente chama `skill_view` — aprendizado indexado, não despejado.

6. Manutenção contra a entropia. Um **curador** periódico (`agent/curator.py`) roda quando o agente está ocioso: consolida skills em umbrellas, arquiva por inatividade (90 dias — arquivar, nunca deletar), protege skills fixadas. Memória que só cresce vira ruído; o curador é o coletor de lixo do conhecimento.