

Agent Harness: Yang Membuat AI Agent Bisa Dipercaya

Agent Harness: Yang Membuat AI Agent Bisa Dipercaya

Penulis: Aan Setianto, Passionate Entrepreneur | AI Enthusiast.

Sep 26, 2026.

[aansetianto.com](#)

Corporate Website: [vascomm.co.id](#)

© 2026 Aan Setianto. Hak cipta dilindungi.

Edisi 1.0 · Aan Setianto

Contents

Part 1 • Memahami harness	5
Ringkasan	6
Dari tanya-jawab ke tugas	7
Pengguna makin ambisius	8
Siklus kerja agent	9
Apa itu agent harness?	10
Contoh agent harness	11
Kenapa agent butuh harness	12
Model Anthropic terbaru	13
Apa yang bisa salah di tugas panjang	14
Peta masalah ke komponen	15
Part 2 • Delapan komponen	16
Delapan komponen harness	17
Agent state	18
Tools	19
Planning	20
Memory	21
Orchestration	22
Context management	23
Safety & governance	24
Observability & evaluation	25

Part 3 • Contoh dan arah	26
Contoh lengkap	27
Contoh bug #122, langkah demi langkah	28
Evaluasi	29
Arah ke depan	30
Part 4 • Praktik dan penutup	31
Checklist merancang harness sendiri	32
Glosarium	33
Rekap	34
About Author	35

BAGIAN 1

1

Memahami harness

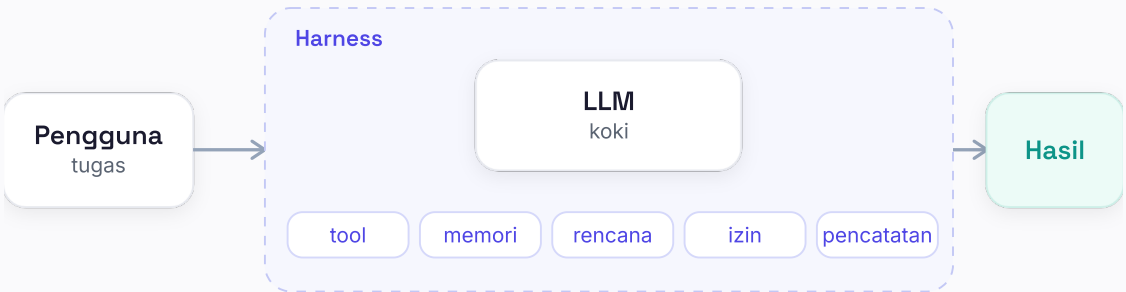
Apa itu harness dan kenapa ia makin dibutuhkan.

DI BAGIAN INI

- | | |
|---|------------------------------|
| 01 Ringkasan | 02 Dari tanya-jawab ke tugas |
| 03 Pengguna makin ambisius | 04 Siklus kerja agent |
| 05 Apa itu agent harness? | 06 Contoh agent harness |
| 07 Kenapa agent butuh harness | 08 Model Anthropic terbaru |
| 09 Apa yang bisa salah di tugas panjang | 10 Peta masalah ke komponen |

Ringkasan

Model AI dan lapisan di sekelilingnya



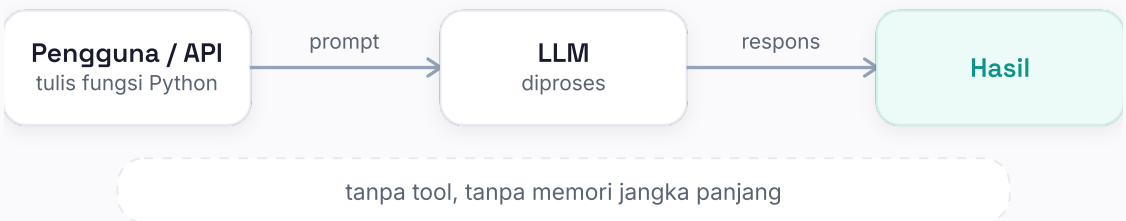
Model AI tidak bekerja sendirian. Di sekelilingnya ada tool, memori, rencana, aturan izin, dan pencatatan. Gabungan semua itu disebut **agent harness**.

Analoginya seperti dapur restoran. Koki adalah model AI. Pesanan yang menempel di papan adalah tugas dari pengguna. Pisau, kompor, dan kulkas adalah tool. Buku resep, catatan stok, aturan kebersihan, dan kepala dapur yang memeriksa hasil piring adalah harness.

Koki paling jago pun tidak bisa melayani 200 pesanan dalam semalam tanpa semua itu.

Dari tanya-jawab ke tugas

LLM tahun 2023 dan agent tahun 2026



Pada 2023, kebanyakan orang memakai LLM (large language model) dengan cara sederhana: bertanya, lalu menerima jawaban. Misalnya seorang programmer mengetik, **"Tulis fungsi Python untuk mengurutkan angka."** Jawabannya muncul hampir seketika, tanpa fase "berpikir".

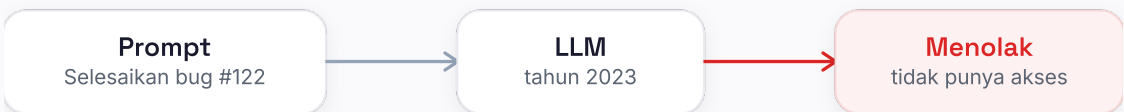
Model saat itu polos. Ia tidak punya tool, tidak punya memori jangka panjang, dan tidak bisa membuka apa pun di luar percakapan.

Itu sama seperti tanya-jawab di warung. Anda bertanya harga nasi goreng, pelayan menjawab, selesai. Pelayan tidak pergi ke dapur, tidak mengecek stok, tidak mencatat pesanan.

Pengguna makin ambisius

Prompt berubah menjadi tugas

2023



2026



Model makin canggih, dan permintaan pengguna ikut naik. Orang berhenti sekadar bertanya. Mereka mulai menyerahkan **tugas**: cari data di browser, ingat percakapan minggu lalu, susun rencana.

Sekarang prompt yang umum digunakan bisa berbunyi, **"Selesaikan bug di issue #122 di GitHub."** Kalimat pendek itu menyimpan banyak langkah: membuka repositori, membaca issue, mengubah kode, menjalankan tes.

Sedangkan LLM tahun 2023 hanya bisa menjawab, **"Maaf, saya tidak punya akses ke repositori Anda."**

Siklus kerja agent

Bernalar, rencana, tool, amati

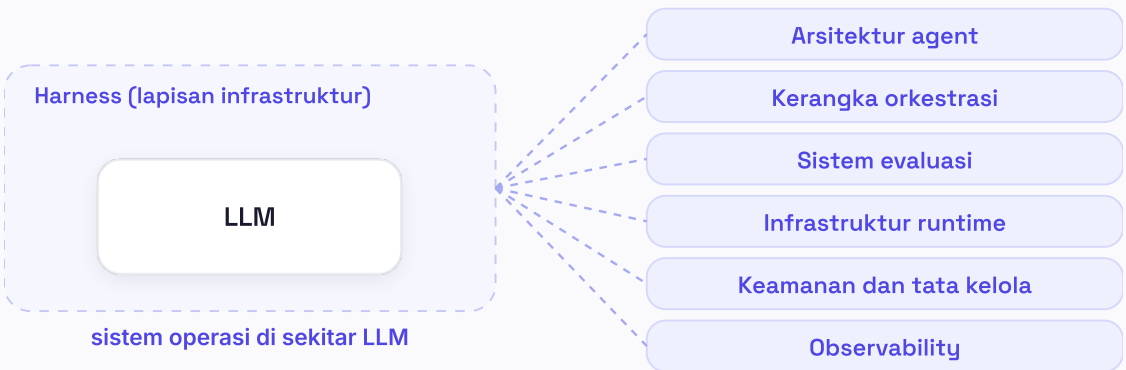


LLM yang dilengkapi harness yang baik, akan masuk ke mode berpikir dan bekerja dalam beberapa putaran. Putaran ini berulang sampai tujuan tercapai atau ada alasan untuk berhenti. Sistem seperti inilah yang disebut **AI agent**.

Analoginya seperti koki menerima pesanan nasi goreng. Ia membaca tiket (bernalar), menyusun urutan kerja (merencanakan), mengambil wajan (memilih tool), memasak (menjalankan), mencicipi (mengamati), lalu memutuskan apakah perlu garam tambahan atau piringnya sudah boleh keluar.

Apa itu agent harness?

Sistem operasi di sekitar LLM



Semua tambahan di sekeliling LLM, yaitu konteks, tool, memori, dan lainnya, perlu diatur supaya bekerja rapi bersama model. Cara mengatur dan menjaganya disebut **agent harness**: lapisan infrastruktur yang membungkus, mengendalikan, menguji, memantau, dan mengevaluasi sebuah AI agent.

Dalam bahasa sehari-hari, harness adalah "sistem operasi" di sekitar LLM. Sistem operasi tidak melakukan pekerjaan Anda, tetapi ia menentukan program mana yang boleh jalan dan apa yang terjadi kalau ada yang macet.

Contoh agent harness

Yang dipakai sekarang

NAMA	PEMBUAT	YANG PERLU ANDA TAHU
Claude Code	Anthropic	Hanya untuk model Anthropic
Claude Agent SDK	Anthropic	Membangun agent sendiri di atas Claude Code
Claude Managed Agents	Anthropic	Layanan hosting untuk agent yang bekerja lama
Codex	OpenAI	Hanya jalan dengan model OpenAI
OpenCode	Anomaly	Lebih dari 75 penyedia model, termasuk lokal
Goose	Agentic AI Foundation	Desktop, CLI, dan API; lebih dari 15 penyedia
OpenHands	All-Hands-AI	Bisa di-host sendiri, sandbox Docker
Zed Agent	Zed Industries	Menyatu dengan editor Zed
Qwen Code	QwenLM (Alibaba)	Untuk model Qwen, lewat API atau lokal
Pydantic AI Harness	Pydantic	Pustaka Python untuk layanan yang sudah ada

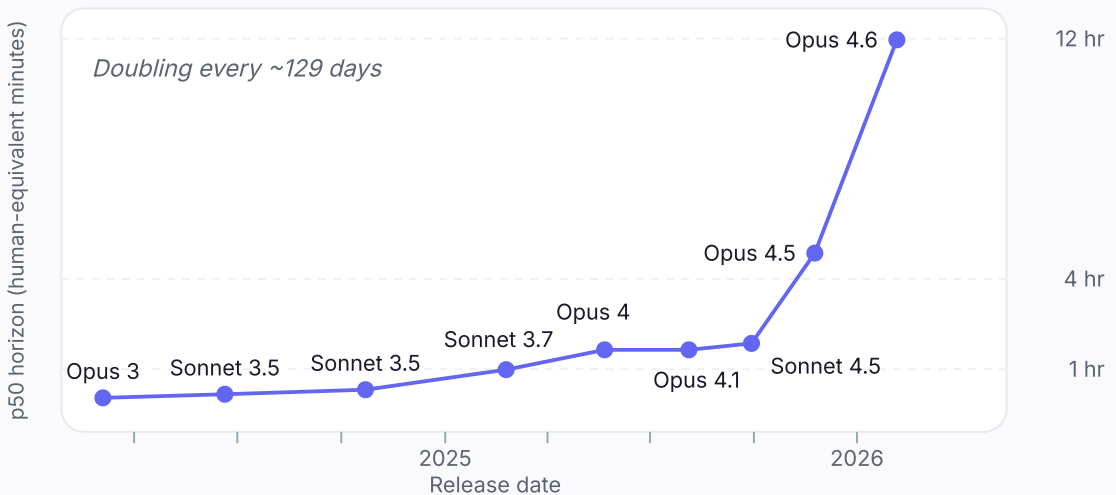
Kalau definisi tadi terasa abstrak, Anda mungkin sudah memakai harness tanpa sadar. Setiap kali Anda menyuruh AI membaca file, menjalankan perintah di terminal, lalu memperbaiki kodenya sendiri, ada harness di belakangnya.

Di luar daftar ini ada juga kerangka kerja (framework) untuk merakit agent: LangGraph, OpenAI Agents SDK, Google ADK, CrewAI, dan lainnya. Untuk satu agent yang mengerjakan tugas panjang dan terbuka, biasanya yang lebih dibutuhkan adalah harness, bukan framework.

Daftar ini cepat berubah. Anggap sebagai potret satu waktu, bukan peringkat.

Kenapa agent butuh harness

Tugasnya makin panjang



Semakin besar tujuan yang kita berikan pada agent, semakin lama ia bekerja. Data grafik dari METR, pihak yang mengukur kemampuan model AI, menunjukkan hal ini.

Panjang tugas, diukur dari lama manusia mengerjakannya, yang berhasil diselesaikan model dengan peluang sekitar 50%.

Semakin panjang tugasnya, semakin besar peluang ada yang salah di tengah jalan.

Model Anthropic terbaru

Sonnet 5, Fable 5.1, dan Opus 5.5

Sonnet 5

30 Juni 2026

- Menyelesaikan tugas kompleks yang di Sonnet sebelumnya berhenti di tengah jalan
- Menjaga fokus lebih lama
- Tidak ada angka jam

Fable 5.1

September 2026

- Lebih nyaman dengan kerja panjang tanpa pengawasan dibanding Fable 5
- 38 jam tanpa pengawasan pada masalah machine learning (Ramp)
- Prototipe kompleks dalam sekitar tiga hari (MongoDB)

Opus 5.5

22 September 2026

- Codebase 200.000 baris diaudit dan diperbaiki dalam kurang dari tiga jam
- Lebih dari 18 jam bekerja di enam repositori
- HAProxy: 9,5 jam, sedangkan Fable 5.1 12 jam

Hasil uji internal dan kesaksian pelanggan, bukan pengukuran independen.

Beberapa waktu ini, Anthropic merilis tiga model baru: Sonnet 5 (30 Juni 2026), Fable 5.1 (September 2026), dan Opus 5.5 (22 September 2026).

Halaman time horizon milik METR, pihak yang mengukur metrik ini, terakhir diperbarui 8 Mei 2026 dan belum memuat ketiga model itu.

Hati-hati membandingkan angka ini dengan grafik tadi. Angka p50 horizon menyatakan panjang tugas menurut waktu kerja manusia. Angka "38 jam" atau "18 jam" menyatakan lama agent benar-benar bekerja.

Apa yang bisa salah di tugas panjang

Sembilan masalah yang tidak muncul di tanya-jawab

- | | | |
|---------------------------|------------------------|------------------------|
| 1 Berputar tanpa henti | 2 Mengarang tool | 3 Merusak state |
| 4 Melewati anggaran token | 5 Kehilangan konteks | 6 Mengulang aksi gagal |
| 7 Gagal diam-diam | 8 Menyalahgunakan izin | 9 Melupakan tujuan |

Garis tebal: dua yang paling mahal, loop tanpa akhir dan keamanan

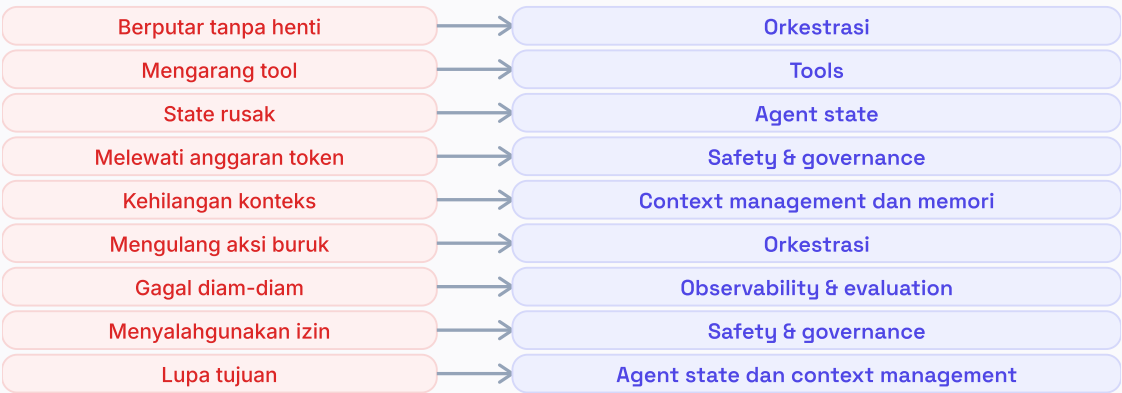
Dua di antara sembilan masalah ini paling mahal. Pertama, loop tanpa akhir. Kalau agent berputar selama 12 jam, token dan komputasi yang terbakar sangat besar. Penangkalnya adalah **anggaran** (budget) di harness: batas waktu, batas token, atau batas komputasi per tugas.

Kedua, keamanan. Kalau model memilih tool yang disruptive, misalnya tool yang menghapus semua file, akibatnya ngeri. Lewat harness, kita mengatur apa yang boleh diakses model.

Harness memberi kita kendali atas agent, dan kendali itu yang membuat kita berani mempercayakan pekerjaan padanya.

Peta masalah ke komponen

Siapa menangkal apa



Setiap masalah di atas punya komponen harness yang paling bertanggung jawab. Gunakan peta ini sebagai panduan berpikir.

Contoh penangkal: batas jumlah langkah dan batas waktu untuk loop; hanya tool terdaftar yang bisa dipanggil; anggaran token per tugas; izin minimum dan persetujuan manusia; log tiap langkah dan pemeriksaan hasil akhir.

Bagian berikut membahas kedelapan komponen satu per satu.

BAGIAN 2

2

Delapan komponen

Setiap komponen menjawab satu pertanyaan.

DI BAGIAN INI

- | | |
|-------------------------------|------------------------|
| 01 Delapan komponen harness | 02 Agent state |
| 03 Tools | 04 Planning |
| 05 Memory | 06 Orchestration |
| 07 Context management | 08 Safety & governance |
| 09 Observability & evaluation | |

Delapan komponen harness

Gambaran umum

1 Agent state Saya ada di mana sekarang?	5 Orchestration Bagaimana proses ini dijalankan?
2 Tools Apa yang bisa saya lakukan?	6 Context management Apa yang perlu dilihat LLM sekarang?
3 Planning Apa yang harus saya lakukan?	7 Safety & governance Apa yang boleh saya lakukan?
4 Memory Apa yang saya tahu dan ingat?	8 Observability & evaluation Apakah ini benar-benar berhasil?

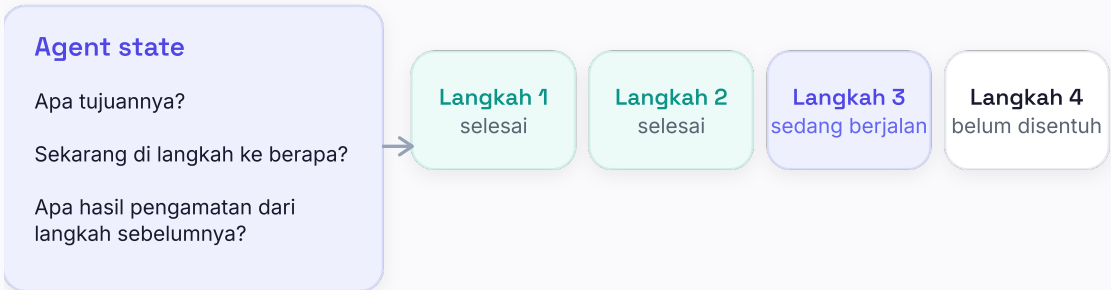
Apa pun namanya, sebuah harness hampir selalu terdiri dari delapan komponen. Setiap komponen menjawab satu pertanyaan.

Disiplin yang merancang, membangun, dan menjalankan kedelapan komponen ini supaya agent bekerja optimal disebut **harness engineering**.

Analogi dapur akan saya pakai di setiap bagian berikut supaya tidak berganti-ganti perumpamaan.

Agent state

Saya ada di mana sekarang?



State adalah catatan posisi agent: tujuannya, langkah ke berapa, hasil pengamatan terakhir, dan keputusan yang sudah diambil.

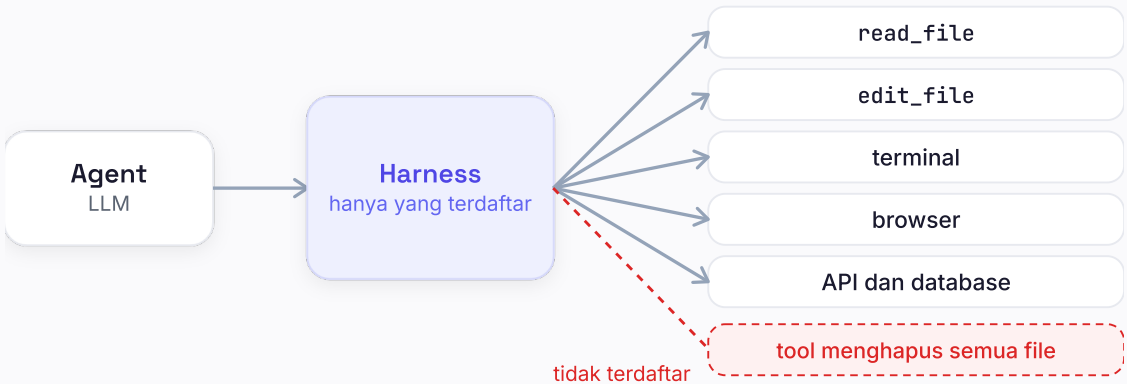
Contohnya, sebuah rencana punya 4 langkah. State mencatat langkah 1 dan 2 sudah selesai, langkah 3 sedang berjalan, langkah 4 belum disentuh.

Analogi dapur: tiket pesanan dengan centang. Koki melihat tiket, tahu bahwa bawang sudah dipotong dan ayam belum digoreng. **Tanpa tiket, koki yang terganggu sebentar bisa mengulang dari awal atau melewati satu langkah.**

State yang hilang atau rusak membuat agent mengulang pekerjaan, melewati langkah, atau lupa tujuan.

Tools

Apa yang bisa saya lakukan?



Tool memberi agent kemampuan bertindak di luar teks: menjalankan kode, membuka browser, memanggil API, mengakses database, memakai terminal.

Tool yang paling jelas dan paling dasar untuk agent coding adalah **read_file**, yaitu fungsi untuk membaca isi file.

Pasangannya biasanya write_file atau edit_file.

Tugas harness untuk tool: menyediakan daftar tool beserta cara memakainya, mengembalikan hasilnya ke model, dan menolak panggilan ke tool yang tidak terdaftar.

Analogi dapur: pisau, kompor, dan mixer.

Planning

Apa yang harus saya lakukan?



Planning memutuskan apa yang terjadi berikutnya, lalu membuat atau merevisi rencana.

Pada tugas coding, komponen planning bisa menyusun 4 sampai 5 langkah berurutan, seperti di diagram.

Rencana boleh berubah. Kalau tes gagal, rencana direvisi: mungkin perlu membaca file lain, atau mencoba perbaikan yang berbeda.

Analogi dapur: urutan kerja koki. Kaldu direbus dulu karena butuh lama, saus dibuat terakhir supaya masih hangat.

Rencana yang baik menjawab dua hal: apa langkah berikutnya, dan apa tanda bahwa langkah itu berhasil.

Memory

Apa yang saya tahu dan ingat?



Memori menyimpan dan mengambil informasi lintas langkah atau lintas sesi. Definisi singkatnya: memori adalah **informasi yang tidak sedang ada di konteks**.

Vector DB adalah penyimpanan yang bisa dicari berdasarkan kemiripan makna, cocok untuk mengambil catatan bug lama yang mirip dengan masalah sekarang.

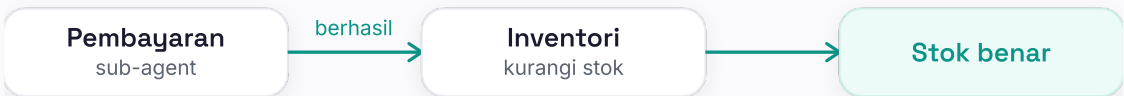
Isi yang diambil dari Vector DB sering perlu diringkas dulu supaya ruang kerja model tidak penuh.

Analogi dapur: buku resep di rak. Koki mengambil satu halaman yang relevan dan menaruhnya di meja. Halaman di meja itulah "konteks".

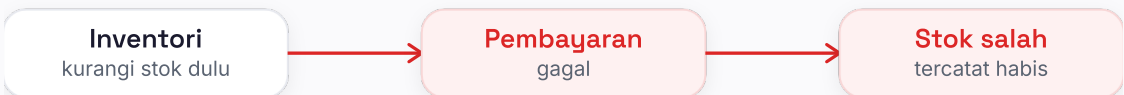
Orchestration

Bagaimana proses ini dijalankan?

BERURUTAN



URUTAN TERBALIK



Orkestrasi mengendalikan putaran eksekusi: kapan tool dipanggil, apa yang dilakukan kalau gagal (retry), kapan sub-agent dipanggil, dan apa yang berjalan paralel.

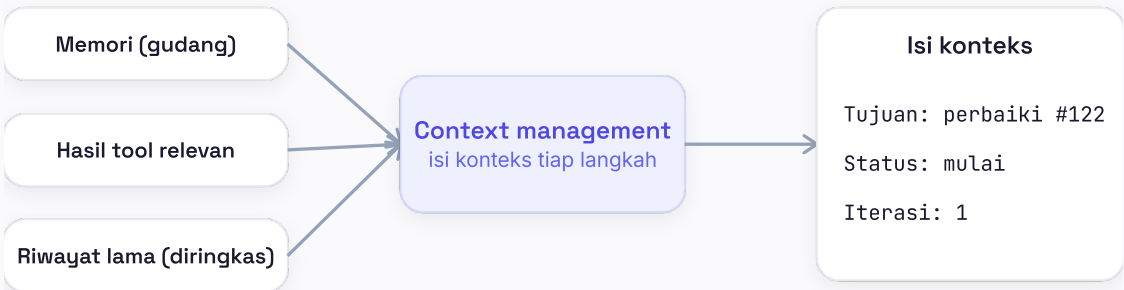
Contoh: agent yang menangani pembelian. Sub-agent pembayaran dipanggil, pembaruan stok dikerjakan sub-agent inventori. **Berurutan** wajib untuk pekerjaan yang bergantung pada hasil sebelumnya.

Sub-agent inventori sebaiknya baru mengurangi stok **setelah pembayaran berhasil**. Kalau stok dikurangi lebih dulu dan pembayaran gagal, **barang tercatat habis padahal tidak terjual**.

Analogi dapur: kasir baru boleh mencetak nota setelah pesanan selesai disajikan.

Context management

Apa yang perlu dilihat LLM sekarang?



Context window adalah ruang terbatas berisi semua yang bisa “dilihat” model dalam satu langkah. Context management menentukan apa yang masuk ke ruang itu di setiap langkah.

Memori adalah gudang di luar konteks. Context management adalah keputusan barang mana dari gudang yang ditaruh di meja kerja saat ini.

Isi konteks pada awal tugas bug #122 kira-kira: tujuan (perbaiki #122), status (mulai), dan nomor iterasi (1).

Analogi dapur: meja kerja koki. Meja yang penuh barang membuat koki lambat dan salah ambil. Meja yang terlalu kosong membuatnya bolak-balik ke gudang.

Safety & governance

Apa yang boleh saya lakukan?

AKSI	KEPUTUSAN
mengedit file	Boleh
menghapus semua atau sebagian file	Tidak boleh
menghapus data	Tunggu persetujuan manusia
mengirim uang	Tunggu persetujuan manusia

Komponen ini mencakup izin (permissions), sandboxing, gerbang persetujuan (approval gates), kebijakan, batas pemakaian, dan autentikasi. Pertanyaan intinya: kalau model punya tool yang sangat kuat, sampai mana ia boleh melangkah?

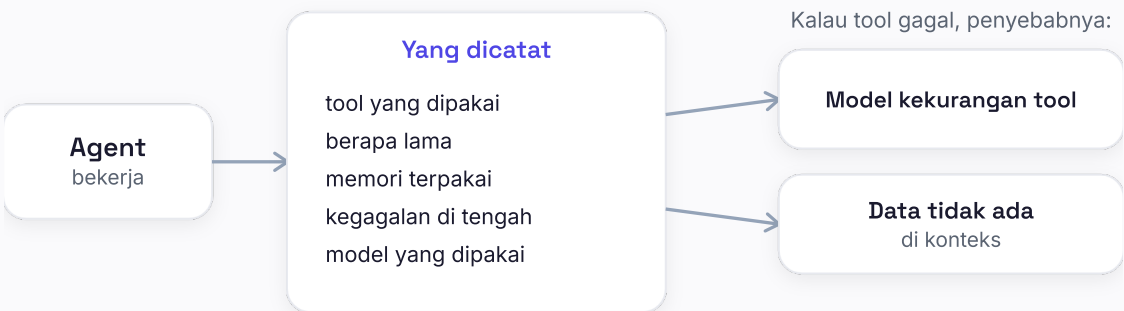
Contoh: pada tugas coding, model boleh mengedit file, tetapi lebih aman kalau ia tidak boleh menghapus semua atau sebagian file di sistem.

Aksi berisiko tinggi, seperti menghapus data atau mengirim uang, menunggu persetujuan manusia.

Analogi dapur: asisten baru boleh memotong sayur, tetapi tidak boleh mengatur gas tabung.

Observability & evaluation

Apakah ini benar-benar berhasil?



Observability melacak apa yang terjadi selama agent bekerja. Evaluation mengukur apakah agent berhasil, gagal, atau melanggar batasan.

Contoh: sebuah tugas mungkin hanya butuh 3 dari 10 tool. Pola pemakaian itu layak dicatat. Kalau tool gagal, penyebabnya bisa dua: model kekurangan tool, atau data yang dibutuhkan tidak ada di konteks. **Tanpa catatan, kita hanya bisa menebak.**

Analogi dapur: kepala dapur mencicipi piring sebelum keluar, dan mencatat piring mana yang dikembalikan tamu.

BAGIAN 3

3

Contoh dan arah

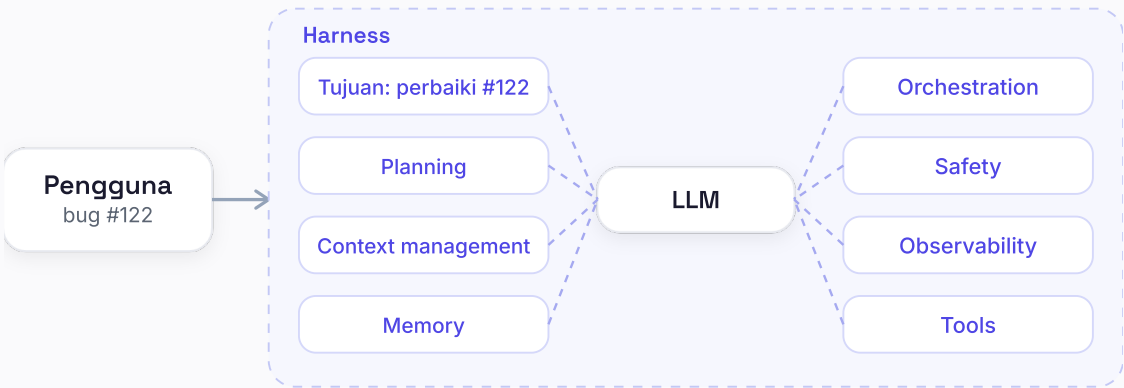
Agent yang memperbaiki bug di GitHub, langkah demi langkah, dan tantangan yang masih terbuka.

DI BAGIAN INI

- 01 Contoh lengkap
- 02 Contoh bug #122, langkah demi langkah
- 03 Evaluasi
- 04 Arah ke depan

Contoh lengkap

Agent yang memperbaiki bug di GitHub



Sekarang kita lihat kedelapan komponen bekerja bersama.

Pengguna mengirim prompt: **"Selesaikan bug di issue #122 di GitHub."**

Ini tugas nyata, tidak sama dengan permintaan "jelaskan kepada saya". Ia butuh perencanaan, tool, memori, dan lainnya.

Satu cara logis untuk mengimplementasikannya ada di bagian berikutnya.

Contoh bug #122, langkah demi langkah

Komponen yang bekerja di tiap langkah

1	Input masuk ke konteks, model menetapkan tujuan	Context management · state
2	Model menyusun rencana	Planning
3	Harness mengambil data bug lama dari Vector DB	Memory · context management
4	Model mengedit file auth.py	Tools
5	Model mengetes perubahan lewat terminal	Tools · safety
6	Model menjalankan test case	Orchestration · state · context
7	Semuanya dicatat	Observability

Diagram ini menunjukkan urutan kejadian dan komponen yang bekerja di tiap langkah.

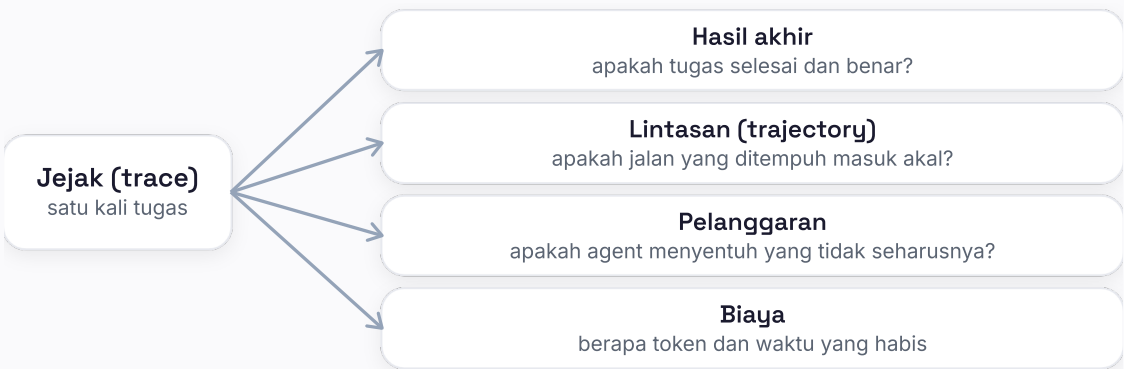
Setiap langkah memperbarui konteks agar posisi dan sisa pekerjaan tetap jelas.

Perhatikan bahwa evaluasi belum dibahas di contoh ini. Evaluasi adalah topik tersendiri dan tidak hanya soal akurasi.

Analogi dapur: tiket masuk, koki menyusun urutan kerja, kepala dapur mengawasi izin, dan semua langkah dicatat di buku dapur.

Evaluasi

Bagian yang paling sulit



Makin canggih harness yang dibangun, makin sulit mengevaluasinya. Setiap tugas meninggalkan jejak (trace) berisi banyak data: memori yang terpakai, tool yang dipanggil, langkah yang diulang.

Dari jejak itu muncul pertanyaan yang sulit dijawab. Sebaiknya model diganti, atau harness diperbaiki? Perlu menambah tool, atau memperbaiki tool yang sudah ada? Kegagalan datang dari model, dari data di konteks, atau dari aturan izin?

Hasil akhir yang benar dengan **lintasan yang berantakan** adalah peringatan, karena kebetulan tidak bisa diulang.

Arah ke depan

Keandalan tugas panjang

Keandalan pada tugas panjang
long-horizon reliability

Evaluasi
tantangan kedua

YANG BISA DILAKUKAN SEKARANG

Harness sederhana
anggaran, izin minimum, log

Baca log
di mana agent tersandung

Tambah kecanggihan

Salah satu tantangan terbesar agentic AI saat ini adalah **keandalan pada tugas panjang** (long-horizon reliability). Agent yang berjalan berjam-jam, bahkan berhari-hari, harus tetap andal. Kalau agent terjebak dalam loop atau state yang rusak, kita membuang sumber daya dan bisa membahayakan keamanan.

Tantangan kedua adalah evaluasi. Komunitas masih aktif membangun harness yang lebih canggih, dan jawabannya belum baku.

LAKUKAN INI

Mulai dari harness sederhana dengan anggaran, izin minimum, dan log lengkap. Tambah kecanggihan setelah data dari log menunjukkan di mana agent sering tersandung.

BAGIAN 4

4

Praktik dan penutup

Checklist, glosarium, rekap, dan tentang penulis.

DI BAGIAN INI

01 Checklist merancang harness sendiri

02 Glosarium

03 Rekap

04 About Author

Checklist merancang harness sendiri

Pertanyaan untuk tiap komponen

☐ **State**
Tujuan dan langkah tercatat di satu tempat?

☐ **Tools**
Setiap tool punya deskripsi jelas?
Hanya tool yang diperlukan yang diberikan?

☐ **Planning**
Agent boleh merevisi rencana
saat langkah gagal?

☐ **Memory**
Apa yang disimpan lintas sesi,
dan siapa yang boleh mengubahnya?

☐ **Orchestration**
Ada batas jumlah langkah, waktu, dan retry?
Pekerjaan mana yang harus berurutan?

☐ **Context management**
Tujuan selalu terlihat di konteks?
Ada cara meringkas riwayat yang panjang?

☐ **Safety & governance**
Izin dibatasi ke minimum?
Aksi berisiko menunggu persetujuan manusia?

☐ **Observability & evaluation**
Tiap langkah tercatat?
Ada ukuran keberhasilan yang disepakati
sebelum agent jalan?

Pakai daftar ini saat menyiapkan agent untuk tugas nyata.

Glosarium

Istilah yang dipakai di panduan ini

LLM

Large language model, model bahasa besar.

Prompt

Instruksi atau pertanyaan yang dikirim ke model.

Context window

Ruang terbatas berisi semua yang dilihat model dalam satu langkah.

Tool

Kemampuan tambahan seperti terminal, browser, atau API.

State

Catatan posisi agent: tujuan, langkah, hasil pengamatan.

Memory

Informasi yang tersimpan di luar konteks.

Vector DB

Penyimpanan yang bisa dicari berdasarkan kemiripan makna.

Sub-agent

Agent kecil yang diberi bagian tugas oleh agent utama.

Orkestrasi

Pengaturan urutan, paralelisme, retry, dan serah terima.

Sandbox

Ruang terisolasi tempat agent bekerja aman.

Approval gate

Titik yang menunggu persetujuan manusia.

Trace

Jejak lengkap satu kali tugas.

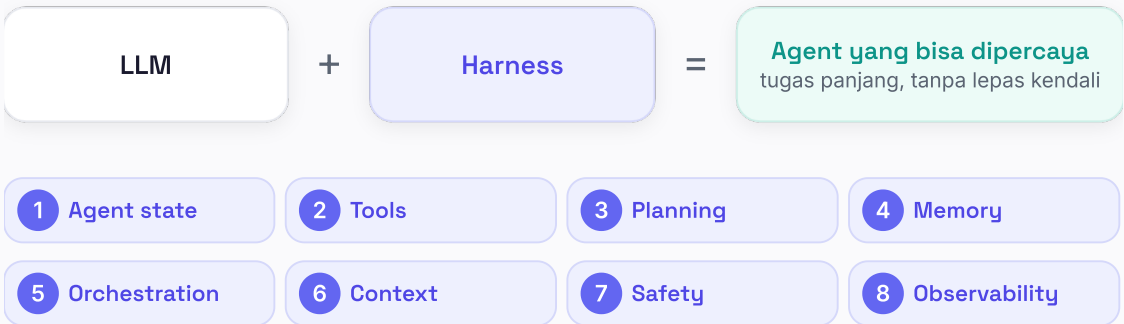
Agent harness: lapisan infrastruktur yang membungkus, mengendalikan, menguji, memantau, dan mengevaluasi AI agent. **AI agent:** LLM yang dilengkapi tool, memori, dan kemampuan merencanakan sehingga bisa mengerjakan tugas.

Observability: kemampuan melihat apa yang terjadi di dalam agent lewat log dan jejak. **Halusinasi:** model mengarang sesuatu yang tidak ada, misalnya tool atau fakta.

Long-horizon task: tugas panjang yang butuh banyak langkah dan waktu. **Harness engineering:** disiplin merancang dan menjalankan komponen harness.

Rekap

Model ditambah harness



Agent harness adalah lapisan infrastruktur di sekeliling LLM. Ia yang mengubah model yang pandai menjawab menjadi agent yang bisa mengerjakan tugas panjang tanpa lepas kendali.

Tugas agent makin panjang, dan masalah yang menyertainya makin mahal. Harness memberi anggaran, batas izin, dan catatan supaya kita bisa memeriksa dan mempercayai hasilnya.

LANGKAH BERIKUTNYA

Pilih satu tugas kecil yang berulang, tulis kedelapan jawaban di atas untuk tugas itu, lalu jalankan agent dan baca lognya.

About Author

Aan Setianto



Aan Setianto

Passionate Entrepreneur | AI Enthusiast

aansetianto.com

You can reach me at

vascomm.co.id

Corporate Website

Saya menulis panduan ini dari dua sudut pandang: pelaku usaha yang butuh hasil nyata, dan AI Enthusiast yang ingin tahu cara kerjanya di balik layar. Saya menyederhanakan materi teknis supaya bisa dipahami orang awam, lengkap dengan contoh dan analogi supaya lebih mudah.

Kalau panduan ini membantu Anda memahami agent harness, lanjutkan dengan mencoba. Pilih satu tugas kecil di pekerjaan Anda, terapkan delapan pertanyaan komponen di atas, dan lihat di mana agent Anda perlu diperketat.

Index

About Author	35	Contoh lengkap	27	Orchestration	22
Agent state	18	Dari tanya-jawab ke tugas	7	Pengguna makin ambisius	8
Apa itu agent harness?	10	Delapan komponen harness	17	Peta masalah ke komponen	15
Apa yang bisa salah di tugas panjang	14	Evaluasi	29	Planning	20
Arah ke depan	30	Glosarium	33	Rekap	34
Checklist merancang harness sendiri	32	Kenapa agent butuh harness	12	Ringkasan	6
Context management	23	Memory	21	Safety & governance	24
Contoh agent harness	11	Model Anthropic terbaru	13	Siklus kerja agent	9
Contoh bug #122, langkah demi langkah	28	Observability & evaluation	25	Tools	19