20 slides · zero AI slop

Agent Harness.
Loop × Graph Engineering.

The model is the intelligence. The harness is what makes that intelligence *useful*. Three disciplines that turn an that turn an LLM into an agent that actually ships.

$ agent = model + harness

> harness = tools + loop + graph + hooks + sandbox

If you're not the model, you're the harness.

Sources: langchain.com · anthropic.com/engineering · openai.com Q2 2026

The rule of the room

A great model in a naive loop is a demo. A modest model modest model in a tight harness is a product.

01 · HARNESS

The scaffolding around the model — tools, filesystem, sandbox, hooks, hooks, memory, skills. Every model deficiency you patch, every capability capability you extend, lives here.

Owned by: you, not the model provider.

02 · LOOP

The control flow that keeps the model working — think → act → observe observe → repeat. Stack it: agent loop, verification loop, event loop, hill-loop, hill-climbing loop.

Where value compounds: loops 3 & 4.

03 · GRAPH

Explicit workflow topology — nodes do work, edges decide what happens next. State moves through. Deterministic where you can be, agentic where you must.

Cognitive architecture = your world knowledge encoded in the graph.

Sources: langchain.com/blog/the-anatomy-of-an-agent-harness · langchain.com/blog/the-art-of-loop-engineering · langchain.com/blog/3-years-of-graph-engineering-with-langgraph.

GRAPH ⊂ LOOP ⊂ HARNESS

One term, three nested lenses. Learn to pick the right one for the problem in front of you. you.

OUTER SHELL · HARNESS ENGINEERING

Everything that is not the model.

MIDDLE LAYER · LOOP ENGINEERING

The control flow inside the harness.

INNER LAYER · GRAPH ENGINEERING

The topology of the loop.

Nodes = work. Edges = transitions. State = what moves through.

WHICH QUESTION EACH ANSWERS

Harness → *what can the agent see, touch, and do?*
Filesystem access, tool inventory, sandbox limits, memory injection, hook policies.

Loop → *when does it stop, retry, or improve?*
Termination rules, compaction thresholds, verification gates, trace-driven refactors.

Graph → *which paths are valid?*
Encode structure you already know. Let the model choose only where the model must.

Sources: martinfowler.com/articles/harness-engineering.html (feedforward + feedback model) · langchain.com stack of loops (loopcraft) · langgraph docs cognitive architectures.

Terminal-Bench 2.0 proof

Agent = Model + Harness.

A raw model is **not** an agent. It becomes one when a harness gives it state, tool execution, feedback loops, and enforceable enforceable constraints.

THE MODEL PROVIDES

Text in, text out. That's it.

Cannot maintain durable state across sessions. Cannot execute code. Cannot access realtime knowledge. Cannot set up environments or install packages. Cannot verify its own output.

Everything else is a harness-level feature.

THE HARNESS ADDS

System prompts · Tools, Skills, MCP · Filesystem, sandbox, browser · Orchestration & sub-agents · Hooks & agents · Hooks & middleware · Compaction · Continuation · Verification · Memory injection.

If you're not the model — you're the harness.

Proof it matters: on Terminal-Bench 2.0, Opus 4.6 scores Top 30 in one harness → Top 5 in another. Same model, different harness. Source: LangChain, Q2 2026.

GUIDES × SENSORS

A harness is a cybernetic governor: two directions × two execution types = four categories to design deliberately.

	FEEDFORWARD (guides — before)	FEEDBACK (sensors — after)
COMPUTATIONAL deterministic · CPU · ms runs on every change	**Examples:** · LSP + type-checker in agent context · Code-mod recipes (OpenRewrite, jscodeshift) · Bootstrap scripts for new projects · Injected schema definitions (Zod, pydantic) *Cheap. Deterministic. Always on.*	**Examples:** · Structural tests (ArchUnit, dependency-cruiser) · Custom linters with remediation-in-error-msg · Test suite + coverage runs (fed back as tool output) · Mutation testing on critical paths *Runs post-action. Fires the retry.*
INFERENTIAL semantic · GPU · seconds runs selectively	**Examples:** · AGENTS.md / CLAUDE.md operating manual · Skills auto-loaded via progressive disclosure · How-to references pulled by task keyword · MCP servers exposing team knowledge *Encodes taste that a linter can't.*	**Examples:** · LLM-as-judge review agent · /architecture-review, /detailed-review skills · Log-anomaly & response-quality sampling · Playwright-driven UX evaluators (Anthropic Labs) *Catches what deterministic sensors can't.*

Source: Martin Fowler, "Harness engineering for coding agent users" (2026) · complements langchain.com harness anatomy.

what to build first

Six primitives, in build order. Each derived from "what can't the model do alone?"

01 · FILESYSTEM

Durable workspace.

Offload state beyond the context window. Multi-agent collaboration surface. Git adds versioning, rollback, branches.

Ship first. Every later primitive depends on it.

02 · BASH + CODE

General-purpose tool.

Skip building a tool for every action. Give it bash — the model writes its own tools on the fly.

"Give models a computer." One tool replaces dozens of narrow ones.

03 · SANDBOX

Safe execution env.

Bash + code without a sandbox = liability. Allow-list commands, isolate network, per-network, per-worktree bootable app.

OpenAI: per-git-worktree Chrome DevTools Protocol + logs.

04 · MEMORY + SEARCH

Continual learning.

AGENTS.md / CLAUDE.md injected at session start. Web search + Context7 MCP for post-MCP for post-cutoff knowledge. RAG over your own vault when local corpus is dense. dense.

"Give the agent a map, not a manual."

05 · SKILLS + MCP

Progressive disclosure.

Load one-line front-matter into context; load body only when triggered. Solves the "too Solves the "too many tools kill performance" problem.

Rule: if a skill is always on, promote it to system prompt; if rarely used, hide it behind a keyword.

06 · HOOKS / MIDDLEWARE

Deterministic control.

Pre-tool-use: block dangerous commands.
Post-tool-use: lint, format, test.
Stop: Ralph loop re-injects prompt.
SessionStart: load memory + CLAUDE.md.

Sources: langchain.com/blog/the-anatomy-of-an-agent-harness · openai.com/index/harness-engineering (per-worktree bootable app + observability) · Ralph Loop pattern (ghuntley.com/loop).

context = scarce resource

Longer context ≠ better answers. Performance rots as tokens pile up — reason more, act less.

TECHNIQUE 01

Compaction.

When: context window ≥ 80% full.

How: summarise older turns; keep recent turns verbatim; preserve plan + open decisions.

Trade-off: summary loses fidelity. Fine for chat; not enough for long-running builds.

Auto in Claude Agent SDK.

TECHNIQUE 02

Context reset.

When: "context anxiety" — model wraps up prematurely.

How: clear window entirely. Fresh agent reads the handoff artifact on disk. Continue.

Cost: orchestration overhead + tokens to re-hydrate.

Anthropic Labs: essential on Sonnet 4.5, less so on Opus 4.6.

TECHNIQUE 03

Tool-call offloading.

When: a tool returns 50k+ tokens (log dumps, HTML pages, pages, large JSON).

How: keep only head + tail in context. Write full output to disk. Give the model the path.

The model can re-read on demand — but usually doesn't need to.

TECHNIQUE 04

Progressive disclosure.

When: you have 20+ skills, docs, or MCP tools.

How: load one-line front-matter into context on start; load the body only when the model asks for it.

OpenAI: AGENTS.md ≈ 100 lines — a table of contents, not an encyclopedia.

Sources: research.trychroma.com/context-rot · anthropic.com/engineering/effective-context-engineering-for-ai-agents · openai.com/index/harness-engineering "give it a map, not a manual".

Deciding where the agent runs is a design choice. It's not the model's job — it's yours.

SANDBOX PATTERN · PER-WORKTREE BOOTABLE APP

One ephemeral world per task.

```
$ git worktree add ../task-042 -b feat/task-042
$ cd ../task-042 && docker compose up -d
># app + observability stack booted just for this task

$ claude --sandbox=workspace-write
> reproduce bug, verify fix via Chrome DevTools Protocol
> query logs (LogQL) and metrics (PromQL)

$ git commit && git worktree remove ../task-042
># entire environment torn down — no leaks
```

TOOL DESIGN · FOUR RULES

Every tool has taste.

01 · Prefer one general tool over ten specific ones.
Bash + code beats a fixed CRUD API surface. Model designs its own tool on the fly.

02 · Errors ARE the interface.
A lint error message is a prompt injection back into the loop. Write it as instructions: *"To fix: change X to Y and re-run npm test."*

03 · Head + tail, not everything.
A 200-line log is fine. A 20k-line log crushes context. Truncate at the tool boundary.

04 · Boring beats novel.
Stable, well-documented libs (JSON, SQL, git) beat clever DSLs. Training data density wins.

Sources: openai.com/index/harness-engineering (per-worktree bootable app + LogQL/PromQL) · anthropic.com/engineering/harness-design-long-running-apps (sprint contracts) · Fowler on inferential vs computational sensors.

think · act · observe

One model + tools + a while-loop = an agent. Everything else is optimisation.

THE PSEUDOCODE — MEMORISE IT

```
while not done:
response = model.invoke(
messages=history,
tools=tools,
system=system_prompt,
)

if response.tool_calls:
for call in response.tool_calls:
result = execute(call)
history.append(result)
else:
done = True# model stopped calling tools
```

THREE WAYS THE LOOP ENDS

01 · MODEL STOPS.
The model returns text with no tool calls — believes work is done. The happy path.

02 · BUDGET HIT.
Max iterations reached (e.g. 40). Max tokens consumed. Max wall-clock. The harness kills the loop.

03 · GUARD FIRES.
A hook detected repetition, an off-limits action, or a critical error. Escalate or stop.

Never leave an agent with no termination guard.
An unbounded loop is a bill, not a feature.

Sources: langchain.com/blog/the-art-of-loop-engineering (Loop 1) · stevekinney.com/writing/agent-loops (production hardening) · docs.langchain.com create_agent primitive.

pick one deliberately

ReAct thinks *while* acting. Plan-and-Execute thinks *before* acting. Different loops, different failure different failure modes.

TOPOLOGY 01 · REACT (REASONING + ACTING)

Thought → Action → Observation → repeat.

```
Thought: I need the user's order status.
Action: query_orders(user_id=42)
Observation: [{"id":9,"status":"pending"}]
Thought: Pending means we can still cancel.
Action: cancel_order(9)
```

Best for: exploratory tasks with unknown paths — support triage, research, debugging.

Failure mode: the model wanders. Repeats tool calls. Over-thinks trivial steps.

Cost: higher token spend per task (many thought turns).

TOPOLOGY 02 · PLAN-AND-EXECUTE

Plan once → execute steps → optionally replan.

```
Plan: 1. read spec.md
2. scaffold /api routes
3. write tests
4. run tests
5. open PR
Executor: dispatch each step; on fail → replan.
```

Best for: multi-step builds with a clear spec — coding agents, migrations, batch pipelines.

Failure mode: plan is wrong from the start and executor grinds through anyway. Replan gate is essential.

Cost: lower token/step, but wasted work if the plan is off.

Related patterns to explore next: ReWOO (plan then execute w/o replan), Reflexion (agent critiques its own trace), Ralph loop (re-inject prompt on stop). Sources: theaiengineer.substack.com/p/the-4-single-agent-patterns · arXiv Yao et al. ReAct paper.

generator × evaluator

The model can't grade itself. Wrap the loop in a separate evaluator — it scores, sends back feedback, retries.

THE FLOW · GAN-INSPIRED

```
iteration = 0
while iteration < MAX_ITER:
output = generator.run(task, feedback)

# evaluator has a DIFFERENT system prompt
# tuned to be skeptical, not generous
score, feedback = evaluator.grade(
output, criteria=RUBRIC
)

if score >= THRESHOLD:
break
iteration += 1
```

THE RUBRIC MATTERS MORE THAN THE PROMPT

01 · Decompose the judgement.
"Is this good?" → "does the API return 4xx on invalid input? Does the UI trap focus? Does the diff match the spec?" Concrete, per-criterion scores.

02 · Weight subjectivity down.
Anthropic Labs weight *design* + *originality* above *craft* because craft is easy — model does it by default.

03 · Give the evaluator real tools.
Playwright MCP → click through the running app. LogQL → verify the API. The evaluator is not just a text critic.

Trade-off: 5-15 iterations × real Playwright driving = hours per run.

Sources: anthropic.com/engineering/harness-design-long-running-apps (generator/evaluator/planner architecture) · langchain.com RubricMiddleware · deepagents rubric docs.

webhook · cron · queue

The agent isn't something you invoke. It's a component running continuously in your ecosystem. your ecosystem.

PATTERN 01 · WEBHOOK-TRIGGERED

React to real events.

```
# GitHub webhook
on: pull_request
→ /agent/review

# Slack event
on: message #docs-plz
→ /agent/docs-fix

# Linear issue created
on: issue.new label=bug
→ /agent/triage
```

Best for: code review, docs updates, bug triage, notification-to-action.

Signal quality: highest — the event IS the trigger, no polling noise.

Pin idempotency keys per event ID.

PATTERN 02 · CRON HEARTBEAT

Proactive by schedule.

```
# crontab-style
0 9 * * MON-FRI
→ /agent/morning-triage

*/15 * * * *
→ /agent/inbox-classify

0 3 * * SUN
→ /agent/dead-code-scan
```

Best for: maintenance work — doc gardening, dead-code detection, SLA drift monitoring. monitoring.

Trap: cron × no dedupe = duplicate PRs. Enforce a "one active run per agent" lock.

OpenAI's doc-gardening agent runs on this.

PATTERN 03 · QUEUE-DRAINED

Backpressure control.

```
# N workers drain a queue
queue: agent_tasks
concurrency: 4
retry: 3 exponential
dlq: agent_tasks_dead

# dead-letter → human
on: dlq.arrival
→ /alert-oncall
```

Best for: bulk work — 10k records to classify, 500 emails to draft, batch migrations.

Why queues: retries, DLQ, concurrency caps, cost ceilings — the boring reliability layer LLM tasks always need.

Sources: langchain.com/blog/the-art-of-loop-engineering (Loop 3) · docs.langchain.com/langsmith cron-jobs + webhooks · docs.openclaw.ai heartbeat pattern.

where value compounds

Loops 1-3 automate work. Loop 4 automates improvement — improvement — traces feed back into the harness itself. itself.

THE PIPELINE

01 collect production traces
→ tool calls, grader feedback, retries, latency

02 analysis agent scans N recent traces
→ clusters recurring failure signatures

03 filed as issues / PRs against the harness
→ prompt tweaks, new hooks, new skills

04 human review (or agent-approved auto-merge)

05 next cohort runs on the improved harness

WHAT TO OPTIMISE — IN ORDER

01 · Prompts & tool descriptions.
Cheapest, fastest signal. Ambiguous tool docs = wrong-tool calls.

02 · Grader rubrics.
Ship a stricter rule the moment a failure escapes review three times.

03 · Retrieved skills / memory.
Traces reveal which docs are actually consulted vs. dead weight.

04 · Fine-tune (open-weight only).
Trace outcomes → RL signal. Only after you've squeezed 1-3.

Sources: langchain.com/blog/the-art-of-loop-engineering (Loop 4 · hill climbing) · langsmith.com Engine · Satya Nadella framing: "companies that build learning loops early will build an advantage that's hard to replicate."

quadratic ≠ linear

Every step re-sends the entire history. A 20-step loop costs 10× what a per-step estimate suggests. estimate suggests.

THE COST MATH · WHY IT COMPOUNDS

Input tokens grow with the sum of turns.

```
# step 1 input = system + task → 2k tokens
# output = plan → 1k tokens

# step 2 input = above + tool result → 4k tokens
# output = new action → 1k tokens

# step 20 input ≈ 40k · output ≈ 1k

total input tokens
≈ Σ(step_context_i) for i in 1..N
≈ N × avg_step_size ≈ O(N²)

rescue: prompt caching cuts prefix cost 90%.
```

FIVE CONTROLS · APPLY ALL

01 · Enable prompt caching. Anthropic + OpenAI both cache stable prefixes. Cost drops ~90% on the reused system prompt. reused system prompt.

02 · Compact at 80% of window. Not at 100% — you need slack for the compaction call itself.

03 · Tier your models. Cheap model for planning & classification, expensive model only for the hard step.

04 · Sub-agents own their own budgets. Parent doesn't see child's context — quadratic growth is bounded per sub-agent. per sub-agent.

05 · Hard ceilings + circuit-breaker. If a task hits 3× budget → auto-abort, page a human, don't just keep spending. spending.

Sources: augmentcode.com/guides/ai-agent-loop-token-cost-context-constraints · mindstudio.ai Claude Code token-budget mgmt · stevekinney.com/writing/agent-loops.

Graph = encoded world knowledge. Not every problem has enough of it to encode.

USE A GRAPH WHEN

Structure is *knowable* up front.

01 · Compliance-gated flows. "Never call the payment API without an approval token." The graph enforces it — the model doesn't get to skip.

02 · Classify → route. Support triage. Content moderation. Different downstream paths for different classifications.

03 · Multi-source parallel fetch. Fan out to GitHub + Notion + Slack, synthesise the results. Deterministic fan-out Deterministic fan-out beats a model "remembering" to call all three.

04 · Human-in-the-loop checkpoints. Approval gates on external actions — the edge waits, not the prompt.

DON'T USE A GRAPH WHEN

The task is *open-ended*.

01 · Deep research. Plan-delegate-search-read-synthesise loops that mutate with each finding. Better handled as Better handled as an agent harness (Deep Agents).

02 · Autonomous coding. "Build me a DAW" doesn't fit a hard-coded topology. Let the loop drive.

03 · Chat. A conversation is not a workflow. Wrap it in a simple loop with tools.

Migration signal: GPT Researcher swapped its graph-shaped multi-agent pipeline for Deep Agents. Structure was constraining, not helping.

Sources: langchain.com/blog/3-years-of-graph-engineering-with-langgraph (when to encode structure) · github.com/assafelovic/gpt-researcher/deep_agents (migration writeup) · anthropic.com/research/building-effective-agents ("simplest solution first").

nodes · edges · state

A graph is a state machine. Nodes do work. Edges decide what happens next. State moves through.

NODES · THEY DO WORK

Anything callable.

```
graph.add_node(
"classify",
classify_fn,
)
graph.add_node(
"github_search",
github_agent,
)
```

A node can be:

· pure Python — deterministic classifier / regex router
· a single LLM call — the classifier
· a tool call — external API
· *a whole agent* — Claude Code as a node

Rule: a node returns a state delta, not the full state.

EDGES · THEY DECIDE

Fixed or conditional.

```
graph.add_edge(
"classify", "route"
)
graph.add_conditional_edges(
"route",
lambda s: s["kind"],
{"bug": "repro", "q": "answer"}
)
```

Edge types:

· `add_edge` — deterministic transition
· `add_conditional_edges` — branch by state
· `END` — terminate this run

Cycles are normal. Real agents retry, ask, resume. LangGraph is *not* a DAG runner.

STATE · WHAT MOVES

Typed. Reducer-composed.

```
classAgentState(TypedDict):
task: str
messages: Annotated[
list, add_messages
]
plan: list[str]
retries: int
```

State rules:

· **TypedDict** — schema is code, not tribal knowledge.
· **Reducer per field** — how nodes' deltas merge (append for messages, replace for retries).
· **Persist state** for pause/resume, human-in-the-loop, time-travel debugging.

Sources: docs.langchain.com/oss/python/langgraph/graph-api (nodes + edges) · langchain.com/blog/3-years-of-graph-engineering-with-langgraph · dev.to/jamesli LangGraph state machines in production.

runtime cardinality

"Split, work, combine" — but the number of workers is only only known at runtime.

SEND API · LANGGRAPH

```
# the planner writes N URLs into state
def plan(state):
return {"urls": model.list_urls(state)}

# fan out ONE Send per url
def route_to_workers(state):
return [Send("scrape", {"url": u})
for u in state["urls"]]

graph.add_conditional_edges(
"plan", route_to_workers,
["scrape"]
)
# reducer on state["docs"] concatenates results
```

WHERE YOU'LL REACH FOR IT

01 · Research fan-out.
Planner decides sources, workers scrape each, synthesiser merges. Count unknown until plan is done.

02 · Batch email/ticket triage.
N inputs → N classifications → one aggregated verdict. Each Send is bounded.

03 · Supervisor → workers.
Supervisor decides *which* workers to dispatch (marketing? legal? both?). Static edges can't express this.

Cap concurrency in the edge closure.
Unbounded fan-out is a rate-limit incident waiting to happen.

Sources: docs.langchain.com/oss/python/langgraph/use-graph-api#map-reduce-and-the-send-api · langchain.com/blog/3-years-of-graph-engineering-with-langgraph (dynamic transitions section).

Three specialised agents, one graph, one sprint contract per unit of work.

ROLE 01 · PLANNER

Expands the wish.

input: "build a DAW"

output:
· 16-feature spec
· product context
· visual design language

Prompted to be ambitious about scope but stay high-level. If it hardcodes technical detail technical detail early, errors cascade.

Gets access to: frontend-design skill, product knowledge, prior specs.

Doesn't decide: file structure, library choices, sprint order.

ROLE 02 · GENERATOR

Builds one feature at a time.

stack: React + Vite
+ FastAPI + Postgres

unit of work: one sprint
artefact: code + tests + git

Sprint contract before code. Generator + evaluator agree on the exact testable criteria for testable criteria for "done." Only then does code get written.

Self-evaluates first. Runs its own tests before handing off to QA — cheap iteration is worth it.

Git for version control — every sprint is a branch.

ROLE 03 · EVALUATOR

Skeptical by design.

tools:
· Playwright MCP
· HTTP client for API tests
· DB introspection

threshold: any FAIL = retry

Different system prompt from generator — tuned to be skeptical, not generous. Same model, different persona.

Drives the app in Playwright — real clicks, real DOM, real API calls. Static-screenshot review is not enough.

Files specific findings — line numbers, function names, exact failure. Not "it's a bit rough."

Source: anthropic.com/engineering/harness-design-long-running-apps (Prithvi Rajasekaran, Anthropic Labs Q2 2026). Measured: 3-agent harness $124, 3h50m to ship a browser DAW that a solo agent couldn't complete.

stop doing these

The demos love these. Production hates them. Kill on sight.

ANTI-PATTERN 01

The god-prompt.

Symptom: a 5,000-line AGENTS.md.

Fix: ~100 lines of table-of-contents. Push detail into referenced docs, skills, and MCPs.

ANTI-PATTERN 02

Self-grading agent.

Symptom: generator asked "is this good?" answers "yes." Every time.

Fix: a separate evaluator, different system prompt, real tools. Never the same agent.

ANTI-PATTERN 03

Unbounded loop.

Symptom: $600 bill, 8 hours later, still "working."

Fix: hard ceiling on iterations, tokens, wall-clock. Circuit-breaker at 3× budget.

ANTI-PATTERN 04

Every task in a graph.

Symptom: a 60-node graph for a task the model could do in a while-loop.

Fix: "simplest first" — use a graph only where structure is known and reused.

ANTI-PATTERN 05

No traces, no truth.

Symptom: "our agent works great" — with zero recorded runs.

Fix: log every tool call, every message, every retry. Loop 4 depends on it.

ANTI-PATTERN 06

Sandbox = laptop.

Symptom: agent has `rm -rf` access to your $HOME.

Fix: ephemeral per-worktree sandbox. Allow-list commands. Network isolation. Never `danger-full-access`.

Compiled from: openai.com/index/harness-engineering (god-prompt) · anthropic.com harness-design-long-running-apps (self-grading) · Fowler harness-engineering (traces) · Anthropic building-effective-agents (simplest-first).

tick everything before demo

If you can tick these twelve boxes, you don't have an agent demo — you have a shipping product.

HARNESS + LOOP

☐ 01 AGENTS.md ≤ 150 lines · progressive disclosure for the rest

☐ 02 Filesystem + git available to the agent

☐ 03 Ephemeral per-task sandbox · allow-list commands

☐ 04 Hard ceilings: max iters, max tokens, max wall-clock

☐ 05 Compaction at 80% window · tool-call offload for >50k output

☐ 06 Verification loop wrapping the agent loop (separate evaluator)

GRAPH + OPS

☐ 07 Graph *only* where structure is known + reused

☐ 08 State schema is a TypedDict · reducers explicit per field

☐ 09 Human-in-the-loop node before any irreversible action

☐ 10 Every run emits a trace — tool calls, tokens, latency

☐ 11 Hill-climbing loop scheduled (weekly trace-review agent)

☐ 12 Cost-per-task ≤ target · alarm at 3× budget

"The model contains the intelligence. The harness is the system that makes that intelligence useful." — LangChain, Q2 2026. Primary sources for this deck: langchain.com/blog · anthropic.com/engineering · openai.com/index · martinfowler.com/articles.