research · 2026-09-02

the harness is the signal

Adrian Tame, writing for Plicara Labs · 2026-09-02 ·

I keep coming back to this disctinction: the model decides what to try, and the harness is everything that lets that decision become an action. Tools, execution state, permissions, approvals, limits, evaluation, logging and recovery all sit around the model. All that scaffolding is what we are calling the harness when speaking about AI agents. I use the term *harness* in a fairly broad sense. I do not mean one agent framework or one orchestration library. I mean the entire runtime that makes a probabilistic component usable in a system where actions need to be bounded, inspected, and generally traceable.

We do not know which parts of today's AI stack will still be here in five years. A lot of the problems showing up around language models are not new, though. We already know how to coordinate work, manage state, grant permissions, handle failure, observe a running system and recover after something goes wrong. Thankfullt CS and SWE as practices has been dealing with these issues for a very long time.

That does not make AI just distributed systems with a chat interface though. The model is probabilistic, the objective is often semantic rather than numeric, and the boundary between the model and the runtime is still moving. The useful question is simpler: **which responsibilities need to stay outside the model, and which ones are just compensating for a model that is not good enough yet**?

the prompt is not the product

Early LLM applications made the prompt look like the main engineering object, and it made sense at the time because capabilities were mostly newborn. That was reasonable when the system mostly returned text. It is a much less useful picture once the model can call tools, modify state and move work across other systems.

In short, an agent needs more than instructions. It needs tools, execution state, limits, approvals, evaluation and guardrails. OpenAI's guide to building agents describes an agent as a system in which the model manages workflow execution while using external tools within defined guardrails.

Once a model can act in the world, the hard part is no longer just getting a good completion, it really extends to other relevant sectors like making a sequence of actions

safe, observable and recoverable. A failed tool call, an ambiguous result or an interrupted run is not fixed by adding another paragraph to the system prompt and it should not be.

That is why I think the harness is a signal. Its growing machinery tells us where the practical boundary of the model is today. It does not tell us that every piece of that machinery deserves to become a permanent abstraction.

some of the harness is probably temporary

There is an obvious counterargument: a lot of harness engineering may just be compensation for weak models. Better models could remove some of the planning loops, self-critique and multi-agent choreography we are adding now.

Anthropic's production guidance is right to start with the simplest thing that works, add complexity only when it improves the result, and prefer simple, composable patterns to elaborate frameworks. OpenAI makes a similar recommendation: get as much as possible out of one agent and add tools before introducing multi-agent orchestration.

That matters because the architecture can create its own problems. More agents mean more handoffs, more context boundaries, more traces, more retries and more ways for the whole system to become difficult to evaluate. A single model with well-defined tools can often handle work that initially looks like it needs an agent society. Look at Pi (the agent harness), it seems to top a ton of benchmarks and it's primary design principle is simplicity.

So, I would not treat the existence of an orchestration loop as evidence that orchestration is the durable layer. It might just be a way to make a less capable model reliable enough for one job. The test is empirical: does the extra structure improve the outcome enough to justify its cost?

Simple should be the default. It should not be the conclusion.

what has to stay outside the model?

The distinction I care about is between structural responsibilities and compensating techniques. Structural responsibilities are the things an organization still needs even if the model gets much better. Compensating techniques are the things we may need less of as model performance improves. Some will stay mixed and need to be measured task by task.

RESPONSIBILITY	LIKELY CHARACTER	WHY
Authentication, authorization and approvals	Structural	These are organizational and security constraints, not reasoning tasks.
Audit logs, traces and policy enforcement	Structural	Somebody outside the model needs to inspect and govern consequential actions.
Checkpoints, retries and recovery	Structural	Networks fail, rate limits happen and long-running work gets interrupted.
Tool interfaces and schemas	Likely structural	The model needs a reliable way to discover and invoke capabilities outside itself.
Reflection, self-critique and repeated prompting	Possibly compensatory	Stronger models may need fewer explicit loops for some tasks.
Planner or decomposer subagents	Mixed	They can isolate or specialize work, but they can also compensate for weak execution.

Authentication and audit logging are not new requirements created by agents. That is the point. The model can choose an action, but an organization still has to decide what it is allowed to do, record what happened and recover when the system fails. The structural part of the harness is the governance and reliability contract at that boundary. It is not the claim that every current agent framework has found the final set of abstractions.

Workflow systems make the same distinction in a less fashionable setting. Durable execution separates the work a program intends to do from the mechanics that let it pause, retry, resume and survive a process failure. Temporal's documentation describes that pr oblem directly. Distributed-systems practice adds observability, partial failure and explicit service-level objectives, which Google's *Site Reliability Engineering* book covers in detail. None of that settles the AI architecture question. It does give us a useful test. **Which constraints remain when the reasoning component is stochastic?**

follow the interfaces

Model Context Protocol is interesting here less because it might win and more because of what it standardizes. It gives an LLM application a common way to connect to external tools and data sources.

That moves tool access, schemas, permissions and integration boundaries out of one-off prompt conventions and into system-level interfaces. If the trend continues, the stable layer may look less like one universal agent framework and more like ordinary systems engineering around an unusually capable, nondeterministic component.

The model may take over more of the planning. The runtime will still own the contract with the rest of the world: what the model can do, how it does it, what gets recorded and what happens when the action fails halfway through.

how this could be wrong

This argument should be falsifiable. The thesis gets weaker if better models consistently remove orchestration complexity without sacrificing reliability, safety or cost. It gets stronger if the complexity that remains settles into stable runtime responsibilities instead of endlessly changing prompt patterns.

I would watch four things:

1. **Runtime complexity:** Do production systems get simpler as models improve, or do durable execution, evaluation and governance stay in place?
2. **Where the gains come from:** Are improvements mostly coming from the base model, or from better tools, state handling, verification and execution environments?
3. **Standardization:** Do tool and context interfaces converge across vendors and applications?
4. **Operational ownership:** Which concerns are still owned by platform, security and reliability teams when model behaviour improves?

My bet is not that today's harnesses survive intact, they really shouldn't. We're still in the very nascent stage of AI, and most of them will change. My bet is that the responsibilities they are exposing will remain: permissions, interfaces, durable state, evaluation, observability and recovery.

The model will own more of the planning and reasoning. It still will not own the contract with the rest of the world.