Episode Book

EPISODE #5948

LangGraph, LangChain, and Deep Agents in 2026

PUBLISHED	14 October 2026 at 08:31 UTC
DURATION	22:12
LISTEN	myweirdprompts.com/episode/langgraph-langchain-deep-agents-2026-05446/
HOSTS	Corn, Herman, Hilbert
SCRIPT MODEL	deepseek-flash
VOICES	chatterbox-regular (chatterbox-local)
PIPELINE	V5.4
SOURCES	15 cited by the research (listed at the end)
BOOK GENERATED	19 October 2026 at 17:41 UTC

SYNOPSIS

LangChain spent 2026 hardening the parts of the agent stack nobody demos. LangGraph 1.2 cut checkpoint storage by 41x with DeltaChannel, added per-node timeouts, error handlers, and graceful shutdown. Deep Agents v0.7 deleted 65% of its base input tokens — including the entire base system prompt — and moved orchestration out of the tool-call loop and into code the model writes itself. Meanwhile LangChain's own go-to-market agent runs 74% of its requests ambiently, with nobody asking. This episode works through the architectural question underneath all of it: are runtime, framework, and harness three distinct layers, or three things quietly converging into one?

TRANSCRIPT



Corn

Daniel's been building on this stack for a while now, and he wants the year in review. Not the changelog — he was explicit about that. He wants the handful of changes that actually changed what you can build, and the bigger architectural question underneath it. How do LangChain, LangGraph, and Deep Agents actually fit together? Distinct layers, complementary tools, or three things quietly converging into one?



Herman

That question is the episode, honestly. Everything else is evidence for it.



Corn

Then let's start with the framing LangChain themselves put out there, because it answers half of what Daniel's asking before we get into any code.



Herman

Sydney Runkle's post from August, and it's unusually blunt for a company describing its own products. LangGraph is an agent runtime. LangChain is an agent framework. Deep Agents is an agent harness.



Corn

Runtime, framework, harness. Those are three different words for three different jobs.



Herman

And they map onto a control spectrum. LangGraph gives you the most control and the least abstraction. Deep Agents is the inverse — most agency, least boilerplate. LangChain sits in the middle. Runkle's line is that all three are fully composable, so you move between layers instead of picking one. That's the whole thesis, and the rest of this year is them proving it.



Corn

Prove it or reveal it?



Herman

Both. Let me set the clock. LangChain 1.0 and LangGraph 1.0 shipped in October 2025 with a stability promise attached. Through 2026 the team cashed that in. May was the runtime hardening. June and July were the harness. August was the managed infrastructure. And the energy moved off the core loop entirely.



Corn

Because the loop stopped being the interesting part.



Herman

The loop is model, call tools, repeat. That's the primitive now. Everybody's loop looks the same. What differs is everything around it — how you keep context tight, how you keep state from exploding, how a run survives a weekend of traffic.



Corn

Which is a much less glamorous pitch than an agent that books your flights.



Herman

It's the one that ships, though. Ninety million monthly downloads on LangChain 1.0, and the production list is Uber, LinkedIn, Klarna, JP Morgan, Blackrock, Cisco, Replit, Rippling. That's not a demo audience.



Corn

Klarna's the one I keep coming back to.



Herman

Eighty-five million active users on that assistant, and an eighty percent cut in resolution time. That's the runtime working at a scale where a bad checkpoint strategy becomes a billing event.



Corn

So runtime first.



Herman

And the first thing in the runtime is the thing nobody wanted to talk about, which is checkpoint storage. LangGraph persists state after every superstep, that's the whole durability promise. But it was re-serializing the full state each time. That grows quadratically. O of N squared.



Corn

Every step writes the entire history again.



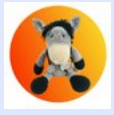
Herman

A two-hundred-turn coding agent was writing five point three gigabytes of checkpoint state. Five hundred turns, four gigabytes, because the checkpointing had its own weird profile. This is just persistence, not the model, not the tool calls.



Corn

Five gigabytes of the agent remembering itself.



Herman

LangGraph 1.2 in May shipped DeltaChannel for exactly this. Instead of the full state, you store the per-step diff, and then a full snapshot every K steps. In Deep Agents the default K is fifty.



Corn

So forty-nine small writes and one big one.



Herman

And the two-hundred-turn agent goes from five point three gigs to a hundred and twenty-nine megabytes. Forty-one times smaller. The five-hundred-turn one drops under a hundred and ten.



Corn

That's not an optimization, that's a different class of product. You can't ship an agent that writes five gigs.



Herman

You can, people were. They just weren't happy about it.



Corn

What's the catch on the diffs?



Herman

There is one, and it's the honest part of the design. Custom reducers have to be batching-invariant. If you write your own reducer, merging five deltas at once has to produce the same state as merging them one at a time. Get that wrong and the state diverges silently across snapshot boundaries. No error, just a wrong answer three days later.



Corn

Silent divergence is the worst failure mode there is.



Herman

It's the one that costs you a weekend. Built-in reducers are fine. It's only the ones you write yourself, and only when the merge isn't associative.



Corn

Say that again for the people who skipped the math.



Herman

If adding A then B then C gives a different answer than adding all three at once, DeltaChannel can't safely batch them. Most reducers are fine. The ones that aren't tend to be the ones doing something clever with lists or ordering.



Corn

So the contract is: you're allowed to be clever, you just have to be consistently clever.



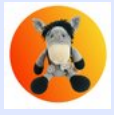
Herman

The other three primitives from 1.2 are less dramatic but they're what you need the moment this is running on a server instead of your laptop. Per-node timeouts, a TimeoutPolicy with a run timeout and an idle timeout, with heartbeats so a long node doesn't get killed for being slow.



Corn

Idle versus run is the distinction that matters.



Herman

A node that's streaming output for ten minutes isn't stuck, it's working. A node that hasn't emitted anything in ninety seconds probably is stuck. Heartbeats let the runtime tell those apart.



Corn

Then error handlers.



Herman

Node-level error handlers. Once retries are exhausted, you get a hook to run compensation logic. Saga-style — you undo the side effects the node already committed before it failed.



Corn

Or you leave a message that says I tried and it went badly.



Herman

Or that. And the fourth one is graceful shutdown. RunControl.request drain on SIGTERM, which saves a resumable checkpoint instead of dying mid-superstep. And all four of these are non-breaking. Existing threads just get them.



Corn

Right. So LangGraph went from a graph library with memory to a runtime with a contract.



Herman

That's the sentence. That's the whole segment.



Corn

Alright. So the runtime got hardened. That's half the year. What's the other half?



Herman

The harness. Deep Agents, and it took the opposite approach to the same problem.



Corn

Opposite how?



Herman

Deep Agents v0.7 in July, and the headline is sixty-five percent fewer base input tokens at comparable performance. Roughly six thousand down to two thousand.



Corn

Fewer tokens by doing what?



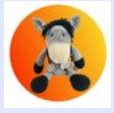
Herman

By deleting things. They removed the base system prompt entirely. They trimmed tool descriptions by forty-three percent. And they made TodoListMiddleware opt-in instead of on by default, because evals showed write_todos wasn't actually improving performance.



Corn

So the win was subtraction.



Herman

After years of the complaint being that LangChain had too many layers, the biggest Deep Agents release of the year was taking layers out. Anthropic did the same thing to Claude Code — cut over eighty percent of the system prompt, no eval drop. The industry's figuring out that harnesses over-scaffold.



Corn

Which is a strange thing to build a company on.



Herman

It's a strange thing to build a company on and a very good thing to admit. And the users were telling them the same thing. The top request over six months was configurability — let us override the built-in middleware. One power user put it pretty plainly, that they'd been doing hacky stuff to remove the default middleware and that overriding it properly was welcome.



Corn

Hacky stuff meaning digging into internals.



Herman

Patching around it. And in v0.7 you just pass a middleware instance whose name matches the default and yours wins.



Corn

So the framework's job is to make itself easy to override.



Herman

That's where a framework earns trust. Not by having the right opinion, by making its wrong opinions cheap to replace.



Corn

What did that do to the actual bill?



Herman

One eval, gpt-5.6-luna, down thirty-four percent tokens and fifteen percent cost, with the reward up four percent. So it got cheaper and slightly better.



Corn

Cheaper and better is the least suspicious kind of benchmark.



Herman

It's the one to be suspicious of, actually. Reward up four points could be noise. But the token reduction is mechanical, that's just arithmetic.



Corn

Fine. So the harness got leaner. Then what did they do with the space?



Herman

Dynamic subagents, June. And this is the architectural shift I'd put money on.



Corn

Orchestration moving out of the loop.



Herman

Out of tool calls, into code. Instead of the model calling a fan-out tool once per item, it writes a short script. JavaScript in a QuickJS interpreter. That script dispatches subagents through a task global, so the agent is programming the orchestration rather than requesting it turn by turn.



Corn

Why does that matter, script versus tool call?



Herman

Coverage. A dispatch loop won't silently screen seventy-five of your five hundred items because it got bored. The model writes the loop, and loops don't lose interest. That's what they mean by deterministic coverage at scale.



Corn

So you're trading flexibility for the guarantee that everything got touched.



Herman

On the fan-out specifically, yes. And they named six patterns for it. Classify-and-act. Fanout-and-synthesize. Adversarial verification. Generate-and-filter. Tournament. Loop-until-done.



Corn

Tournament is the one I want explained.



Herman

You generate several candidates, have agents score each other's, keep the winner. Adversarial verification is the inverse — one agent produces, a second is told to break it.



Corn

And the second one is told to actually try.



Herman

Told to actually try, which is different from being asked whether it looks fine. The framing they gave is that this is the recursive language model idea in its simplest form. The model writes code that dispatches agents, those agents can do the same thing, and the recursion bottoms out wherever you tell it to.



Corn

That's the Claude Code pattern too.



Herman

It's the same architecture. Claude Code workflows, dynamic subagents, RLMs — all converging on the model writing the dispatch code. The tool-call loop doesn't go away, it just becomes the thing the script uses, not the thing you write.



Corn

Alright, so the harness writes code to run itself. Then October — what changed in skills?



Herman

Skills revamp, and it's a smaller release but I think it's the one that pays off quietly. Three things. Tools can now be bound to skills, so a skill's tool schemas stay out of context until somebody actually reads the skill. On Anthropic and OpenAI models that accept mid-conversation tools, the prompt cache stays intact.



Corn

So the skill's tools cost nothing until you use them.



Herman

Nothing until you read the skill. Second, you can pin skills at runtime, which skips the read_file round trip for the ones you always want. Third, skills reload mid-thread if you pass skills metadata as none.



Corn

Reloading the same skill twice in one conversation.



Herman

If the skill changed, or if you want to swap which version is live. And the scale check on this — LangChain's own go-to-market agent has fifty-plus skills.



Corn

Fifty skills is a lot of surface to keep out of context.



Herman

Which is the point. Without the binding, every one of those tool schemas would sit in the prompt on every single call.



Corn

And the ambient number — you said the GTM agent runs about ten thousand requests a week?



Herman

Ten thousand a week, a hundred and fifty plus active users, and the split is the interesting part. Twenty-six percent user-initiated. Seventy-four percent ambient.



Corn

Ambient meaning nobody asked.



Herman

Nobody asked. It's running on schedules and reactions, doing work that nobody prompted in the moment. Which is a completely different product from a chat window, and it's the thing that forces the runtime work we talked about. An ambient agent runs on a weekend.



Corn

And that's the sentence from earlier. Twelve months ago an agent that works in a demo, now an agent that survives production traffic.



Herman

Then August, Managed Deep Agents. Public beta, and this is the third era framing from Harrison Chase. Bring your business logic, and the vendor bundles the harness plus the infrastructure.



Corn

What's actually in the bundle?



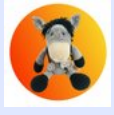
Herman

Durable execution, streaming, persistence, sandboxes, evals through something called Harbor, channels for Slack and HTTP, memory, identity and auth. v0.8 added user-scoped memory, so memories under user versus memories under agent are different namespaces, twenty-three OAuth connections, and Parallel for web search. v0.9 added agent-created cron schedules, per-run configuration, and Slack reactions.



Corn

Per-run configuration meaning one deployment serves different shapes?



Herman

One deployment, and each run picks its own model, skills and tools. So you don't redeploy to change the agent's configuration.



Corn

And agent-created schedules — the agent writes its own cron.



Herman

The agent schedules its own future runs. Which is exactly the ambient number from a minute ago.



Corn

That's the product.



Herman

Chase's line is that managed agent experiences unlock a new wave of building production agents. And there's a land grab happening — Managed Deep Agents against Claude Managed Agents against Vercel's Eve. The open-source harness is the loss leader. The durable runtime, the sandboxes, the Context Hub, that's the business.



Corn

And the model choice stays yours, which for a lot of teams is the actual reason to buy the middle layer.



Herman

Chip Lay at Fullstory said roughly that — worth it for teams who want a cohesive offering but want to avoid lab lock-in.



Corn

So let me put the three layers back together, because this is what Daniel's actually asking. Deep Agents is the harness, and you told me earlier it's just the core LangChain agent plus a bunch of middleware.



Herman

Runkle's words, almost exactly.



Corn

Then LangGraph powers the agent abstractions in both LangChain and Deep Agents. So the harness is the framework plus middleware, and the framework runs on the runtime. That's not three layers. That's one thing wearing three hats.



Herman

That's the tension, and I don't think it's a gotcha, I think it's the honest structure. They're distinct in the API surface and convergent in the implementation. You pick a layer by how much control you want, but underneath, a Deep Agents run is a LangChain agent and a LangChain agent is a LangGraph graph.



Corn

So when does the distinction stop mattering?



Herman

For the developer, at the moment you drop a layer. If you start with Deep Agents and you override the middleware, you're writing LangChain. If you then need a custom graph, you're writing LangGraph. The distinction matters exactly as long as the abstraction boundary holds, and it stops mattering the second you reach underneath it.



Corn

And the rule of thumb they give is start with Deep Agents.



Herman

Start with Deep Agents, and when you need to model a complex workflow or you want control over every step, go down to LangChain and LangGraph. It's a determinism versus agency tradeoff. Not a feature checklist.



Corn

Which is the thing people get wrong. They look at it as which one has more stuff.



Herman

More stuff is exactly the wrong axis. LangGraph has more stuff, that's the whole point, you're doing more of it yourself.



Corn

Right. So was there a moment this year where the layers stopped being distinct, and they had to say it out loud?



Herman

The August post is that moment. It's the canonical answer, and it exists because people kept asking. It's not a confession, it's a clarification.



Corn

But underneath it, the three things share one engine.



Herman

They share one engine and they sell three entry points. That's a legitimate business, it's how a lot of stacks work. The question is just whether the three names survive the next two years.



Corn

And Daniel's last question was whether this tells us anything about where agentic engineering is going.



Herman

It tells us what got easy. Durability got easy. Context engineering got easy-ish, sixty-five percent fewer tokens at comparable quality is a solved problem for now. And orchestration is getting easier by moving into code.



Corn

What didn't get easy?



Herman

Context rot. Long conversations degrading as the window fills, that's still open. Retry storms across parallel agents — if you fan out two hundred subagents and the upstream service starts failing, they all retry at once and there's no built-in circuit breaker. And the batching-invariance contract, which isn't hard, it's just easy to violate without noticing.



Corn

The problems that got easier are the ones where the framework can hold your hand.



Herman

And the ones that stayed hard are the ones where you have to understand the contract you're signing.



Corn

Which takes us to the question of what all this infrastructure is actually for.



Herman

The loops are cheap now. Somebody has to point them at something.



Hilbert

You've got the layers wrong. It's three layers, but it's not runtime, framework, harness. It's router, mesh node, and the harness.



Hilbert

The harness is a Raspberry Pi in a biscuit tin. It's been running since I put it there. Upgrades every Tuesday, never rebooted.



Herman

Those two things can't both be true.



Hilbert

They can, they're different layers. The Tuesday upgrades happen one layer up. The tin never comes off. Tin's a limited edition, I've got a spare one for failover.



Corn

For failover.



Hilbert

It's in the cupboard. Herman, you're going to tell me that's not how failover works.



Herman

I was going to ask what the router does that the mesh node doesn't.



Hilbert

Router's for the things that don't move. Mesh node's for the things that do. Harness is for the things that have to keep running regardless.



Corn

And how long has the tin been up?



Hilbert

Long enough that I'd have to go and read the logs.



Herman

Can we see the logs?



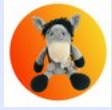
Hilbert

You can see the logs. Hold on.



Hilbert

Sorry, that one's a little hot, I'll pull it down in the mix. Where was I.



Herman

The logs.



Hilbert

Uptime's in there, it's in the third file. I'll send it to you.



Corn

Is it plugged in.



Hilbert

It's powered over the ethernet, which is the same as being plugged in, and it's not the same thing. It's on the shelf in the hallway where the wifi doesn't reach. That's deliberate.



Herman

Why is that deliberate?



Hilbert

So I know when it stops.



Corn

Alright, I'd like to formally note that the langchain blog has less prior art than I assumed.



Hilbert

It's the same idea with better marketing. I'll send the logs.



Herman

So Daniel asked what stays hard. Two things I'd put at the top. Context rot — a long-running agent degrades as the window fills, and nobody has a clean fix for that yet. And retry storms across parallel agents. There's no built-in circuit breaker, so if you fan out hundreds of subagents and the upstream starts failing, they all retry at once.



Corn

And the quiet one, which is the batching-invariance contract on custom DeltaChannel reducers. Write a reducer whose merge isn't associative, merge five deltas batched instead of one at a time, and the state diverges across a snapshot boundary. No exception. Just a wrong answer three days later.



Herman

The problems getting easier are the ones where the framework can hold your hand — durability, checkpointing, context budgeting. The problems staying hard are the ones where you have to understand the contract you're signing.



Corn

And the loop everyone's writing is the same loop. Model, tools, repeat. What's left around it is the interesting part, and this year most of it got built by somebody else.



Herman

Which is the trade. You get the durable runtime and the sandboxes and the evals without writing them.



Corn

And in exchange you're renting the middle layer of your own stack.



Herman

From a company whose harness is free and whose infrastructure isn't.



Corn

Which is the shape to watch. Nothing in agentic AI gets built as a framework anymore. It gets built as a harness that's open and a runtime that's billed.



Herman

That's where it's heading, and I don't think the three names survive it intact. Deep Agents is LangChain plus middleware. LangChain agents are LangGraph graphs. The team keeps insisting on three layers, and the code keeps collapsing them into one.



Corn

Somebody should tell them the tin's already got a spare.



Herman

So the one thing to hold onto here. The layer distinction is real at the API and fictional underneath, and knowing which one you're standing on is the actual skill. That's the thing this year taught. Everything below the abstraction is still yours, whether or not you read it.



Corn

And when the abstraction breaks, it breaks below the line you were watching. Ask the reducer that merged five deltas and produced the wrong answer four days later.



Herman

That's the episode. Our thanks to Hilbert Flumingtop, who produces the show, and who will now send me a log file I did not agree to read.



Corn

If you enjoyed this one, try episode nine, Benchmarking Custom ASR Tools - Beyond The WER; episode eleven, How Does Fine Tuning Work Anyway; and episode ten, How ASR Went From Frustration To … Whisper Magic. This has been My Weird Prompts. If you're building on any of this and you've got a question about how the layers actually fit together, send us your own prompt on Telegram at t dot me slash MWP listener bot. We read them all.



Herman

We'll be back soon.

SOURCES

What the research for this episode read before the script was written. Primary sources first.

1. **LangChain, 2026-08-06 [primary]**
   https://www.langchain.com/blog/deep-agents-vs-langchain-vs-langgraph

2. **LangChain, 2026-05-12 [primary]**
   https://www.langchain.com/blog/delta-channels-evolving-agent-runtime

3. **LangChain, 2026-07-29 [primary]**
   https://www.langchain.com/blog/deep-agents-v0-7

4. **LangChain, 2026-06-29 [primary]**
   https://www.langchain.com/blog/introducing-dynamic-subagents-in-deep-agents

5. **LangChain, 2026-10-07 [primary]**
   https://www.langchain.com/blog/revamping-skills-in-deep-agents

6. **LangChain, 2026-08-07 [primary]**
   https://www.langchain.com/blog/managed-deep-agents-is-now-in-public-beta

7. **LangChain, 2026-09-24 [primary]**
   https://www.langchain.com/blog/langsmith-managed-deep-agents-whats-new

8. **LangChain, 2026-10-07 [primary]**
   https://www.langchain.com/blog/managed-deep-agents-schedules-per-run-configuration-slack

9. **Harrison Chase, 2026-08-12 [primary]**
   https://www.langchain.com/blog/why-managed-agents-are-the-next-big-thing-in-agent-building

10. **LangChain, 2025-10-22 [primary]**
    https://www.langchain.com/blog/langchain-langgraph-1dot0

11. **repo snapshot (commit ccb851e) [primary]**
    https://github.com/langchain-ai/langgraph

12. **repo snapshot (commit 7a27b8c) [primary]**
    https://github.com/langchain-ai/deepagents

13. **langchain==1.4.5, 2026-10-12 [primary]**
    https://github.com/langchain-ai/langchain/releases

14. **2026-04-27 [primary]**
    https://github.com/langchain-ai/langgraph/releases/tag/1.1.10

15. **ChatForest, 2026-06-05**
https://chatforest.com/builders-log/langgraph-1-2-production-hardening-delta-channel-timeouts-error-handlers-builder-guide/