Context Engineering for Agents - Lance Martin, LangChain

63 MIN · YOUTUBE · HTTPS://YOUTU.BE/_ILTCWCIEC4?SI=HX8R-NYX4NXSVWK0
https://youtu.be/_IlTcWciEC4?si=Hx8r-NyX4NXsvWK0

SUMMARY

Lance Martin, founder of Lang Chain, discusses the emerging concept of context engineering in AI, particularly in the development of agents. He highlights the challenges of managing context effectively as agents interact with various tools, emphasizing the importance of offloading and summarizing context to optimize performance and reduce costs.

- *Context engineering is crucial for feeding language models the right context during agent operations, especially as tool calls increase.*
- *Prompt engineering is a subset of context engineering, with agents requiring more complex context management than chat models.*
- *Offloading context to external storage (like disk) can significantly reduce token costs and improve efficiency.*
- *Summarization of context is vital, but it must be done carefully to avoid information loss, especially in coding tasks.*
- *The use of multi-agents can enhance context isolation, but careful management is required to prevent conflicting decisions.*
- *Caching can improve latency and cost but does not solve the long context problem inherent in agent operations.*
- *The "bitter lesson" in AI suggests that simpler, more general approaches often outperform complex, structured ones as model capabilities improve.*
- *Continuous reassessment of assumptions and structures in AI applications is necessary to leverage advancements in model performance effectively.*

Hey everyone, welcome to the latest space podcast. This is Allesio, founder of Kernel Labs, and I'm joined by Swixs, founder of Small AI. >> Hello. Hello. Uh, we are so happy to be in the remote studio with Lance Martin from Lang Chain Langraph and everything else he does. Welcome. >> It's great to be here. I'm a longtime listener of the pod and uh is finally great to be on. >> Yeah. um you you've been uh part of uh you know our orbit for a while. You you spoke at uh one of the AIES uh and also obviously we're we're pretty close with with Lang Chain. Um I think uh recently though you know I think you you've you've also like been doing a lot of tutorials. I remember you did like 01 deep researcher sorry R1 deep researcher which is uh a pretty popular project um and uh an async ambient agents. Uh but the thing the thing that really sort of prompted me to reach out and say like okay it's finally time for the Lance Martin pod is your recent work on context engineering which is all the rage. Uh how'd you get into it? Well, you know, it's funny. Um, buzzwords emerge often times when people have a shared experience. And I think lots of people started building agents kind of early this year, mid this year, quote unquote, the year of agents. And I think kind of what happened is when you put it put when you kind of put together an agent, it's just tool calling in a loop.

It's relatively simple to lay out, but it's actually quite tricky to get to work well. And in particular, managing context with agents is a hard problem. Carpathy put out that tweet kind of canonizing the term context engineering and he kind of mentioned this uh kind of nice definition which is context engineering is the challenge of feeding an LM just the right context for the next step which is highly applicable to agents and I think that really really resonated with a lot of people and I in particular had that experience over the past year working on agents and I wrote about that a little bit in my piece talking about building open deep research over the past here. So I think it's it was kind of an interesting point that the term kind of captured a common experience that many people were having and it kind of took hold because of that.

>> How do you define the lines between prompt engineering and like context engineering? So is the prompt optimization like context engineering in your mind? Like I think people are kind of confused like are we replacing the the term like what what is it? >> Well, I think that you know prompt engineering is kind of a subset of context engineering. I think when we kind of moved from chat models and chat interactions to agents there was a big shift that occurred. So with chat models working with chat GBT the human message is really the primary input and of course a lot of time and effort is spent in crafting the right message that's passed to the model. With agents the game is a bit trickier though because the agents getting context not just from the human but now context is flowing in from tool calls during the agent trajectory. And so I think this was really the key challenge that I observed and many people observed is like oof um when you put together an agent, you're not only managing of course the system instructions, system prompt and of course user instructions, you also have to manage all this context that's flowing in at each step over the course of a large number of tool calls. And I think there's been a number of good pieces on this. Manis put out a great piece talking about context engineering with Manis and they made the point that the typical Manis task is like 50 tool calls.

Anthropics multi-agent research is another nice example of this. They mentioned that the typical production agent and this is probably referring to Claude Code could be other agents that they've produced is like hundreds of tool calls. And so when I had my first experience with this, I think many people have this experience. You put together an agent. You're sold the story. That's just tool calling in a loop. That's pretty simple. You put it together. I was building deep research.

These resource tool calls are pretty tokenheavy. And suddenly you're finding that my deep researcher, for example, with a naive tool calling loop was using 500,000 tokens. It was like a dollar to $2 per run. I think this is an experience that many people had. And I think it's kind of that the challenge is realizing that o building agents is actually a little bit tricky because if you just naively plum in the context from each of those tool calls naively you just hit the context window of the LM that's kind of the obvious problem but also Jeff from Chroma spoke about this on the recent pod. There's all these weird and idiosyncratic failure modes as context gets longer. So Jeff has that nice report on context rot. And so you have both these problems happening. If you build a naive agent, context is flowing in from all these tool calls. It could be dozens to hundreds.

And there's degradation in performance with respect to context length. And also the trivial problem of hitting the context window itself. So this was kind of I think the motivation for this new idea of actually it's very important to engineer the context that you're feeding to an agent. And that kind of spawned into a bunch of

different ideas that I put together in the blog post that people are using to handle this um drawn from anthropic from my own experience from Manis and others.

>> So I'm just going to put some of the um relevant materials on screen just because I we like to you know part this we like to have some visual aid. Uh we did our post on GT5 and we call it thinking with tools. So where you know you part of part part of the tools is to get context. Um and I think using tools to to obtain more context like the agent can figure out what context it needs and if you just tell it to. Uh and then the other one is actually I thought you would you did a blog post on this but apparently it was just like that this is it. Um I will say it's funny and actually I was hoping you'd bring this up. I also have a blog post but it's all moving so quickly >> that I did a meet up after the blog post and updated the story a little bit with this meetup. So actually this is a better thing to show but I do have a blog post too but things change between my blog post and the meetup which were like two weeks apart. So that's how quickly these things are moving.

Exactly. That's the blog post. >> Should we should we do this sequentially then? Um >> I think it's actually okay to just hit the meetup >> because that's it's just easier to follow one thing and it it's it's like a supererset of the blog post story. >> Okay. >> How do you define the five categories? So I mean I understand what offload kind of means but like can you maybe yeah go deeper. >> Yeah. Yeah. We should let's walk through these actually. So when I talked about naive agents and the first time I built an agent agent makes a bunch of tool calls. Those tool calls are passed back to the LM at each turn and you naively just plum all that context back. And of course what you see is the context window grows significantly because these this tool feedback is accumulating in your message history. So a perspective that manage shared in particular I thought was really good is it's important and useful to offload context. Don't just naively send back the full context of each of your tool calls. You can actually offload it and they talk about offloading it to disk. So they talked about this idea of using the file system as externalized memory rather than just writing back the full contents of your tool calls which could be token heavy.

Write those to disk and you can write back a summary. It could be a URL something so the agent knows it's retrieved a thing. It can fetch that on demand but you're not just naively pushing all that raw context back to the model. So that's this offloading concept. Um note that it could be a file system. It could also be for example agent state. So Lang graph for example has this notion of state. So it could be kind of the the agent runtime state object. It could be the file system. But the point is you're not just plumbing all the context from your tool calls back into the agents message history. You're saving it in externalized system. You're fetching it as needed. This saves token cost significantly. So that's the offloading concept. I I guess the question on the offloading is like what's the um minimum you know summary metadata or whatever you need to keep in the context to let the model understand what's in the offloaded context like if you're doing deep research obviously you're offloading kind of like the full pages maybe but like how do you generate like an effective summary or blurb about what's in the file so this is actually a very interesting and important point so I'll give an example from what I did with open deepep research so open deep research is a deep research agent that I've been working on for about a year and it's now according to deep research bench the best performing uh deep research agent um at least on that particular benchmark. So it's it's pretty good. Listen, it's not as good as open as deep research which uses end to end RL. It's all fully open source and it's pretty strong. So I just do carefully prompted summarization.

I try to prompt the summarization model to give kind of an exhaustive set of bullet points of the key things that are in the post just so the agent can know whether to retrieve the full context later. So, I think it's kind of prompting if you're doing summarization carefully for recall, compressing it, but like making sure that all the key bullet points necessary for the for the LLM to know what's in that piece of kind of full context is actually very important when you're doing this kind of summarization step.

Now, Cognition had a really nice blog post talking about this as well, and they mentioned you can really spend a lot of time on summarization, so I don't want to trivialize it, but at least my experience has been it's worked quite effectively. Prompt a model carefully to capture exactly. So, in this post, they talk a lot about even using a fine-tuned model for performing kind of summarization. In this case, they're talking about um agent agent boundaries and summarizing, for example, message history. But the same applies the same challenges apply to summarizing for example the full contents of tokenheavy tool calls so the model knows what's in context. So I basically spent a lot of time prompt engineering to make sure my summaries capture with high recall what's in the document but compress the content significantly. I I I do think that the compression that that was also part of the meetup findings of yesterday where we were we were at the context engineering meet up that Chroma hosted uh that you do want frequent compression because you don't want to hit the context raw limit. Um yeah, I don't I'm not sure there's much else to say like offloading is important and you should probably do it. the um there was also a really interesting link I guess somebody I think Dex um was linking it to the concept of multi-aents and why you why you do want multi- aents is because you can compress and load in different things based on the role of the agent and probably a single agent would not have all the context >> yeah you know so that's exactly right and actually one of the other big themes I hit and and talk about quite a bit is context isolation with multi- aent and I do think this does link back to the cognition take.

So which is interesting. So their argument against multi-agent >> is that it can be hard multi- aent. >> Correct. And what they're arguing is a few different things. One of the main things is that it is difficult to communicate sufficient context to sub aents. They talk a lot about um spending time on that summarization or compression step. They even use a fine-tuned model to ensure that all the relevant information. Yeah. So they actually show it a little bit down below as kind of a linear agent. But even at those agent agent boundaries, they talk a lot about being careful about how you compress information and pass it between agents.

>> Yeah, I think the the biggest question for me I mean uh coding is kind of like the the main use case that I have. Um, and I think I still haven't figured out how much of value there is in showing how the implementation was made to then write. If you have a sub agent that writes tests or you have a sub agent that does different things, how much do you need to explain to it about how you got to the place the codebase is in versus not? And then does it only need to return the test back in the context of the main agent? if it has to fix some code to match the task, should it say that to the main agent? I think that's kind of like it's clear to me like the deep research use case because it's kind of like atomic pieces of content that you're going through. But I think when you have state that depends between the sub agents, I think that's the thing is still unclear to me.

>> So I think that's one of the most important points about this context isolation um kind of bucket. So cognition argues which actually I think is a very reasonable argument. They argue don't do sub agents because

each sub aent implicitly makes decisions and those decisions can conflict. So you have sub agent one doing a bunch of tasks, sub aent two doing a bunch of tasks. They those kind of decisions may be conflicting and then when you try to compile the full result in your example with coding there could be tricky conflicts.

I found this to be the case as well and I think a perspective I like on this is use multi-agent in cases where there's very clear and easy parallelization of tasks. Cognition in Walden Yen spoke on this quite a bit. He talks about this idea of kind of read versus write tasks. So for example, if each sub agent is writing some component of your final solution, that's much harder. they have to communicate like you're saying and agent agent communication is still um quite early but with deep research it's really only reading they're just doing context collection and you can do a write from all that share context after all the sub aents work and I found this worked really well for deeper research and actually anthropic report on this too so their deep researcher just uses parallelized sub aents for research coalation and they do the writing in one out at the end. So this works great. So it's a very nuanced point that what you apply context isolation to in terms of the problem. Yeah. So you can see this is their work matters significantly.

Coding may be much harder. In particular, if you're having each sub agent create one component of your system, there's many potentially implicitly conflicting decisions each of the sub aents are making. When you try to compile the full system, there may be lots of conflicts. with research you're just doing context gathering in each of those sub aent steps and you're writing in a single step. So, I think this was kind of a key tension between the cognition take, don't do multi-agents, and the anthropic take. Hey, multi-agents work really well. It depends on the problem you're trying to do with multi-agents. So, this was a very subtle and interesting point. What you apply multi-agents to matters tremendously and how you use them. I like the take that apply multi- aents to problems that are easily paralyzable that are read only for example context gathering for deep research and do like the final quote unquote write in this case report writing at the end. I think this is trickier for coding agents. I did find it interesting that claude code now allows for sub aents. So they obviously have some belief that this can be done well or at least it can be done.

But I still think I actually kind of agree with Walden's take. it can be very tricky in the case of coding if sub agents are doing tasks that need to be highly coordinated. >> I think that's a a well u explained uh contrasting comparison. Um not much to add there. I think um it's interesting that they have different use cases and different architectures involved. Um I don't know if that's a permanent thing that that might fall to the bitter lesson as as uh you would put it.

>> Yes, we should probably talk about uh some of the other parts of the system that uh you set up. So um >> is that there's a lot of interesting techniques there. >> Well, let's talk about classic old retrieval. So rag is obviously uh it has been in the air for now many years obviously well before LLMs and this clamp whole wave. One thing I found pretty interesting is that for example different code agents take very different approaches to retrieval. Verun from windsurf shared an interesting perspective on how they approach retrieval in the context of windsurf.

So they use classic code chunking along carefully designed semantic boundaries embedding those chunks. So

classic kind of semantic similarity vector search and retrieval but they also combine that with for example GP. Then they do for example knowledge knowledge graphs. They they then also mention knowledge graphs. They then talk about combining those results doing reanking. So this is kind of your classic complicated multi-step rag pipeline. Now what's interesting is Boris from Enthropic and Cloud Code has taken a very different approach. He's spoken about this quite a bit. Cloud code doesn't do any indexing. It's just doing quoteunquote agentic retrieval.

just using simple tool calls, for example, using GP to kind of poke around your files, no indexing whatsoever, and it obviously works extremely well. So, there's very different approaches to kind of rag and retrieval that different code agents are taking. And this seems to be kind of an interesting and emerging theme like when do you actually need kind of more hardcore indexing? when can you just get away with simple just kind of a gentic search using very basic file tools?

>> Yeah, one of the more uh viral moments from one of our recent podcasts was Boris's partnered with us and Klein also mentioning that uh they just don't do code indexing, they just use agentic search. Um and yeah, probably like you know that's a that's a really good 8020. And then if you really want to fine-tune it, probably you want to do a little mix, but maybe you don't have to for for your needs. Yeah, I actually just saw a client posted uh I think yesterday talking about that they only use script, they don't do indexing. And so I think within the retrieval kind of area of context engineering, um there are some interesting trade-offs you can make with respect to are you doing kind of classic vector storebased semantic search and retrieval with a relatively complicated pipeline like Verun's talking about with Windsurf or just good old kind of agentic search with basic file tools. I will note I actually did a benchmark on this myself. I think there's a shared blog post somewhere. I'll bring it up right now. Yep. I actually looked at this a bit myself. Um, this was a while ago, but I compared three different ways to do retrieval on all Langraph documentation for a set of 20 coding questions related to Langraph. So, I basically wanted to allow different code agents to write Langraph for me by retrieving from our docs. I tested cloud code and cursor. I used three different approaches for grabbing documentation. So one was I took all of our docs around 3 million tokens. I indexed them in a vector store and just did classical vector store search and retrieval. I also used an LM.ext with just a simple file loader tool. So that's kind of more like the agentic search. Just basically look at this element.ext text file which has all of the URLs of our documents with some basic description and let the LM or the code agent in this case just make tool calls to fetch specific docs of interest and I also just tried context stuffing to take all the docs 3 million tokens and just feed them all to the code agent. So these are just some results I found comparing cloud code to cursor and interesting what I actually found this this is only my particular test case but I actually found that lm.ext text with good descriptions, which is just very simple. It's just basically a markdown file with all the URLs of your documentation and like a description of what's in that doc. Just that passed to the code agent with a simple tool just to grab files is extremely effective.

And what happens is the code agent can just say, "Okay, here's the question. I need to grab this doc and read it." It'll read it. I need to grab this doc, read it, read it. This worked really well for me and I actually use this all the time. So I actually personally don't do vector store indexing. I actually do lm.ext with a simple search tool with cloud code is kind of my go-to. Cloud code in this case. This was done a few months ago. These things are always changing. In this particular point in time, cla code actually outperformed cursor for my test case. That's

this actually cloud code pled me. And this was I did this back in April. So I've been kind of on cloud code since. Um but that was really it. So this kind of goes to the point that Boris has been making about claude code about inclined as well. You give an LM access to simple file file tools. In this case I actually use an LM.ext to help it out so it can actually know what's in each file is extremely effective and much more simple and easy to maintain easier to maintain than building an index. So that's just my own experience as well.

>> The scaled up form of LMS.ext which I really like and I use uh quite a bit is uh actually the the deep wiki from cognition. So, I made a little Chrome extension for myself where I like any repo, including yours, uh, I can just hit e wiki and this is an llm.txt kind of, but also I I read it. Um, it's just a better wiki. No, no. So, this this is a great example. So, and I actually I think that this could be a very nice approach. Take a repo, compile it down to some kind of easily kind of readable Yeah. lm.ext.

What I actually found was even using an LM to write the descriptions helped a lot. So I have actually a little package on my GitHub where it can rip through documentation and just pass it to a cheap LLM to write a high quality summary of each doc. This works extremely well. And so that lm.ext then has LLM generated. Yeah. Let's see where it is. It's uh >> you go to my repos. It's one of my newer ones. No, that's old. That's much older.

I have a million things here. Go to my repos. Uh it's go back up to the top. It's I've too much here. Try repositories. >> You do too much, man. >> Yeah, too much open. >> It won't be top. It'll be uh Yeah, this one this is a little repo. It got almost no attention and but I found it to be very useful. So, basically, you can it's it's trivial. You just point it to some documentation. It can kind of rip through it, grab all the pages, send each one to an LLM, and LM writes a nice description, compiles it into an LM.ext file. I found when I did this and I then fed that to Claude Code, Cloud Code's extremely good at saying, "Okay, based on the description, here's the page I should load. Here's the page I should load for the question asked."

Dead simple. I use this all the time. Um, well, I use it when I'm trying to generate element.ext for new documentation, but I've done this for Langraph. I've done it for a few other kind of libraries that I use frequently. You just give that to Cloud Code. Then cloud code can rip through and grab docs really effectively. super simple. And the only catch is I found that the descriptions in your element of text matter a lot because LM actually has to use the descriptions to know what to read, you know. Anyway, that's just uh a nice little utility that I use all the time.

>> When we had a client that said the context 7 MCP by Upstach, which is like an MCP for like um project documentation and stuff like that, was one of the most used. Have you seen Have you tried it? Have you seen anything else like that that kind of like automates some of the stuff away? >> Well, you know, it's funny. We have an MCP server for Minecraft documentation that basically gives, for example, CLA code the an LM.ext file and a simple search file search tool. Now, Claude has built-in kind of uh fetch tools, but at the time we built it, it didn't. But it's a very simple MCP server that exposes LM.ext files to, for example, cloud code. It's called MCP doc. Uh, so it's a little very simple utility. I use that all the time. Extremely useful. So you basically can just point it to all the lm.ext files you want to work with.

Yeah, this one. >> Yeah. Well, the MCP docs have a MCP server that you can search the docs with. So it's kind of rolls all the way down. But I guess my question is like should this be like one server per project you know or like at some point you're going to have kind of like a meta server which is like the and I think part of it is um you know once you move on from just doing tool calling in servers to doing things like sampling and kind of like uh you know prompts and resources and stuff like that you can do a lot of the extraction in the server itself as well. And again it goes back to like your point on context engineering. It's like maybe you do all that work not in the context but in the server and then you just put the final piece that you care about in the context. Uh but it seems like very early.

>> Yeah, this is actually a very interesting point. I've spoken with folks from Enthropic about this quite a bit. It is I found that storing prompts in MCP servers is actually pretty important in particular to tell the LM or code agent how to use the server. And so I actually end up do having kind of separate servers for different projects with specific prompts. Um and also sometimes I'll have you can also sort of have resources. So some have specific resources for that particular project in the server itself. Um so I actually don't mind separating servers project-wise um with project specific kind of context and prompts necessary for that particular task. Yeah, a lot of people actually may have missed some features of the NCP spec and uh you do have prompts in there. Uh it's probably one of the the first actual uh features that they that they have which uh actually maybe kind of underrated like people kind of view MCP as just you know uh in tool integration but uh there's actually a lot of stuff in here including uh sampling which um is is underrated too.

>> Yeah, that's right. That's exactly right. And actually that the prompting thing is pretty important because even to use our little simple MCP doc server for Langraph docs you actually I found it's better of course if you prompt it but then I had to put in the readme initially like oh okay here's how you should prompt it but of course that prompt can just live in the server itself and uh so you can kind of compartmentalize the prompt necessary for the LM to use the server effectively within the server itself. And this was a problem I saw initially. A lot of people were using our MTP doc server and then finding, oh, this doesn't work well. And it's like, oh, it's a skill issue. You need to prompt it better. But then that's that's our problem. This the prompt should actually live in the server and should be available to the code agent, >> right?

>> Uh so it knows how to use the server, >> right? >> So that that's maybe retrieval and that's a whole retrieval is a big theme. Uh and it's you know it it obviously predates this new term of context engineering, but there's a lot going on in the retrieval bucket. certainly is an important subset of context engineering. >> I'm wondering if there's any other trends in retrieval. Before we leave the topic, um you know, I think one other thing I was tracking was just coar and like the the general concept of late interaction. I don't know if you guys do a do a ton on that, but um some sort of in between element between full agentic and full pre-indexing and sort of um two-phase indexing maybe is is what I would call it. Any comments on that?

>> I haven't personally looked at Cold Bear very much. I played with it only a little bit, so I don't have much perspective there, unfortunately. >> All right, happy to move on. >> We could talk about maybe reducing context briefly. Everyone's had an experience with this because if you use cloud code, you hit that kind of 95, you know, you've hit 95% of the context window and you're about to and cloud code's about to perform

compaction. So that's like a very intuitive and and kind of obvious case in which you want to do some kind of context reduction when you're near the context window.

I think an interesting take here though is there's a lot of other opportunities for using summarization. We talked about it a little bit previously um with offloading but actually at tool call boundaries is a pretty reasonable place to do some kind of compaction or pruning. I use that in open deep research hugging face actually has a very interesting open deep research implementation. It actually uses like not a coding agent but the code agent agent implementation. So instead of tool calls as JSON, tool calls are actually code blocks. They go to a coding environment that actually runs the code.

And one argument they make there is that they perform some kind of summarization or compaction and only send back limited context to the LLM, leave the raw kind of tool call itself, which is often token heavy as we're talking about deep research in the environment. So it's another example anthropic and their multi- aent researcher also does kind of summarization of findings. Um so I think you see pruning show up all over the place. It's pretty intuitive. I think an interesting counter to pruning was made by Manis. They make the point and the kind of warning that pruning comes with risk particularly if it's irreversible.

And cognition kind of hits this too. They talk about we have to be very careful with summarization. You can even fine-tune models to do it effectively. That's actually why Manis kind of has the the perspective that you should definitely use context offloading. So perform tool calls offload the observations to for example disk so you have them. then sure do some kind of pruning summarization like Allesia was asking before to pass back to the LM useful information but you still have that raw context available to you so you don't have kind of lossy uh compression or lossy summarization so I think that's a an important and useful caveat to note on the point of summarization or pruning you have to be careful about information loss uh this is something that people do disagree on and I'll just flag this uh on pruning mistakes pruning wrong paths.

Um, Mana says keep it in and so you can learn from the mistakes. >> Some other people would say that well once you've made a mistake it was going to keep going down that path that there was a mistake you got to you got to unwind or you just got to like prune it and tell it do not do the thing I know to be wrong. So So then you just do the other thing. I don't know if you have an opinion but like I would call this out there. There was someone that that spoke yesterday that disagreed with this.

>> So that's actually very interesting. So Drew Brunig has a really nice blog post on context failure modes. >> So he has a few. I'm just gonna >> Yes. This one context poisoning. This is interesting. Drew Brunick has a nice blog post that hits this point. He talks about this theme of context poisoning and apparently Gemini reports on this in their technical report. So he talked about um for example a model can perform a hallucination and that hallucination then is stuck in the history of the agent and it can kind of poison the context so to speak and and kind of steer the agent off track and I think he he cited a very specific example from Gemini 25 playing Pokemon they mentioned in the tech report. So that's one perspective on this issue of we should be very careful about mistakes in context that can poison the context.

That's perspective one. Perspective two is like you were saying is if an agent makes a mistake for example calling a tool you should leave that in so it knows how to correct. So I think there is an interesting tension there. I will note it does seem that clawed code will leave failures in. I notice when I work with it for example it'll it'll kind of have an error the error will get printed and it'll it'll kind of use that to correct. So and in my experiences when working with agents in particular for tool call errors I actually like to keep them in personally. That's just been my experience. uh I don't try to prune them. Also, for what it's worth, it can be kind of tricky to prune from the contact from the from the message history. Um you have to decide when to do it. So, if you're introducing a bunch more code you have to manage. Um so, I'm not sure I love the idea of kind of selectively trying to prune your message history when you're building an agent.

It can add more logic you need to manage uh within your kind of agent scaffolding or harness. That's a classic sort of precision recall, but like sort of reinvented for uh context in a in an agentic workflow. >> Exactly. Exactly. Right. While we're on the topic of Drew, Drew, Drew is obviously another really good author. Uh he's created he's coined a bunch of like sort of context engineering lore. Um any other uh commentary on on stuff that you know you particularly like or disagree with?

>> I'll show you something kind of funny. if you go to his post. Um, so he and I did did a meet up on this and I I kind of like this quote from Stuart Brand. It was kind of comical. If you want to know where the future is being made, look for where language is being invented and lawyers are congregating. So, and it was talking about this this idea of kind of why buzzwords emerge. And he actually was the one who turned me on to this idea that a term like conduct engineering kind of catches fire because it captures an experience that many people are having. They don't come out of nowhere. And if you scroll down a little bit, he kind of talks about this.

He's a whole post about kind of I think it's how to build a buzzword. Um but he talks a lot about this idea of of kind of successful bud buzzwords are capturing a common experience that many of us feel and I think that's kind of the the genesis of context engineering is also largely because many of us build agents. Ooh, there's lots of ways that can be quite tricky and oh, contact engineering is kind of what I've been doing and you hear a number of people saying and then you it kind of resonates. You say, oh, okay, yes, that describes my experience. So, I think that's just an interesting aside on on kind of how language emerges anthropologically kind of in in in different communities.

>> Uh, well, I mean, I I will cosign this because that's exactly what I use to coin or come up with AI engineer. >> A engineer. No, exactly. just because uh people were trying to hire software engineers that were uh more up to speed with the AI and engineers wanted to work at companies that um would respect their work, you know, and uh maybe also come out from the baggage of classical ML engineering. Uh a lot of AI engineers don't even need to use PyTorch because you can just prompt and do typical software engineering. Um and I think that's probably the right way. uh at least in a world where most models are most of the frontier models are coming from closed labs.

>> I think an interesting counter on this is uh when you for example people try to create language that doesn't really resonate that doesn't capture common experience it tends to flop. So which is to say that buzzwords kind

of co-evolve with the ecosystem. They tend to kind of become big and and resonate because they actually capture experience. Many people try to coin terms that don't actually resonate that go nowhere. >> Allesia, do you have experience with that?

I'm the worst at naming things. Uh, but you do a great job, Sean. >> Yes, >> you nailed it. The the few ones you put on lat space. So, >> that's right. Cool. Uh, well, you know, I I wanted to talk about context engineering. Uh, okay. So, so, uh, sorry, I I don't know if I sidetracked you a little bit with >> No, that's perfect. >> the meta stuff on on >> that hit that hits a lot of the major themes. I can maybe just talk very brief about one more. We could talk about bitter lesson and some other things.

>> Yeah. Um, if you go back to that table, I just wanted to give Manis a shout because I thought they had one other very interesting point. >> Oh, the the table that you had. >> Yes, exactly. So, we've talked about offloading, reducing context, retrieval, context isolation. Those are I think the big ones you can see very commonly used. I do want to highlight Manis. So, I thought they had a very interesting take here about caching. And it's a good argument. When people have the experience of building an agent, the fact that it runs in a loop and that all those prior tool calls are passed back through every time is quite like a shock the first time you lo an agent. You have one token every tool call and you incur that token cost every pass through your agent. And so, Manis talks about the idea of just caching your prior message history. It's a good idea. I haven't done it personally, but seems quite reasonable. So, caching reduces both latency and cost significantly.

>> Yeah. But don't most other APIs auto cache for you? I mean, if you're using like OpenAI, you would just automatically have a cash hit. >> I'm actually not sure that's the case. For example, when you're building a you're passing your message history back through every time. As far as I know, it's stateless. >> I think um so there's different APIs for this across the different providers. Um, but especially if you use just the responses API, the new one. Um, it should be that if you're just if you're never modifying the state, uh, which is good for people, good for you if you believe that you shouldn't compress conversation history. Bad for you if you do. Uh, if you never modify the state, then you can just use the SS API.

Everything that you passed in prior is going to be cached, which is which is kind of nice. Enthropic used to require weird header thing and they've made it more automatic. >> Yeah. Okay. Okay, so that's a good call out. So I had used Anthropics kind of caching header explicitly in the past, but it may be the case that caching is automatically done for you, which is which is fantastic if that's the case. I think it's a good call out for Manis.

>> Yeah, Gemini also introduced implicit caching. Yeah, it's like it's it's really hard to keep up. Like you basically have to follow everyone on Twitter and just like read everything. Um, so that's my bullet bot for it. >> Yeah. Yeah. Yeah. Yeah. Yeah. Well, you know, you know, it's it's interesting though. So APIs are now supporting caching more more. That's fantastic. Um, I had used Anthropics explicit cacher caching header in the past. I do think an important and subtle point here is that caching doesn't solve the long context problem. So, it of course solves the problem of like latency and cost.

But if you still have 100,000 tokens in context um whether it's cached or not, the LM is utilizing that that

context. this came up. I actually asked Anton this in their context rot meetup um or in their context rot webinar um and and they kind of had mentioned that the characterization of context rot that they made they they think they would expect to apply whether or not you're using caching. So caching shouldn't actually help you with all the context rot and long context problems. It absolutely helps you with latency and cost.

>> Yeah, I I I do think I do wonder what else can be cached. Um I feel like there this is definitely a form of lock in um because you ideally want to be able to run prompts across multiple providers and uh and all that and uh yeah caching is a is a hard problem like I think ultimately like you control your destiny if you can run your own open models because then you also control the caching uh here every everything else is just a half approximation of that >> that's right that's exactly right >> that overall broad uh context engineering. Allesia, I don't know if you have any other takes from like the meetup yesterday or questions.

>> No, I think my main take from yesterday was like um quality of compacting. Um I think there was like one of the charts was using the automated compacting of like a open code and some of these tools is basically the same as not doing it on like the quality of what you get from the previous instructions. And um I think Jeff at this charge is like curated compacting is like 2x better but I'm like how to you know it's like how do you do curated compacting? I think that's something that uh maybe we can do a future blog post on. I I think that's interesting to me like how do you compact especially coding agents things where like it can get very very long. I think for things like deep research is like look once I get the report it's fine you know but for coding it's like well I would like to keep building. I I found that like even when you're like writing tests or like you're doing changes having the previous history it's like helpful to the model it seems to perform better when it knows why it made certain decisions and I think how to extract that in a way that is like more token efficient and still unclear. Um so I don't have I don't have an answer but maybe like a request for for work by people listening.

>> Yeah, you know that that's a great point. It actually echoes some of Wald and Dan's points from cognition also that the summarization compaction step is just is non-trivial. You have to be very careful with it. Devon uses a fine-tuned model for doing summarization within the within the context of coding. Um so they obviously spend a lot of time and effort on that particular step. Uh and Manis kind of calls out that uh they are very careful about uh information loss whenever they do pruning, compaction, summarization. they always use a file system to offload things so they can retrieve it. So it's a good call out that compaction is risky when you're building agents and very tricky.

You know, I think there were a lot of there's a lot of previously a lot of interest in memory and um I'm always I was thinking about the interest interplay between memory and uh context engineering. I mean are they kind of the same thing? Is it just a rebrand? uh are there parts of memory and you know you guys recently uh relaunched Langm that's also a form of context engineering but I I don't know if there's there's like a qualitatively or philosophical difference >> yeah so that's a good thing to hit actually I maybe think about this on two dimensions writing memories reading memories and then the degree of automation on both of those so take the simplest case which actually I quite like claude code how do they do it well for reading ing memories, they just suck in your cloud MDs every time. So every time you spin up cloud uh cloud code, it pulls in all your cloud MDs for writing memories, the user specifies, hey, I want to save this to memory and then cloud code writes it

to cloud MD. So on this axis of like degree of automation across read, write, it's kind of like the 00. It's very simple and it's kind of very like Boris Pilled like super simple and I I actually quite like it. Now the other extreme is maybe chatbt. So behind the scenes chatb decides when to write memories and it decides when to suck them in. And actually I thought Simon at a engineer had a great talk on this and he it wasn't about memory but he hit memory in the talk and he mentioned I don't know if you remember this but it was a failure mode in image generation because he wanted an image of a particular scene and it sucked in his location and put it in the image like it sucked in half like half moon bay or something and suck it in the image and it was a case of memory retrieval gone wrong he didn't actually want that so even in a product like chap chatbt that spent a lot of time on memory memory uh it's non-trivial and I think my take is the well the writing of memories is tricky like when actually should the system write memories is is non-trivial reading of memories actually kind of converges with the contextary theme of retrieval like memory retrieval at large scale is just retrieval right I I kind of view them as it's retrieval in a certain context which is your past conversations which uh >> that's right >> you know it is different than retrieval from a knowledge base different than retrieval on the public web. Uh by the way, this is uh S's write up on his website uh on on here where he was just trying to generate images and then suddenly it shows up that that's exactly it. So that there you go. Uh actually it's a subtle point.

I I don't know exactly know what open I does behind the hood with respect to memory retrieval. My guess is they're indexing your past conversations and using semantic vector search and probably other things. So it may still be using you know some kind of knowledge base uh or or vector store for retrieval. >> So in that sense I kind of view it just simply as um >> you know in the case of sophisticated memory retrieval it is just like a a you know complex rag system in the same way we talked about with like Verun and building wind surf it's kind of a multi-step rag pipeline. Um so I I kind of view memories at least the reading part as just you know it's just retrievable. Um and actually I quite like clause approach is very simple just the retrieval is trivial just suck it in every time.

>> Uh totally. Um I would also highlight um the sort of uh the semantic differences that you've you've established you know episodic semantic procedural uh and background memory processing. We've done an episode with the letter folks on sleeptime compute uh which you know I think these are just like if you if you have ambient agents very longunning agents you're going to run into this kind of context engineering which is previously the domain of memory and uh I would say that the classic context engineering discussion doesn't have this stuff not yet >> yeah so actually there there's an interesting point there uh I did a course on building ambient agents and I built I have this little email assistant that I use to run my email.

I actually think this is bit of a sidebar in memory. Memory pairs really well with human in the loop. So, for example, in my little email assistant, it's just an agent that runs my email. I have the opportunity to pause it before it sends off an email and correct it if I want, like change the tone of this email or I can literally just modify the tool call to have a little UI for that. And every time when you have these ambient agents, you edit for example or you give it feedback, you edit the tool calls itself, that feedback can be sucked into memory. And that's exactly what I do. So I actually think memory pairs very nicely with human loop. And like when you're using human loop to make corrections to a system, that should be captured in memory. And so that's a very nice way to use memory in kind of a narrow way that's just capturing user preferences uh in a over time. And actually use an LLM to actually reflect on the changes I made, reflect on the prior instructions in memory and just

update the instructions based upon uh my edits. And that's a very simple and effective way to use memory when you're building ambient agents that I quite like.

>> Uh there's there is a course which you can find on the GitHub. Um and yeah, I mean you know you guys have done plenty of talks on agents. >> That's right. But I I think it I think it's a very good point that memory is often kind of confusing when to use it. I think a very clear place to use it is when you're building agents that have human loop because human loop is a great place to update your agent memory with your preferences. So it kind of gets smart over time and learn streaming.

It's exactly what I do with my little email assistant. So Harrison, I'm sure I think he said this publicly, uses an email assistant for all his emails. Uh and he gets a lot as a CEO. I I get much fewer because I'm just a lowly guy. Uh but I still use it. Um, and uh, that's a very nice play way to use memory is kind of pair it with human in the loop. >> Yeah, totally. Um, I've I've tried to use the the email system before, but like uh, you know, I'm still still very married to my superhuman.

>> Yeah, fair enough. That's right. >> That's right. >> Okay, so cool. Um, I think that, you know, that was that was about the coverage that we planned on context. >> That's great. >> You have a little bit on a bit of lesson that we could uh, wrap up with. >> Yeah, that's a fun theme to hit on a little bit. I'd love to hear your perspective. So there's a great talk from Hyong Wong Chung Y >> previously open AI now have MSL >> Y >> on the bitter lesson in his approach to AI research.

So the take is compute 10xes every 5 years for the same cost. Of course we all know that the kind of history of machine learning has shown yeah exactly this slide exactly history machine learning has shown that actually capturing this scaling is the most important thing in particular algorithms that are more general with fewer inductive biases and more data and compute tend to beat algorithms with more for example handtuned features inductive biases built in which is to say just letting a machine learn how to think itself with more compute and data rather than trying to teach a machine how we think tends to be better. So that's kind of the bitter lesson piece simply stated. So his argument is this subtle point that at any point in time when you're for example doing research you typically need to add some amount of structure to get the performance you want at a given level of compute. But over time that structure can bottleneck your further progress. And that's kind of what he's showing here is that in the kind of low compute regime kind of on the left of that x-axis adding more structure for example more modeling assumptions more inductive biases is better than less.

But as compute grows less structure and this is exactly the bitter lesson point less structure more general tends to win out. So his argument was we should add structure at a given point in time in order to get something to work with the level compute that we have today but remember to move it later. And a lot of his argument was like people often forget to remove that structure later. And I think my link here is that I think this applies to AI engineering too. And if you kind of scroll down I have the same chart showing my little exactly this is this is my little example of building deep research over the course of a year. So I started with a highly structured research workflow. Didn't use tool calling. I embedded a bunch of assumptions about how research should be conducted. In particular, don't use tool calling because everyone knows tool calling is not reliable. This was back in 2024.

Decompose the problem into a set of sections and parallelize each one those sections written in parallel into the final report. What I found is you're building LM applications on top of models that are improving exponentially. So while the workflow was more reliable than building an agent back in 2024, that flipped pretty quickly as LMS got better and better and better. And so it's exactly like was mentioned in the Stanford talk. You have to be constantly reassessing your assumptions when you're building AI applications given the capabilities of the models. And I talk a lot about here the structure, the specific structure I added, the fact that I used the workflow because we know tool calling doesn't work. This was back in 2024. the fact that I decomposed the problem because it's how I thought I should perform research and this basically bottlenecked me. I couldn't use MCP as MCP got for example, you know, much more popular. I couldn't take advantage of the fact that tool calling was getting significantly better over time. So then I moved to an agent, started to remove structure, allow for tool calling, let the agent decide the research path. A subtle mistake that I made which links back to that point about failing to remove structure. I actually wrote the report sections within each sub aent. So this talks this kind of links back to what we talked about with sub aents in isolation. Sub agents don't communicate effectively with one another. So if you write report sections in each sub aent the final report is actually pretty disjoint. This is exactly Allesio's challenge and problem about you know using multi- aent. So I actually hit that exact problem. So I ripped out the independent writing and did a oneshot writing at the end and this is the current kind of version of open deep research which is quite good and this is kind of the thing that's at least on deep research it's the best performing open deep research assistant at least that that's open source so it was kind of my own arc although we do have some results with GPD5 that that are quite strong so you know the models are always getting better and so indeed our open source assistant actually takes advantage and rides that wave um but I actually kind of experienced I felt like I I actually got bitter lessons myself because I started with a system that was very reliable for the current state of models back in mid 2024, early 2024, but I was completely bottlenecked as models got better. I had to rip out the entire system and rebuild it twice, rechecking my assumptions um in order to kind of capture the gains of the model. So, I think I just want to flag I think this is an interesting point. It's hard to build on top of rapidly expanding models, rapidly improving model capability. And actually, I really enjoyed from a engineer Boris's talk on cloud code. And they're very bitter lesson build. He talks a lot about the fact that they make cloud code kind of very simple and very general because of this fact. They want to give users unfettered kind of access to the model without much scaffolding around it.

>> Yeah, exactly. He he hits it in one of these slides. Yeah. I I don't know where. >> Yeah. Yeah. Yeah. But but I but I think it's it's an interesting consideration in a engineering that we're building on top of models that are improving exponentially. And one of the points he makes is a core layer of the bitter lesson is that more general things around the model tend to win. And so when building applications we should be thinking about this. We should be adding structure necessary to get things to work today, but keeping an eye on improving models and keep but keeping a close eye on models improving rapidly and removing structure in order to unbottleneck ourselves. I think that was kind of my takeaway. So, I really liked the talk um from Hyong Wan Chung. I think that's worth everyone listening to. Um and I think a lot of lessons apply to engineering. I think this is similar to incumbents kind of like adopting AI putting in existing tools because you already have the workflow right so you already have all the structure you just put AI it becomes better um but then kind of like the AI native approaches catch up as the models get better and then there's like there's no way for existing products to remove the structure because the structure is the product you know >> and that's why then you have you know cursor and windsurf are being are better than vs code for like yeah native thing just because they didn't have to deal

with removing things and why cognition is like you know uh again it's like it doesn't even think about the ID as like the first thing the ID is like a piece of the agent um and so I think you see this in a lot of markets which is like hey >> again if you have a workflow and you put AI the workflow is better >> like the workflow is not the end goal you know um and so I think we're now at a place where like you should just start without a lot of structure just because now the models are like so good but I think the first >> two 2 and a half years of the market there was kind of like the stance of like should I just put AI into the workflow that works? Should I rewrite the workflow but the workflow is not that good because the models are not that good. Uh but I think we're past we're past that point now.

>> That's an amazing example actually. If you show your chart again there's another interesting point in your chart. An interesting point here is that in the kind of earlier model regime the structure approach is actually better. And so an interesting take on this. So um Jared Kaplan the founder of Anthropic has a great talk at startup school from a couple weeks ago and he mentions this point about oftentimes building products that explicitly don't quite work yet can be a good approach because a model under them is exponentially and it'll kind of unlock the product. We saw that with cursor. So like part of the cursor lore is that it it did not work quite it did not work particularly well. Cloud35 hits and then boom it kind of unlocks the product. And so you kind of hit that knee near the curve when the model capability kind of catches up to the product needs. But in that in that earlier regime, the structure approach appears better. So it's kind of this interesting subtle point that for a while the more structure approach appears better and then the model finally hits the capability needed to unlock your product and suddenly your product just takes off. There's kind of another correlary to this that you can get tricked into thinking your structured approach is is indeed better because it'll be better for a while until the model catches up with less structured approaches. Your chart looks very similar to the winds surf chart. Uh I got to bring it up because uh I I was involved in this in the writing of this one. Isn't that similar? There's a there's a there's the ceiling, you know, and then like the boom you you go slow. It's this computer lesson but in like uh Enterprise S.

>> That's right. That's right. Very similar. Very similar. I I I just like you know for me like okay the lines are important but to me the bullet points are the main thing. If you understand the bullet points then you cannot you can actually learn from the the the mistakes of others. >> Uh I spend a lot of effort on the bullet points. >> Right. Cool. Yeah. I mean uh so generally you know um there is one spicy take on on this which is like you know how much is langraph aligned with the bitter lesson. Yes, obviously you guys are obviously aware of it, so it's not going to be a surprise. But I do think that making things making abstractions easy to unwind is very important if you believe in a bitter lesson, which which you do.

>> No, no, this is super important actually and I actually talked about this in the post. >> Yeah, there's an interesting subtlety when you talk about Asian frameworks and a lot of people are anti-framework. I completely understand and sympathetic to those points. But I think when people talk about frameworks, there's two different things. So there can be a low-level orchestration framework. There's a great talk for example at um Anthropic from Shopify. They built this kind of orchestration framework called roast internally and uh it's basically langraph.

It's some kind of way to build kind of internal orchestration workflows with LMS and roast langraph provides you low-level building blocks nodes edges state which you can compose into agents you can compose into workflows I don't hate that I like working low building blocks they're pretty easy to tear down rebuild in fact I I used for example langraph to build open research I had a workflow I rip it out I re agent the building blocks are low-level just nodes, edges, state. But the thing I'm sympathetic to is there's also in addition to just kind of low-level orchestration frameworks, there's also agent abstractions like from framework import agent.

That is actually where you can get into more trouble because you might not know what's behind that abstraction. I think when a lot of people kind of are anti- framework, I think what they're really saying is they're also anti-abstract. They're they're largely anti-abstraction which I'm actually very sympathetic to and I I don't particularly like agent abstractions for this exact reason and I think Walden Yans made a good point like we're very early in ARC of agents we're like in the HTML era uh and and agent abstractions are problematic because you don't know what's necessar under the hood of the abstraction you don't understand it and if I was building for example you know open research with an with an abstraction I wouldn't necessarily know how to rip it apart and rebuild it uh when models got better so I'm actually wary of abstractions. I'm very sympathetic to that part of the critique of frameworks, but I don't hate low-level orchestration frameworks that just provide nodes, edges, you can just recombine them in any way you want. And then the question is why use orchestration at all? And actually, I use Langraph because you get some nice you get checkpointing, you get state management. It's low-level stuff. And that's the way I happen to use Langraph and that's why I like Langraph and that's actually why a lot of I found like a lot of customers like Langraph.

It's not necessarily for the agent abstraction, which I agree can be much trickier. Some people like agent abstractions. That's completely fine as long as you understand what's under the hood. But I think that's a very interesting debate about frameworks. I think the critique is it should be made a little bit more on abstractions because often people don't know what's under the hood. >> For those who are looking for resources, uh it was a bit hard to find the Shopify talk because it's unlisted.

>> Yeah, it's unlisted now. Exactly. I don't know why it's unlisted, but it's a nice talk. I found it through the this Chinese Chinese ripoff of the talk. >> Funny. There you go. >> Yeah, it's hard. It's actually hard to find now. >> I think there should be a browse comp where uh you find obscure YouTube videos because that's something I'm very good at. Just kind of my bread and butter. >> It's good. And what you know what's funny is this talk follows exactly the arc we often see when we're talking to companies about Lang graph. It is people want to build agents and workflows internally. Everyone rolls their own. it becomes hard to kind of manage and coordinate and review code in large in this context of large organizations. It can be very helpful to have a standard library or framework that people are using with low-level components that are easily composable. That's that's what they build with roast. That's effectively what Lang graph is and that's why a lot of people like Langraph. I actually thought the talk on MCP that uh I believe it was it was u at a engineer it was um Jean Welsh.

>> Yes. I I I think that was like a super underrated talk. I tried yelling about it. No one listened to me. But like, you know, if you listen this far into the podcast, do us a favor. Did actually listen to uh John Welsh's talk. It's

actually very good. >> It's very good. So, so actually he makes a case for a lot of the reason why people actually, for example, enterprises, large companies like Langraph, which is the fact that when tool calling got good within anthropic in, you know, sometime mid last year, he actually makes this point explicitly.

Yes, exactly. It's it's somewhere right around here. Actually, there's a timeline slide if you go back like one or two. It's very good. So, this this is very interesting. So, he mentions, okay, so your anthropic tool calling gets good in mid124. Everyone's building their own integrations. It becomes complete chaos and that's actually where kind of MCP came from. Let's build a kind of a standard protocol for accessing tools. Everyone adopts it. Much easier to kind of have a and have review and you minimize cognitive load. And this is actually the argument for standardized tooling whether it be frameworks or otherwise within larger orgs is practicality. And he actually his whole talk is making that very pragmatic point which is actually why um people do tend to like kind of frameworks uh for example in larger in large organizations. Agreed. Um and then and then ship it as a gateway. Uh this is the the other uh big um thing that they do. Um >> that's right Lance. you've been so generous of your time. Thank you. Uh any shameless plugs, uh call to action, stuff like that.

>> Yeah, if you made it this far, thanks for listening. Um we have a a bunch of different courses I've taught uh one on ambient agents, uh one on building open research. So I I actually was very inspired by uh Carpathy had a tweet a long time ago talking about building on-ramps. So he talked about he had his microrad repo. A few people looked at it, but not that many. He made a YouTube video and that created an on-ramp and the repo skyro skyrocket in popularity.

So I like this onetwo punch of building a thing like open research then creating a class so people can actually understand how to build it themselves and I kind of like that build a thing create an on-ramp for it. So I have a class on building open research feel free to it's for free um but it it walks through a bunch of notebooks as to how I build it and and you can see the agent is quite good. We even have better results coming out soon with GPD5. So, uh, if you want kind of an open source deep research agent, have a look at it.

Um, it's, uh, it's been pretty fun to build and that's exactly what I talk about in that bitter lesson blog post as well. >> Awesome, Lance. Thank you for joining. >> Yeah, a lot of fun. Great to be on. [Music]