Lamis Mukta - Learning while you sleep: Beyond memory to dreaming - AI Native DevCon June 2026

31 MIN · YOUTUBE · HTTPS://WWW.YOUTUBE.COM/WATCH?V=TTCXVV8HHNW
https://www.youtube.com/watch?v=tTcxVv8HHNw

SUMMARY

Lamish from Anthropic discusses the importance of context engineering in AI, particularly for enhancing the performance of agents through effective memory management. He outlines the evolution of context engineering over the past year, emphasizing the need for agents to autonomously manage their memory while implementing guardrails for production environments.

- *Context engineering is crucial for translating model intelligence into practical applications.*
- *Key developments include the use of markdown files for context, memory tools allowing agents to manage their own memory, and skills for procedural workflows.*
- *Challenges such as context bloat, stale memories, and concurrency issues arise when scaling memory systems for multiple agents.*
- *Best practices for memory management include versioning, concurrency controls, permissioning, and portability of memory systems.*
- *The concept of "dreaming" is introduced as an out-of-band process for reviewing and updating memory stores based on agent interactions.*
- *Effective memory systems lead to improved accuracy, speed, and cost efficiency in agent performance.*
- *The journey of context engineering is ongoing, with significant potential for future advancements in AI applications.*

Welcome back.
And we have another wonderful session
with a topic that is very deep to my
heart, sleep. And
I'd love to welcome Lamish from
Anthropic to the stage. Please, a warm
welcome
for our next talk.
>> [applause]
>> Hey everyone.
Great to see you all today and hope
you're all having a wonderful day so far
here at AI DevCon. I certainly am.

I hope that your contacts windows are not too full and you have a bit of space for a little bit more information about context.

By way of introduction, my name is Lamish. I'm a member of technical staff at Anthropic.

I work on our applied AI team and this is a team which sits between research, product and go-to-market. So we do a mixture of working on internal projects as well as directly with customers who are building agents at the frontier.

Me specifically, I work with startups and founders, many of whom I'm sure are in this room today.

And I think I have the best seat in the house because

these are the users that are constantly pushing us right up against the boundary of what is possible with models and products today.

And as such, we just get to like ride the exponential together.

One thing that constantly comes up, as I'm sure you're all aware, is what it really takes to take the raw model intelligence that we have today and translate that into durable, scalable, useful products.

And one of the main levers that we have in order to do this is context engineering, which will be the focus of my talk today.

So on this journey, I want to quickly do a recap of where context engineering has gone in the past year. It's a space that's completely blown up.

And through that, we'll kind of distill the primitives that have proven to be really useful, some stuff that has been a little bit less useful.

Secondly, we'll talk about what the state of the art is for memory management today.
And thirdly, and and in particular with that, we'll talk about not just what nice theoretical principles are, but what it takes to actually build these systems in production.
And then finally, we'll talk about where this will go on the path to continual learning, and in particular touching on a paradigm called dreaming.
So,
we said this before, and models we release new models all the time, they are more and more intelligent.
But when it comes to actually deploying these models in your agents, in your environments, in your organization, the intelligence alone is not going to compound because they need this context that helps them perform the specific tasks that you need them to.
In particular, a lot of this context is often kind of orthogonal to the model intelligence, right? Like,
the newest model, we just released one, um isn't going to neces- isn't going to out of the box know exactly what it takes to succeed in your organization and with the tasks that you want them to.
And so, it's a really great investment to work on the context engineering part because this
uh
over time has the effect of multiplying the intelligence even as models get smarter.
So, I'm sure you'll all be familiar with these problems. It's like agents not

knowing their way around a code base or knowing enough about your own user preferences.
Um
And then additionally, like you don't have the effect where agents are better at the task the next time they perform it. So, they they might not learn from their mistakes, um and as such, you don't have this uh continual learning effect. So,
just to recap
uh where we've got so far on this journey of context engineering.
At Anthropic, we like to say do the simple thing that works.
And this is a timeline that's only really spans the past year, and where we started
was with these Claude MD files that we launched with Claude code.
And what we learned from this was that it was kind of unreasonably effective.
Like this markdown file that just gives the agent a couple of instructions about maybe your way around the code base, the organization, um your own user preferences.
That injected into the beginning of the model context at the beginning of a session was so good at steering it towards the things that mattered and helping it navigate
and align its actions towards your preferences.
However, we also learned a couple of things about what doesn't work here. So, when we're injecting this at the beginning at the beginning of the session into context, we obviously start to run into problems where you get problems with context bloat, like what

happens when this file with very important preferences gets very, very long. Um how do I manage that over time? And so, we went back to the drawing board and thought about like ways that we could improve this.
Separately though, what was true was that having just a very simple markdown file, which is human readable, which your agent can write to, which you can write to, is really, really effective.
So, a second avenue that we investigated was these memory tools.
And this is interesting because it leans into this idea of, okay, what happens if we let agents autonomously manage their own memory systems. So, we let them decide when they read, when they write, um and when they update memories.
And this is all happening in band, which means that it's within the context of a session. So, during a session, a agent is thinking about like what might be interesting to pull from memory, what might be interesting to write to memory.
So,
autonomy proved to work really well in this case.
Um
and over time, we've kind of developed that into systems where we're even less opinionated about what these tools need to look like and I'll touch on that in a second.
So, the next uh
the next stop on this journey was skills, which I'm sure you've heard a bunch about today.
And what this solves is
this problem of the like ever-growing context. So, we have this really clever idea of um progressive disclosure.

The way I like to think about it is um actually first on skill what skills are good at, it's processes where you have like a procedural workflow. So, something where you have an opinion about how the process works end to end that you want the agent to follow.
Um and what's very clever is that the agent only looks at this front matter a couple of sentences at the top of the file before loading the skill.
But you can still load as much detail as you want into the main body of the file.
So, you're able to at the same time have very deep levels of detail
whilst not overloading the model's context.
And the way I like to think about it is as if I'd had a bookshelf in my room and every time someone talks to me um I can kind of scan and look at my list of books and see if any of the titles might be relevant to the conversation and kind of pick that off the shelf and read it when I need to. So, for example, if someone walked up to me and started speaking to me in French and I noticed that I have a French dictionary, I could pull that out and it would give me context kind of loaded during the conversation that would help me
um without me having paid attention in like seven years of French classes at school and having that all loaded into my context already.
So, this was a really really great innovation.
But one bottleneck potentially with it is that it's still kind of driven by humans and agents together. So, you're still even if you're using your agents to write the skills with you, you're

still being quite opinionated about like
what things need skills.
So, that takes me to the final step on
this path, which is what we perceive to
be state of art for memory systems
today.
>> [snorts]
>> And what we have done and yeah, what we
think is best practice,
is modeling these memory systems just as
file systems.
So, this kind of aggregates a couple of
the learnings from this path. So, file
systems are great. You can just fill
them up with markdown. Um agents are
actually just very good at using normal
file system tools like bash and grep.
So, just let them search over the file
system, rather than being opinionated
about the specific tools that they
should use to read and write to memory.
Um and then yeah, that that search kind
of mirrors this idea of progressive
disclosure. You can index these memory
systems really well, so that agents can
intelligently search over them. And
that's where we have kind of got to so
far.
So, just a
recap the key learnings from that
format. Markdown is great um for reading
memories.
You know, [snorts] allow the memories to
grow large, but
give agents tools to quickly index and
search for what's relevant. And finally,
like give agents autonomy when they're
writing to memories.
>> [snorts]
>> And if you were to go out and build this
system, it would work really well. You
would have the feeling of continual

learning because your agents would get
better at the individual um whatever
individual tasks you're doing.
However,
as with everything, this very neat idea
runs into some problems when you try and
scale it to production. So, we have and
I we have a concept for
theoretically what works. And when we
then think about scaling these to
production in environments where we have
many agents collaborating at the same
time, where they run over very long
periods of time, um and where
potentially these like code bases get
really complicated, all manner of
problems start to arise. And we've seen
these in production time and time again.
So, one a couple things just to like,
yeah, spark your imagination.
Think about multiple agents trying to
write to a memory file at the same time.
How do you manage that? Think about one
agent running into a problem and
deciding to like write to the
organizational worldwide contacts which
every other agent is currently reading
from. Like if something was incorrect
there, that would scale to all of your
agents and be pretty disastrous.
Um
and think about when you have humans and
agents collaborating on on memory
contacts together. Like how do you keep
track of what's going on?
The final problem is that memories can
go stale, of course. Something that was
relevant in the past uh might not be
relevant today or maybe it was written
incorrectly
or even maliciously injected by someone
trying to uh

prompt inject your agents to write bad things to memory. So, you have to have a lot of guardrails in place to make sure that these
nice autonomous memory systems actually work in production.
And so, I'm going to talk through a couple of key principles that we use when designing memory systems in production to make sure that we do get to use all of those nice effects that we've talked that we've spoken about.
So, the very first thing is versioning. So, when you're designing any kind of memory system, you need to be able to store versions
>> [snorts]
>> to keep track of what's going on, to allow you to roll back should you need to if uh a new update isn't particularly good. Um [snorts]
Additionally, you probably want to think about like what context was this update based on. So, which agent session, which transcript
resulted in me wanting to make this update.
Um
And additionally, like you might want to track like who did it, which agent, which human, etc., etc.
So, this is really important. The second thing is concurrency.
So, we've talked about, okay, what happens when I deploy thousands of agents all working off the same memory system.
And the solution that we've adopted here is to have this hashing system. So, when an agent decides that it wants to write an update to a memory, it takes a hash. It then drafts its edit. And then,

before it writes the update, it takes another hash. If those two things do not match, then the agent cannot write it because it means that some update was made in the meantime.
>> [snorts]
>> And in order to handle that, the agent repulls the memory,
drafts its new update, and then tries to commit this again.
So, these are the kinds of just engineering practices that allow you to scale multi-agent architectures, um scale memory to these kinds of architectures.
Another couple key principles. So, permissioning is really important. When you have large memory bases, you probably have a mixture of top-level organizational-wide knowledge. It might be like your key, uh like what your organization is trying to achieve, or key principles about the code base, which you've really carefully curated. All the way down to the level of like a scratchpad for an agent, where it writes down its working memory,
and [snorts] it's very like individualized. And all the way in between, you could have things for specific organizations or cross-sections.
And so, it's really important that you have guardrails when it comes to permissioning uh these memories. So, like I said, you wouldn't want one agent to just decide that it should update the organization-wide context, probably. You might want that as read-only. However, for its own scratchpad, you would want it to have write access.
Um

and yeah, yeah, that's that's
permissioning. Um
A final thing, which is kind of
peripheral, but still really important
when you design memory systems,
is portability.
So,
your curation of your memory systems is
going to be so important like throughout
the future. This is really, really
important organization, user, or like
work task specific context. And so, it's
likely that if you're putting a lot of
effort into curating this,
you want it to be accessible across
potentially multiple product surfaces,
um
>> [snorts]
>> and
access accessible by multiple systems.
So, designing it in a way with a clean
API in which it's portable and you can
access it is really important.
And
so, when you put all these things
together, we have the kind of
learnings we have from allowing agents
to creatively manage their memory, and
then these production level guardrails
that allow them to like reasonably use
all of those principles in practice.
And when you do this, you get very
effective results. So, just sharing here
a couple of learnings from what we've
seen deploying these large-scale memory
systems in production.
And
for example, we see you get better
accuracy, so you have this effect where
the second time the agent does the task,
it actually does it better
with higher results

because it's noted all of those memories about what went wrong.
Secondly, that then has second-order effects on the speed and latency, sorry, speed and cost of your agents running because they're then spending fewer tokens, they can be more easily one-shot these tasks because they actually know what they're doing.
And you'll see that it across all sorts of different processes, um agents are just able to do the task better and faster. Finally, having this process where your agents are starting to autonomously write their own memories,
frees up capacity and context for you as product developers potentially to focus on product wins while you know that the agents are doing this kind of self-learning, continual learning loop in the background. And yeah, once that infrastructure is set up really well, this this works very symbiotically.
Um
as ever,
we do then reach a new bottleneck.
And this specifically is about in-band memory.
So, in-band memory, as I mentioned before, is when agents are writing to and reading from memory within a specific session. So, if you think about Claude code, for example, um when you like spin up a new session,
it's it's largely like focusing on that specific context when it's reading to reading and writing from memory.
Um and this just architecturally or philosophically
has limitations in the general like agent fleets continual learning

objectives.
There's two two main reasons why.
First of all, is that you have this inherent split of focus and resources.
So, you're asking an agent to complete a task, but at the same time, you're also asking it to invest in memory curation, which would help it perform better in a future run.
So, when you put these things together, it's just a very difficult optimization problem, because how much capacity should an agent put into like helping future versions of itself versus doing the task that you actually asked it to do.
And also, there's like other effects like latency, for example.
The second thing is that the agents just have an inherent visibility limitation.
So, they only have the context of what's going on in their session.
As such, they just won't see patterns that happen across sessions. So, when you get frustrated that your agent keeps making the same mistake over sessions, it just doesn't understand how frustrating that is, because it has a new context window in each of those.
Secondly, when you're running multiple fleets of agents in different environments,
you these single agents just don't have the context of what other failures other agents are running into.
So, for these two reasons, we introduced this concept of some out-of-band memory curation,
and this helps to make these problems go away. And just to introduce an analogy for why this in theory should work,
I'd like you to think about a school,

for example, where you have lots of students that submit a lot of work, and you also have uh teachers that mark it and a head teacher that reviews everything.
This is a a system that we have in the real world for good reason, and it's because when you have certain individuals that have dedicated capacity for helping people learn, that's really effective. And when you also [snorts] have people that have visibility over the whole fleet of agents or learners, and they're [snorts] able to spot patterns and then kind of steer context, or let's say in this case the curriculum,
uh that's also really effective. So, as always, we kind of look to the real world world to think about how to build these systems.
Uh sorry, I also didn't touch on a final limitation, which is that memories go stale. So, you need something, some process that checks that everything that's written there is still correct.
And so, we introduce this concept of dreaming,
which is like a second o- second-order process over memory.
So, if we think about how that's been constructed, we have the like actual context, which agents reference and is has useful information, the memory processes, which allow agents to kind of autonomously manage that context themselves, and then dreaming, which is a process that runs in batch and asynchronously
with its own allocated resources to ensure that those memories themselves

are effective, up-to-date, um and
helping the agents learn over time.
So, what does dreaming look like?
Essentially, what we do
is we take an existing memory store, so
this is a collection of memories.
We then take a bunch of sessions or
transcripts from agent interactions over
a period of time,
and we give these together to an agent
which reviews all of the transcripts,
looks at the memory store, and starts to
identify patterns for where there could
be uplift in the memories.
It then outputs a new memory store
where there are proposed changes to the
existing memory store.
And what the agent is able to do, as I
mentioned, is spend tokens on solving
this problem of making agents learn
better,
identify patterns for where agents are
consistently failing,
and then propose changes for what might
make a more effective memory store, such
that next day when you run these agents
again, they're actually feel smarter and
they're running better.
To go back to my analogy, just to paint
some pictures of like what this could
look like in practice.
Let's imagine that the head teacher
reviewing all these transcripts notices
that
every geography student has incorrectly
answered a question. They're just all
writing like complete garbage to this
question.
The teacher notices that actually, by
kind of
in this case, analyzing the memory
store, that entire topic is missing from

the curriculum. So, what the teacher is able to do is notice that pattern, look at the memory store, and suggest a new change to that curriculum such that the next day when these agents run, they now have that information that they needed.

To give another example, the teacher might notice that um in a certain math exam, all of the answers are wrong in the same way. Uh all of the students are outputting radians when it's meant to be degrees. I don't know if anyone else in like GCSE maths had that problem, too. Uh but what they're able to do is give an instruction saying like this is how you should configure your calculators. And in the case of agents, that's like noticing in the transcripts that there's something wrong with the tool configuration. So, you might notice that something in the tool calls keeps failing.

And what's important here is that when we look at those transcripts, we're not just looking at kind of the passes of like uh responses between agent and the system or the user. We're also really scrutinizing like those tool calls and all of the other metadata that is really central to the agent's performance.

Finally, you could also notice something that's like fleet-wide or organization-wide. So, for example, like everybody's using too many m dashes and you don't like that. So, you want to add some organizational-wide um announcement or context change that says not to do that.

>> [snorts]

>> And so, I hope that paints a picture of like why this could be really, really effective.
And now I'll just talk about how you would go about designing such a system in production.
So,
you have some concept of your memory store, which is a concept which is a collection of memories.
Memories themselves might just be markdown files organized in this directory.
You then take a number of the transcripts. And like I mentioned, that is a mixture of like the back-and-forth passes between the agents as well as metadata on tools, uh other skills they used, etc.
And the way that we've designed it, we have the orchestrator deploy a fleet of sub-agents that go and analyze all those transcripts.
And one point worth making here is that when you design these systems, you have the ability to steer how these agents, which both write and coordinate dreaming, go about the problem. And by steering, I mean that you're able to tell them like, "In your specific case, these are the kinds of things I think are important and relevant. These are the kinds of things that are not important and relevant." So, you do have the ability there to start to curate that memory and dreaming process to your organization specifically.
And
the orchestrator then reviews all of the responses from the sub-agents and it then decides like where there are prevalent enough patterns that it thinks

this warrants a change in the memories.
From there, it proposes individual
changes to the memory store.
And in our case, the way that we design
this in production is
the agent will additionally give you
examples of transcripts where it's
noticed this pattern has happened and
also some stats on like how prevalent
this issue is and why this warrants
actually updating uh the memory store.
And so all of this works really neatly.
You get this output and you as the
individual
can decide where you want to accept
changes to the memory, um where you want
to reject them, etc.
And this works really effectively.
So together
we have these two processes that run in
parallel.
The first is memory
and these
agents are using some of their uh
like in-band contacts and in-band
resources to write to memory where they
think it's important. And this is neat
because it means that in the actual next
run, the next session, that agent will
be better. So there's a shorter time to
kind of seeing that change.
But inherently these agents have
competing resources when they think
about what to dedicate to memory, what
to dedicate to um completing the task,
and additionally a lack of visibility.
So on the other side, we have dreaming
which is this out-of-band process.
And this allows, again
broader visibility and dedicated
capacity
i.e. token spend which is specifically

directed towards helping agents learn
better.
And you might think okay, that sounds
really expensive.
>> [laughter]
>> Why would I want to chuck extra
resources at this?
But if we kind of go back to the
improvements we saw when you build
effective memory stores, actually you
can see a bunch of costs go down
because um agents are able to one-shot
things more effectively. Uh they have
more information that they need
in order to uh perform a task well.
So, to summarize,
at the very least, do the simple thing
that works. Context management uh makes
such a huge difference to your agent
performance.
Implementing things like a Claude MD
file, like skills, which I'm sure you've
heard about a bunch, and allowing agents
to autonomously manage these systems
themselves goes a really long way.
Once you think about scaling those
things into architectures with many
agents, agents that run over a very long
time,
situations where you will like continue
to work and develop on a workspace or
code base over a long period of time,
or very complex domains,
you should start thinking about adding
some features or some guardrails that
allow those agents to manage their
memory in a way that is safe,
verifiable, auditable.
I should also say here
that whilst this kind of
these kinds of practices are really
effective when it comes to coding tasks,

for example, this by no means is just
specific to coding. Like I use memory
all the time when I'm producing
presentations that has context on like
how I like to write things, how I like
my slides, etc., etc., and that develops
over time. So, this is really not coding
specific.
The final thing
is
if you really want to kind of like close
the loop here,
think about adding an additional
out-of-band process, like dreaming, as
we call it, to consolidate your memory
and
cut things that are no longer relevant,
add things that agents are missing, and
clean up and organize memory systems.
So, to close,
I want to say that this journey that
we've been on with Context Engineering,
a lot of this stuff has only happened in
this past year.
This is very much an open area of
research and development, and one which
we see huge value in the future.
So we're so excited to see the kinds of
things and contributions that you'll all
make to this space. So I encourage you
to keep thinking, keep learning, and
keep dreaming.
Thank you.
>> [applause]
>> Wonderful.
Thank you, Lamis. That was great.
Uh do we have any questions in the room?
Oh golly gosh. I'll dive straight here
first.
>> Thank you. Thanks for the presentation.
Um do you have any memory store
implementation that you would like to

suggest?
>> Papers
or papers?
>> memory storage implementation that you
would like us
>> Sorry, you said papers or No.
>> Any memory memory storage
implementation.
>> Oh, okay. Okay. That we like
>> like to suggest.
>> To solve what problem specifically?
>> Well, one thing is put things in files
that I have on my laptop. Yeah. But I
think I'm looking to something more
enterprisy. So what kind of solution do
you suggest over there?
>> Okay. Yeah. So I mean, maybe I was kind
of coy in the talk because we're not
allowed to make product call to actions.
But given that you asked, um
>> [laughter]
>> we have uh a lot of this references the
architecture that we used in our memory
uh
our memory infrastructure for our
managed agent solutions. And so when I
talk about these things about
productionizing um memory, so everything
like versioning
uh
hashing, etc., that's all available
within our memory and dreaming API
through Cloud Managed Agents. So if you
did want an out-of-box solution for this
kind of thing, that is where I would
point you to.
>> [snorts]
>> Um hi.
Uh
the
earlier on you talked about guardrails
and permissions.

Um and I think sure most of us have probably read the Cloud Code leak and the memory stuff. The dreaming stuff was definitely some of the most interesting in it.
But [gasps] how do you scale that enterprise if you've got hundreds of users with different permission sets? How do you make sure dreaming follows those same
guardrails?
If it's happening out of band and the context is different compared to say the context the agent might have when a user is using it.
>> So, just to check that I understand, um we have like some permissioning about what agents can access like in terms of memory. And then a separate like Yeah, so I mean I think this these things compose quite well, um because when you set up a dreaming procedure,
you decide exactly which session transcripts to attach. And so, you could build a process which mirrors whatever permissioning you have on the agents. Um so, yeah. I mean, if
that's to say that it's not the case that when you kind of trigger a dreaming job, it just takes like everything in a certain time period. You can configure it that way, but you can also just search over whichever transcripts have the same permission set as like whatever your memory store is and then make sure that that matches, if that makes sense.
>> Hi. Thank you. This was really, really interesting. I I found it fascinating earlier when you were mentioning about like versioning, concurrency,

durability, all of these all these
things. At what point are we like
reinventing databases from like first
principles again?
>> Um
>> [snorts]
>> yeah, this is this is an interesting
point.
I still think
like one of the things that
I think and I I this is a good reminder
for me that something I didn't say. Um
What we're trying to do here is like
thread the needle like sorry, find the
right boundary to draw between
kind of letting these agents
autonomously act and then also like
which things should just be kind of
programmatic um
things that are baked into the harness.
And so I think what you allude to is
like
first of all, we were just kind of like
letting these agents write in markdown
files and just like commit whatever they
wanted and now we're kind of seeing
having seen which primitives work really
well, we're thinking about like kind of
codifying that in the harness. And so
when we think about like the hashing or
the versioning, yeah, we are kind of
going back to the software engineering
principles that we've seen work well in
the past but in a way that kind of
autonomous agents can act and like
uh can in can interact with those really
effectively. So I think
to to some extent like we sort of arm
merging back into those practices but
that's because we have enough signal now
to know that those things should just be
done in a very deterministic way and

there's no need to reinvent the wheel. I hope that answered your question.
Perfect.
>> We're absolutely out of time. Thank you once again. Big round of applause for Lamis. Thank you.
>> [music]