Long Context Reasoning in LLMS

Cody Comyns

February 20, 2026

Table of Contents

Part 1: Hardware & Commercial Considerations of Frontier Labs
1.1: Unpacking the Research Motives of Frontier Labs
1.2: Memory Bottleneck & Next-Gen GPUs: Blackwell Ultra & Rubin
1.3: Implications of Next-Gen Hardware
1.4: White Collar Work: Largest TAM for Frontier Labs
1.5: Verifiable White Collar Tasks
1.6: Challenges of Deployment in Real World Settings
1.7: Putting Everything Together
Part 2: Long-Context Reasoning: Inference-Time Approaches
2.1: Agentic Workflows: Inevitability of Large Search Spaces
2.2: Basics of Retrieval-Augmented Generation (RAG)
2.3: Command F + Grep Approaches
2.4: Subagent Approaches
2.5: Kosmos & AI-Scientist-v2: Agentic Research Systems
2.6: Cursor's Browser & Anthropic's C Compiler: Agentic Coding Systems
Part 3: Interlude (Related Research Categories)
3.1: Addressing the Practicality of Continual Learning
3.2: Mechanistic Interpretability: a Kuhnian Crisis
3.3: Different Variations of Attention
Part 4: Long-Context Reasoning: Model-Side Capabilities
4.1: The Tradeoff between Reasoning & Context Length
4.2: Long-Context Benchmarks
4.3: Gistify & Dependency Invocation Rate (DIR)
4.4: A Note on Synthetic Data
Part 5: Concluding Thoughts
5.1: What Diffusion Might Look Like
5.2: OpenAI's Speedrun of Sora for Android
5.3: Coinbase vs. Ramp: Agentic Coding in Enterprise Codebase

Hardware & Commercial Considerations of Frontier Labs

Unpacking the Research Motives of Frontier Labs

Within the LLM paradigm, what things are guiding the research roadmap of frontier labs?

1) Hardware considerations: how do frontier labs get the most out of new generations of GPUs (ie. GB200 vs. GB300 vs Rubin)? What is possible now that certain compute/memory constraints are no longer relevant?

2) Commercial considerations: how are these labs going to justify their massive CapEx? What model capabilities are the most valuable to end consumers?

Memory Bottleneck & Next-Gen GPUs: Blackwell Ultra & Rubin

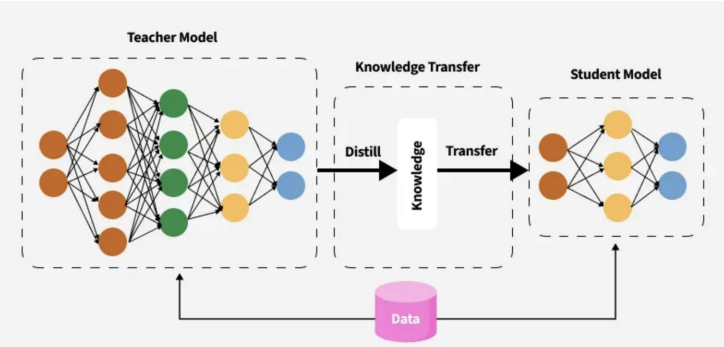
NVIDIA Generation	High-Bandwidth Memory (HBM)	HBM Bandwidth	NVLink Bandwidth	Deployed at Scale Date (Approx)
H200 (Hopper)	141GB	4.8 TB/s	900 GB/s	June 2024
GB200 (Blackwell)	192GB	8 TB/s	1.8 TB/s	May 2025
GB300 (Blackwell Ultra)	288GB	8 TB/s	1.8 TB/s	October 2025
Rubin GPU	288GB	22 TB/s	3.6 TB/s	Expected 2026 H2

- Current frontier models (GPT 5.2, Opus 4.5, Gemini 3, Grok 4.1) have been trained on GB200 GPUs or equivalent (Ironwood TPUs)
- Have yet to see frontier models from large-scale training runs with GB300s
- FLOP performance is also improving but memory-related issues are still the main bottleneck
  - Even with MoE approaches, trillion parameter models have to be split up between many different GPUs
  - Outside of parameters, the KV cache introduces additional memory constraints. Roughly speaking,
    - Memory used by KV Cache = (# tokens) x (# of layers) x (# of K & V heads per layer) x (d_head)
  - To put this in perspective, with group query attention (GQA) and FP8:
    - KV Cache for 1 million tokens for Llama-2-70B is approx ~160GB,
      - Where n_kv = 16 (8 key heads, 8 value heads), d_head = 128, L = 80

Implications of Next-Gen Hardware

Professional

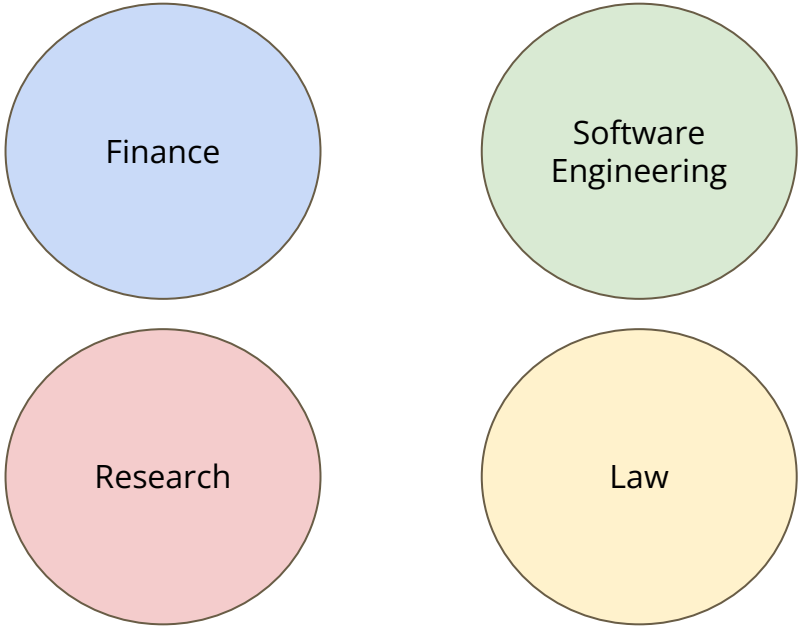
	GPT-5.2 Thinking	GPT-5.2 Pro
GDPval (ties allowed, wins or ties)	70.9%	74.1%
GDPval (ties allowed, clear wins)	49.8%	60.0%
GDPval (no ties)	61.0%	67.6%
Investment banking spreadsheet tasks (internal)	68.4%	71.7%



- Further scaling in inference-time techniques (ie. reasoning effort):
  - Main difference between GPT-5.2 Pro and GPT-5.2 (parallel test time compute ⟶ multiple full generations per prompt, verifier model grades each potential response and selects best one)
- Further scaling in training techniques
  - Param Count, RL, Data Filtering (train model on subsets of original training corpus and look at what subsets improve loss curve the most)
  - In practice, training runs usually performed on larger models and then distilled down to smaller models to reduce costs (using techniques like on-policy distillation)
    - Gemini 3 Flash performs nearly the same on all benchmarks as Gemini 3 Pro (benchmark's aren't everything though*)
- Longer context windows become feasible:
  - 1 million tokens ⟶ 10 million, 100 million+
  - Arguably the most important area of improvement

White Collar Work: Largest TAM for Frontier Labs

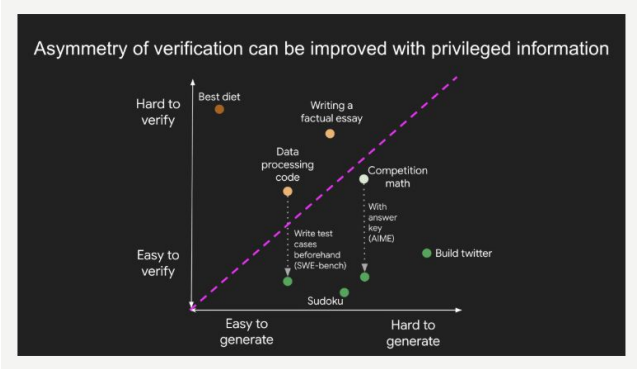
To justify CapEx, models need to start contributing in a meaningful way to the workforce, taking on larger and larger chunks of work. The biggest addressable market is white-collar work

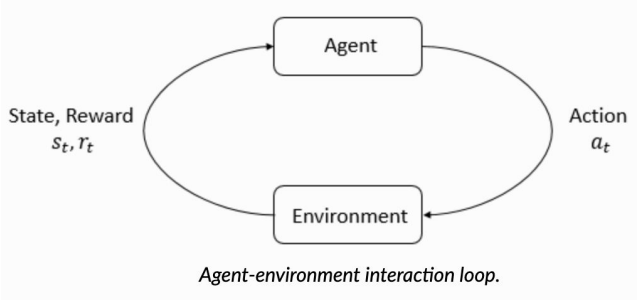


Verifiable White Collar Tasks

- Jason Wei's (led research team behind OpenAI's IMO breakthrough) Perspective on Verifiability: the ability to train AI to solve a task is proportional to whether the task has the following properties
  - Objective truth: everyone agrees what good solutions are
  - Fast to verify: any given solution can be verified in a few seconds
  - Scalable to verify: many solutions can be verified simultaneously
  - Low noise: verification is as tightly correlated to the solution quality as possible
  - Continuous reward: it's easy to rank the goodness of many solutions for a single problem

- The more verifiable a task is, the easier it is to improve a model's performance on that task
  - Software engineering: Does the code compile? Does the code pass the test cases?
  - Research: Was the model able to recreate the discovery? Was the model able to locate the relevant literature?
  - Finance: Was the model able to recreate the Excel file? Was the model able to identify the fraud?
  - Law: Was the model able to predict the outcome of the court case? Was the model able to locate the correct precedent?

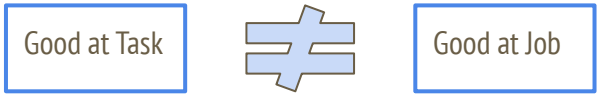




Agent-environment interaction loop.

Challenges of Deployment in Real World Settings

- Finite Context Window
  - Most valuable LLM applications involve strategy and planning (which in turn rely on a large search space much larger than 1m tokens)
- Context Bloat
  - Tool calls fill up the context window in a way that humans would automatically disregard as irrelevant in their own workflow
    - Ex. computer use agents needs to reason about the position of the cursor on the screen (to the granularity of pixels)
- Deterioration of Reasoning Capabilities as Context Grows (more on this later)
  - Rule following, dependency invocation, and many other measures of long term reasoning degrade as context window size increase
- Continual Learning (more on this later)
  - The model doesn't learn from past experiences, struggles to understand the intent and purpose of its actions without specific instructions



Human Action	Action Description	Agent Action
Click	Click at a specific position	**click**(x, y, button)
Middle Click	Middle click at a specific position	**middleClick**(x, y)
Double Click	Double click at a specific position	**doubleClick**(x, y, button)
Triple Click	Triple click at a specific position	**tripleClick**(x, y, button)
Mouse Move	Move mouse to a specific position	**moveTo**(x, y)
Drag	Drag mouse from one position to another	**dragTo**(x, y)
Scroll	Scroll vertically or horizontally	**scroll**(dx, dy) / **hscroll**(dx, dy)
Type	Type a string of text	**write**(text)
Press	Press a specific key	**press**(key)
Hotkey	Perform a combination of keys	**hotkey**(key1, key2)
Wait	Wait for a few seconds	**wait**()
Terminate	End the task with success or failure	**terminate**('success' or 'failure')

Available tools to a computer-using agent

Putting Everything Together

HARDWARE CONSIDERATIONS

Next Generation of Hardware Will Support:

- Scaling up of Parameter Count (pretraining)
- Scaling up of Train-Time Compute (post-training)
- Scaling in Test-Time Compute (reasoning budget)
- Increasing the Size of the Context Window

COMMERCIAL CONSIDERATIONS

3 Fundamental Things Holding Back Commercial AI Applications

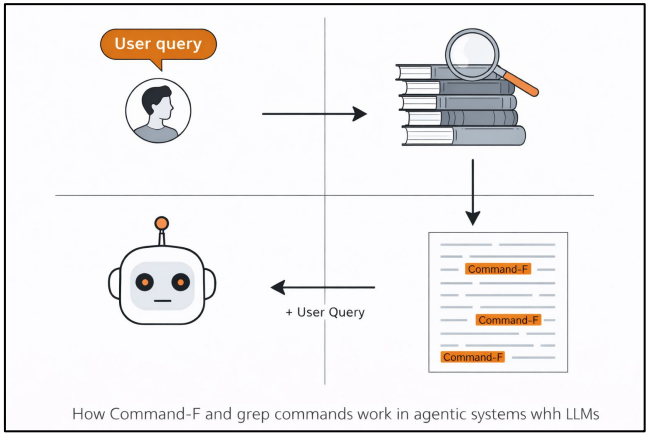
- Finite Context Window
- Continual Learning
- Deterioration of Reasoning Capabilities as Context Grows



Long-Context Reasoning in LLMs

Long-Context Reasoning: Inference-Time Approaches

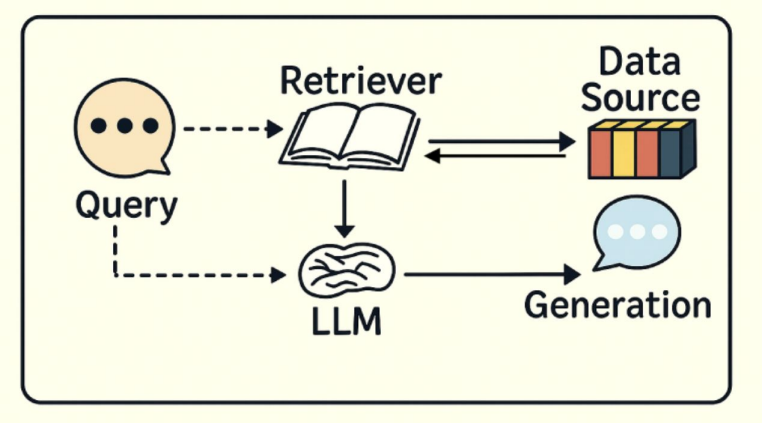
Agentic Workflows: Inevitability of Large Search Spaces



How Command-F and grep commands work in agentic systems whh LLMs

- Almost all agentic applications begin with a very large search space
  - Software Engineering: codebases
  - Law: contracts/statutes/precedent
  - Finance: articles/earnings calls/excel sheets
  - Research: publications/tooling/experiment results
- Even if context windows expand to 10M or 100M tokens from hardware improvements:
  - Reasoning capabilities deteriorate as context grows
  - Every additional token is exponentially more expensive (computational cost is n^2 with respect to n input tokens – except when KV caching)
- So, if you have a corpus of documents that is billions of tokens, it doesn't make sense to search over this entire corpus for every query
- Workarounds to this problem:
  - Retrieval-Augmented Generation (RAG)
  - Command-f/grep strategies
  - Sub-Agent Approaches

Basics of Retrieval-Augmented Generation (RAG)



2 Main Steps

- 1) Indexing Pipeline
  - Begin with your large corpus of data
  - Separate the data into chunks (smaller pieces of data)
    - Ex. 5 chunks per substack article
  - Create vector embeddings for each chunk (using an LLM)
    - Each chunk is represented by a 1000 dimension vector
  - Store in a vector database

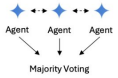
- 2) Retrieval & Generation Pipeline
  - Begin with user query
  - Create a vector embedding for the user query
  - Similarity search between query vector and article vectors
  - Retrieve the most similar chunks of text
  - Send the user query + the similar chunks of text to an LLM

Command F + Grep Approaches

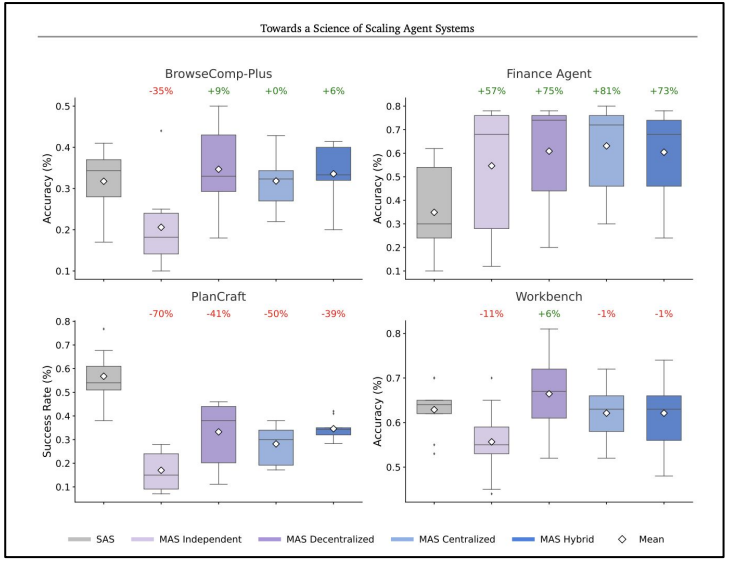


- Just like a human would but can span many more documents ⟶ include (let's say) 100 words of context around each command F occurrence for processing (processing could be parallel, OpenAI for example allows you to have 25 API keys so you could distribute the load)
- Cursor uses RAG, but Claude Code uses Command F/Grep
- RAG is not a strong moat (built a barebones RAG pipeline with 5,000 scraped substack articles in a weekend project)
    - Reason why Cursor pivoted and started training their own Composer 1 Coding Agent (Anthropic's subsidizing their tokens to convert people, they also benefit from people using their tools at scale because of cached tokens being cheaper)

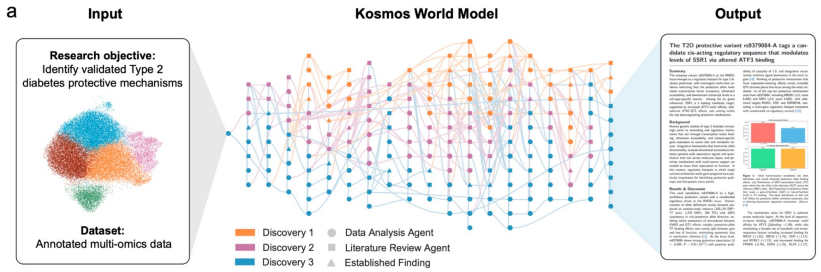
Subagent Approaches

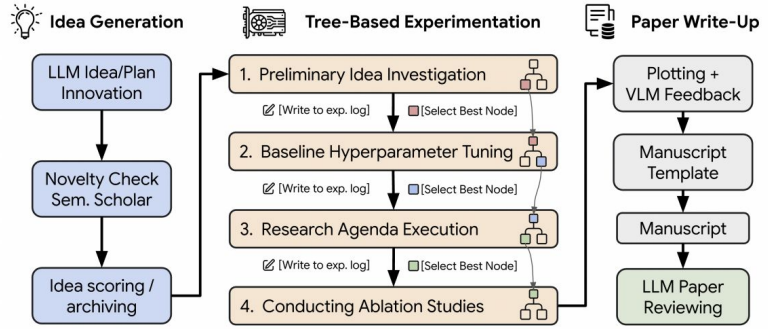
Characteristic	SAS	MAS (Independent)	MAS (Decentralized)	MAS (Centralized)	MAS (Hybrid)
Interaction Type					
LLM Calls	$O(k)$	$O(nk) + O(1)$	$O(dnk) + O(1)$	$O(rnk) + O(r)$	$O(rnk) + O(r) + O(p)$
Sequential Depth	k	k	d	r	r
Comm. Overhead	0	1	$d \cdot n$	$r \cdot n$	$r \cdot n + p \cdot m$
Parallelization Factor	1	n	n	n	n
Memory Complexity	$O(k)$	$O(n \cdot k)$	$O(d \cdot n \cdot k)$	$O(r \cdot n \cdot k)$	$O(r \cdot n \cdot k + p \cdot n)$
Coordination	Sequential	Parallel + Synthesis	Sequential Debate	Hierarchical Orchestrator	Hierarchical + Peer
Consensus	-	Synthesis	Debate	Orchestrator	Orchestrator

* k = max iterations per agent, n = number of agents, r = orchestrator rounds, d = debate rounds, p = peer communication rounds, m = average peer requests per round. Communication overhead counts inter-agent message exchanges. Independent offers maximal parallelization with minimal coordination. Decentralized uses sequential debate rounds. Hybrid combines orchestrator control with directed peer communication.



Kosmos & AI-Scientist-v2: Agentic Research Systems

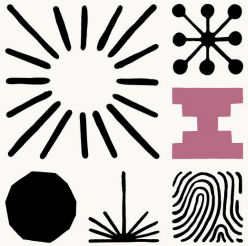




- Kosmos: an AI system by Edison Scientific capable of doing what takes a PhD student 6 months in a single day with ~80% accuracy
  - Can process up to 1,500 publications and write up to 42,000 lines of code to run experiments
  - Measured by number of papers it can read, number of lines of code it can write, and its ability to recreate publications not in training data
- AI-Scientist-v2: an autonomous research system (open-source code) developed by Sakana AI
  - Given a research topic or a research idea, their pipeline does tree-search literature exploration and then iteratively runs experiments, refining hypotheses
  - Their system autonomously generated a paper that passed peer review and was accepted at one of the most prestigious AI research conferences (ICLR 2025)
  - Note: this conference-accepted paper was generated with Claude Sonnet 3.5

Cursor's Browser & Anthropic's C Compiler: Agentic Coding Systems

- Anthropic's C Compiler
  - Stress test of a new feature their building called "agent teams" (essentially a swarm of agents working in parallel on a shared codebase)
  - 16 coding agents worked together over 2,000 Claude Code sessions and $20,000 in API costs, producing a 100,000 line-of-code compiler
  - Each agent works with its own copy of the codebase (git worktrees) and then merges their changes to the main codebase after their done with a feature
  - When stuck on a bug, agents maintain a running doc of failed approaches
  - LLM-written code frequently re-implements existing functionality, so tasked one agent with condensing any duplicate code
- Cursor's Browser
  - Ran hundreds of concurrent agents, producing >1M lines of code and spending trillions of tokens
  - Flat "equal agents" structure failed → agents avoided harder tasks (risk-averse)
  - Architecture that worked: planner agents (create tasks, spawn in workers), worker agents (implement tasks), and a judge agent
  - Initially built an integrator agent to handle conflict resolution between worktrees (found that this added more problems than it solved, worker agents were already capable of handling majority of conflicts)



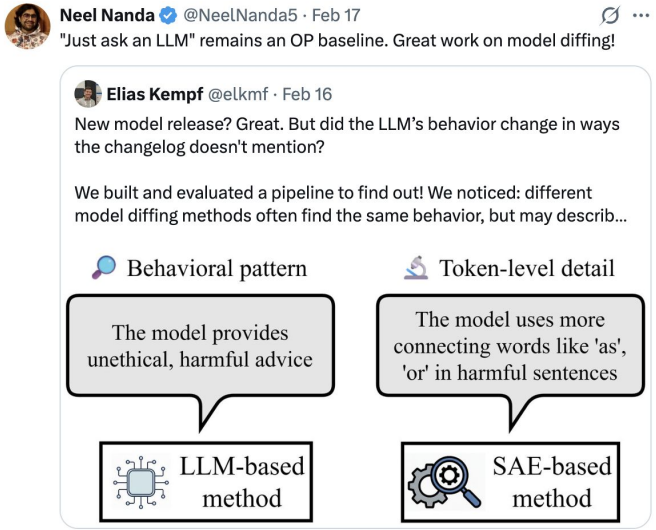
Building a C compiler with a team of parallel Claudes



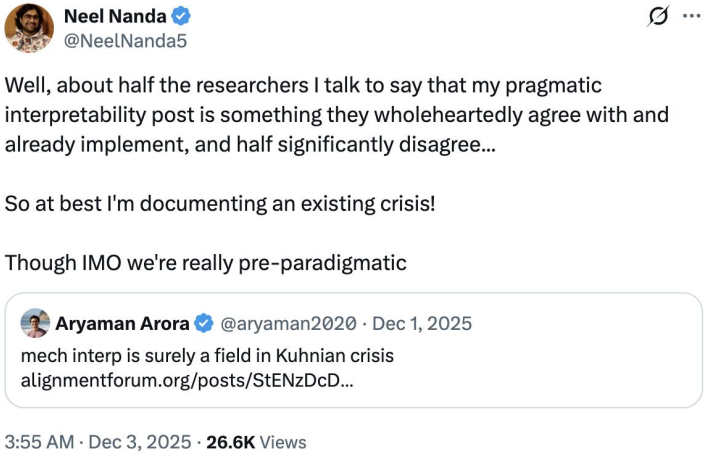
Interlude: Related Research Categories

Addressing the Practicality of Continual Learning

- AI researchers are obsessed with solving continual learning (especially in academia)
- Recall from my last presentation that the parameters/weights of a model are fixed and that activations change with every user query
    - Think of activations as:
        - (numbers associated with user query) x (static parameters)
- Broadly speaking, continual learning involves updating the model's weights dynamically (ie. continuous model training, rather than one big training run)
- Why I don't think continual learning is a worthwhile research bet:
    - Prohibitively expensive
    - Memory is already the fundamental bottleneck in inference (and this is just related to the activations, not parameter)
    - Each user would need to have a copy of their own weights stored somewhere physically, and loading these to and from GPUs would be very challenging
    - Alignment issues could arise from weight updates (CBRN risks, MechaHitler situation, models encouraging suicide or other harmful behavior)
- Dario's belief is that the current scaling paradigm for increased context window + in-context learning will work (and reusing cached context in a prompt is 90% cheaper)



Mechanistic Interpretability: a Kuhnian Crisis

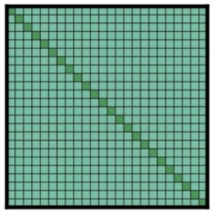


- The field of mechanistic interpretability is in absolute shambles
  - The ROI on research bets has been pretty awful
- Sparse Auto-Encoders (called SAEs) are often talked about as one of the most significant discoveries in the field
  - SAEs lose badly to simply re-prompting an LLM with its own response and asking if it contains dangerous information/behaviors
- Interpretability research often works by observing behaviors in toy models, but small architectural changes are made to models all the time (powering breakthroughs in reasoning and memory), and these breakthroughs negate a lot of interpretability progress
  - See next slide for examples of different variations of attention
- Obviously, field of interpretability has done well enough to the point where it is nearly impossible to get instructions on how to make a bomb or biological weapon
  - But alignment researchers can't say with 100% confidence that a model will never provide such an answer (if prompted in the right way)
- Shout out the Anthropic guy "the world is in peril"

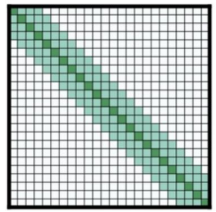
Different Variations of Attention

- **Sliding Window Attention (SWA)**
  - Alternate attention length for certain layers (ie. Gemma models have a ratio of 5 local layers: 1 global layer), where the local layers can only look back a maximum of 1028 tokens
- **Group Query Attention (GQA)**
  - Traditionally # Q heads = # K heads = # V heads
  - Here many Q heads share the same K and V head (ex. 4 Q's, 1K, 1V)
  - Cuts KV Cache Memory by a factor (# Q heads / #KV Heads)
- **Multi-Head Latent Attention (MLA)**
  - Comes from DeepSeek V2/V3
  - Compress each token into a small latent vector (using low rank factorization)
  - Cache only this latent
  - When you need K/V for attention you decompress that latent into per-head K & V
    - Traditionally: $K = XW_K$ and $V = XW_V$
    - Here: $Z = XW_{DOWN} \rightarrow K = ZW_{K,UP}$ & $V = ZW_{V,UP}$
  - Massively reduces KV cache memory & shifts the bottleneck from memory bandwidth to compute (DeepSeek reports ~90%+ reductions in some setups)





(a) Full n^2 attention



(b) Sliding window attention

Long-Context Reasoning: Model-Side Capabilities

The Tradeoff between Reasoning & Context Length

- Why is it hard to teach LLMs how to reason over long contexts
  - Pretraining: limited long context data available (mostly books and codebases)
  - Posttraining: difficult making RL environments for long context tasks (and reward signals get messier)
  - Long context reasoning benchmarks are very scarce (and very hard to build)
- Training solely on long context data degrades the models' performance on short context tasks and vice versa
  - Best training data split: 60% long-context data/40% short context data
- Even within the realm of long context data, some types of long context data are better than others at instilling long context reasoning abilities

Long Data (60%)	Long-Context							Short-Context
	Recall	RAG	Re-rank	ICL	QA	Summ.	Avg.	Avg.
CommonCrawl	84.1	53.3	28.1	67.5	**35.2**	37.0	50.9	66.5
ArXiv	90.3	51.8	28.0	68.0	33.7	36.7	51.4	67.5
Books	94.9	**53.9**	**30.7**	**72.2**	33.2	**37.7**	53.8	65.5
Code Repos	**99.2**	**53.8**	29.0	61.2	34.7	36.2	52.3	65.9
Books/Repos/ArXiv 1:1:1	98.3	53.9	29.4	66.9	35.5	35.5	53.3	66.9
Books/Repos 1:1	96.0	54.9	29.4	73.9	35.7	37.9	**54.6**	65.5

	HSwag	MMLU	ARC-c	WG	GSM8K
Llama-3-8B	82.1	66.5	59.4	77.1	44.7
+ PE	81.5	64.7	58.1	75.5	40.1
+ SlimPajama	81.0	63.1	57.8	75.1	40.6

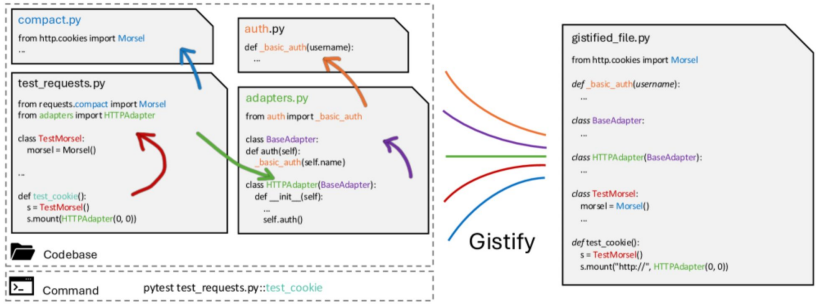
- HSwag, MMLU, ARC-c, WG, and GSM8K are short-context benchmarks
- Llama-3-8B is the base model
- Positional Extrapolation (PE) is a training-free technique to extend context window of base model
- SlimPajama represents a version of base model fine tuned on a long-context dataset

Long-Context Benchmarks

Models	Claimed Length	Effective Length
Llama2 (7B)	4K	-
Gemini-1.5-Pro	1M	>128K
GPT-4	128K	64K
Llama3.1 (70B)	128K	64K
Qwen2 (72B)	128K	32K
Command-R-plus (104B)	128K	32K
GLM4 (9B)	1M	64K
Llama3.1 (8B)	128K	32K
GradientAI/Llama3 (70B)	1M	16K
Mixtral-8x22B (39B/141B)	64K	32K
Yi (34B)	200K	32K
Phi3-medium (14B)	128K	32K
Mistral-v0.2 (7B)	32K	16K
LWM (7B)	1M	<4K
DBRX (36B/132B)	32K	8K
Together (7B)	32K	4K
LongChat (7B)	32K	<4K
LongAlpaca (13B)	32K	<4K

- Needle in a haystack: glorified command f questions (don't require a lot of conceptual/logical reasoning)
- RULER: measures long-context reasoning across 4 different tasks:
  - Retrieval: diverse types and quantities of needles
  - Multi-Hop Tracing: variable tracking (across chains of logical steps)
  - Aggregation: common/frequent word extraction (that are used in many different contexts)
  - Question Answering: add distracting information to passages relating to concepts that appear in a reading comprehension QA exam
- Nowadays: more common/useful benchmarks are agentic benchmarks
  - MCP-Atlas: evaluates tool-use competency for agents using MCPs (emphasis on multi-step tasks and cross-tool orchestration)
  - τ2-bench: evaluates agents in long, tool-using customer service dialogues
  - Browse-Comp: agentic web search agents (ability to locate multi-hop information)

Gistify & Dependency Invocation Rate (DIR)



- Gistify:
  - Given a codebase and an entry point to a method, can an LLM recreate the functionality of that method in a completely self-contained file outside the codebase (ie. get the gist)
  - Complexity here: methods are often chained together to other methods/classes via dependencies (ie. import pandas)
  - On a subset of dependency-rich gistify tasks, performance of LLMs dropped from 43% to 21%
- Dependency Invocation Rate (DIR):
  - Models will often rewrite methods and clutter up the codebase, rather than reuse existing functionality
  - Couldn't find any publications reporting results on frontier LLMs (only older ones, likely cause it's hard to measure and write tests for)
  - One publication mentions that the more context it gets on the relevant dependencies (ie. code implementation of dependencies), the better the results, issue with this is context rot though
- Any implementation of a coding harness (Cursor, Anti-Gravity, Claude Code) uses dependency graphs to feed the model the relevant dependency context
  - Also very relevant for continuous integration/development systems

A Note on Synthetic Data

- From a pretraining perspective, we want long-context training data that is dependency rich (for codebases) and involves complicated logical and conceptual mappings
- If the models aren't good at this and we've exhausted existing data sources, how do we get more data ⟶ synthetic data, constructed from common (& practically useful) system design patterns
- Take a high-level problem, break it down into smaller modular pieces, have an LLM create these smaller pieces, combine the smaller pieces into long-context data (to make the model better at this specific system design pattern)
- Good way to think about this:
    - Models are very good at implementing code assignments (entry-level SWE tasks)
    - Models struggle at systems design patterns (senior-level SWE patterns)
    - Intuitively, this makes sense because all relevant context for an implementation task might span 5,000 lines of code vs. all relevant context for systems design might span 1M lines of code



Concluding Thoughts

What Diffusion Might Look Like

- Humans learn a lot of implicit things about their colleagues, their company, and the way that things are done
- If you fully articulated these things to a model, they would probably be able to recreate your output, but in the time that you onboard "this colleague", you could probably just get the job done faster yourself
- This narrative is true if you're just using the Chat-GPT user interface

- Now that software is so cheap to build, it's possible people will start encoding this implicit knowledge in bespoke software (relying on LLM APIs, rather than LLM UIs)
- Are people willing to build automations that fundamentally replace what they do (massive perverse incentive)? On the other side of things, would an employer ever fire someone capable of building automations like this?
- Maybe there's an explosion of freelancers who just go from company to company and make bespoke software/scaffolding for them to orchestrate agents and automations (maybe this is what SWEs become). Alternatively, there may be an explosion of observability/recorded workflow tools that process people's workflows, extract the implicit "know hows", and then have a coding agent design scaffolding based on this

December 12, 2025 Engineering Company

How we used Codex to build Sora for Android in 28 days

By Patrick Hum and RJ Marsan, Members of the Technical Staff

American computer architect Fred Brooks famously warned that “adding more people to a late software project makes it later.” In other words, when trying to ship a complex project quickly, adding more engineers can often slow down efficiency by adding to communication overhead, task fragmentation, and integration costs. We leaned into this insight instead of ignoring it; we assembled a strong team of four engineers – all equipped with Codex to drastically increase each engineer’s impact.

Working this way, we shipped an internal build of Sora for Android to employees in 18 days and launched publicly 10 days later. We maintained a high bar on Android engineering practices, invested in maintainability, and held the app to the same reliability bar we would expect from a more traditional project. (We also continue to use Codex extensively today to evolve and bring new features to the app).

7:04

Chintan Turakhia
5.6K posts

Follow

Posts Replies Highlights Videos Photos

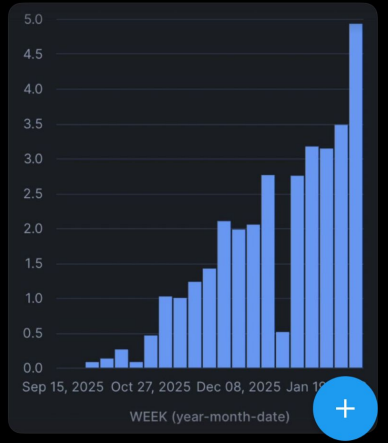
Chintan Turakhia @chi... · 2/6/26

We hit a real milestone today.

5% of all PRs merged now come from our in-house background agents.

Last year we built an agentic harness: a Slack-native framework with the same tools and context as our engineers.

Tag the agent in any thread → it plans, debugs, and ships a PR. Show more



6:51

Post activity

Quotes Reposts

Aakash Gupta @aakashg... · 2/10/26

57% of merged PRs at Ramp in the last 24 hours came from a background agent. Most companies haven't even started.

The architecture Ramp built matters. Their agent Inspect runs in sandboxed VMs on Modal with full access to everything a Ramp engineer has: Sentry, Datadog, GitHub, CI/CD, feature flags, databases, live preview environments. The agent doesn't just write code. It runs tests, checks telemetry, verifies frontend changes with screenshots, and opens PRs that pass the same review bar as human-written code.

This is why the number is so high. Model intelligence was already sufficient. Environment parity was the missing piece. Once agents have the same context and tooling humans have, adoption compounds because engineers stop treating the agent as a side tool and start treating it as a parallel teammate running unlimited concurrent sessions.

The product development implications are massive. PMs at Ramp now use Inspect during QA to make changes in real time instead of writing tickets. Designers can ship fixes without waiting for sprint capacity. The marginal cost of implementing a small change drops to near zero, which means the backlog starts to dissolve.