How LLMs Work

Context Is King

13 February 2026

An LLM is a next-token prediction engine
operating over a context window.

Every "feature" — system prompts, tools, MCP,
RAG, agents, skills —
is just a mechanism for putting text into that
window.

Today's Agenda

1. **First Principles** — tokens, attention, the context window
2. **Context Is King** — what fills the window and why it matters
3. **Agents** — systems that manage context automatically
4. **Putting It Together** — where this lands and what to learn next

It's Not as Complicated as It Looks

The paper

$$\mathcal{L}(y, \hat{y}) = -\sum_{i=1}^{N}\sum_{c=1}^{C} y_{i,c}\log(\hat{y}_{i,c})$$

"Negative log-likelihood of the true class under the predicted distribution"

The code

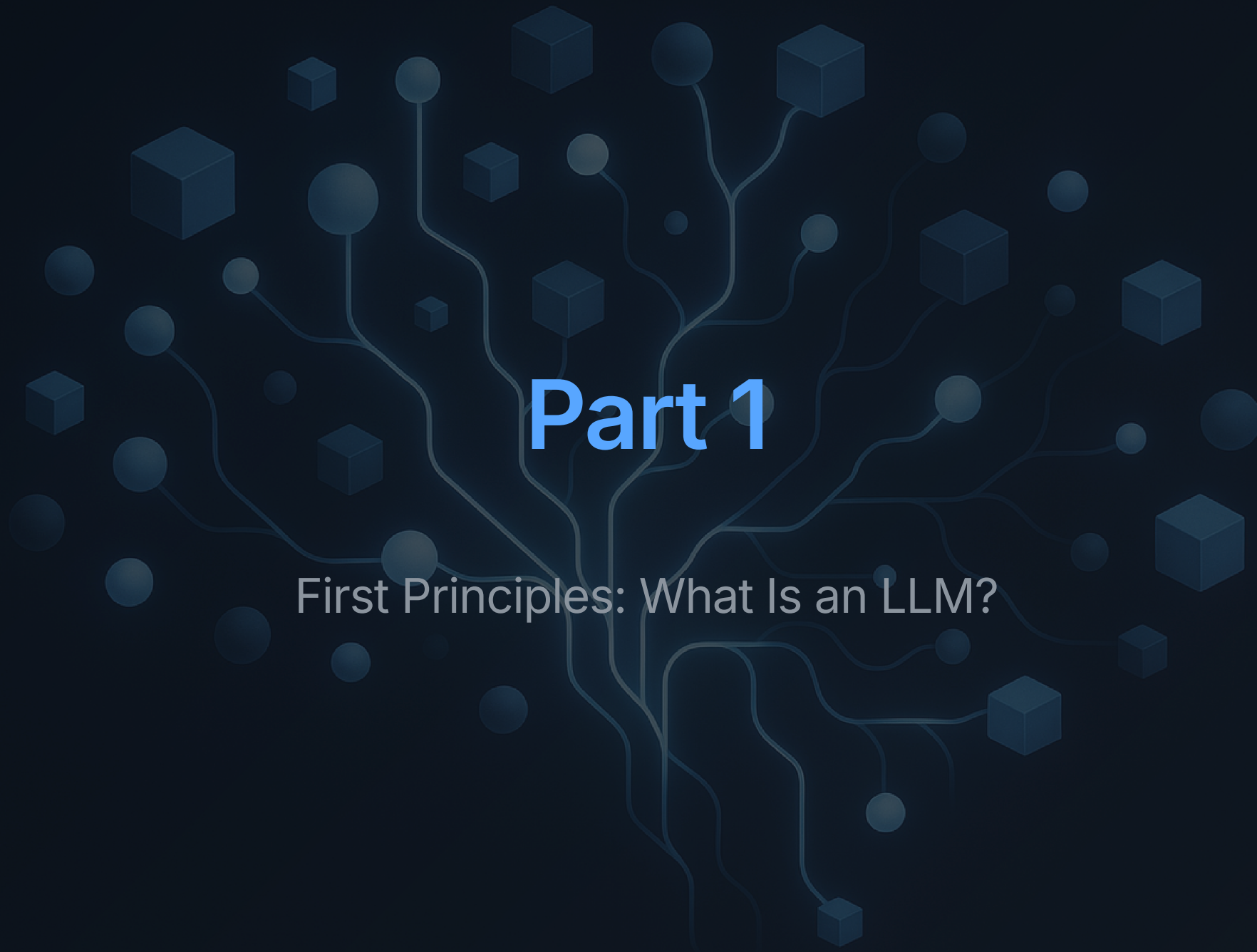
```
# true      = [0, 0, 1, 0]    ← correct answer
# predicted = [0.1, 0.2, 0.6, 0.1] ← model's guess
loss = -(true * log(predicted)).sum()
#          ^ element-wise multiply (×)
```

The `*` multiplies each pair — zeros cancel everything except the correct answer's confidence.

Cracking the Cryptic

They say	They mean
AI researcher	Software engineer
AI lab	Software company
Token	A chunk of text (word or part of a word)
Large Language Model	Text prediction program
Latent space	A grid of numbers
Neural network	Functions chained together
Training	Adjusting numbers to reduce errors
Inference	Running the program
Parameters	The numbers it learned
Fine-tuning	More training on specific data

> **nanoGPT:** *a full working LLM in* **~600 lines** *of Python.*



Part 1

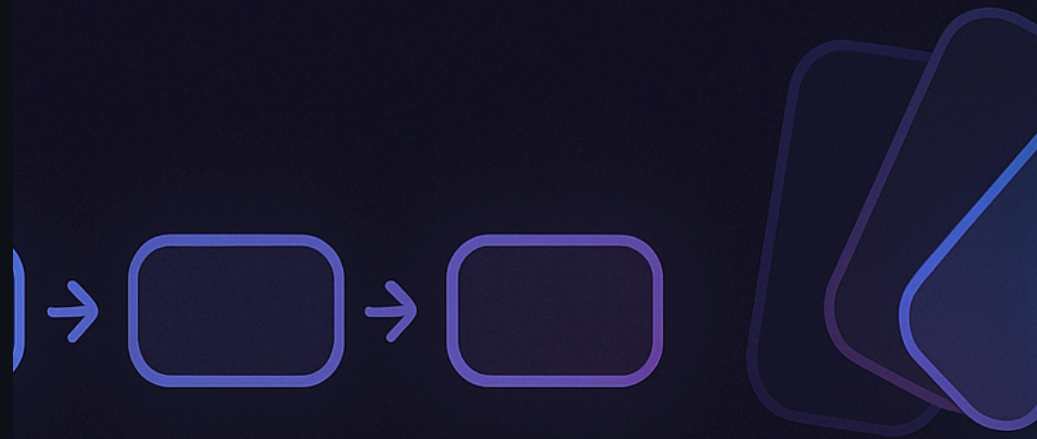
First Principles: What Is an LLM?

Guess the Next Token

Given tokens → predict next token

```
Input:  "The cat sat on the"
  → P(mat)=0.23, P(floor)=0.18 ...
  → sample → "mat"
```

Ref: Raschka, *LLMs-from-scratch* — github.com/rasbt/LLMs-from-scratch · Karpathy, *Neural Networks: Zero to Hero* — excellent starting point for the fundamentals



Tokens ≠ Words

Text is split into **sub-word chunks** by a tokeniser.

Text	Tokens	Count
`"unhappiness"`	`["un", "happi", "ness"]`	3
`"ChatGPT is great"`	`["Chat", "G", "PT", " is", " great"]`	5
`"123456"`	`["123", "456"]`	2

Rule of thumb: 1 English word ≈ 1.3 tokens

💡 Demo: Show a tokeniser in action — platform.openai.com/tokenizer

⛄ Temperature (randomness) 🔥

Sampling controls how the next token is chosen:

- **Temperature 0** = greedy (highest probability wins) = **fully deterministic**
- **Temperature > 0** = adds randomness = creative, varied output
- **top-k**, **top-p** = additional randomness controls

"LLMs aren't deterministic" — that's a *setting*, not a limitation. Randomness is usually desirable.

The Transformer & Attention

Every modern LLM uses **self-attention** — each token looks at **every other token**.

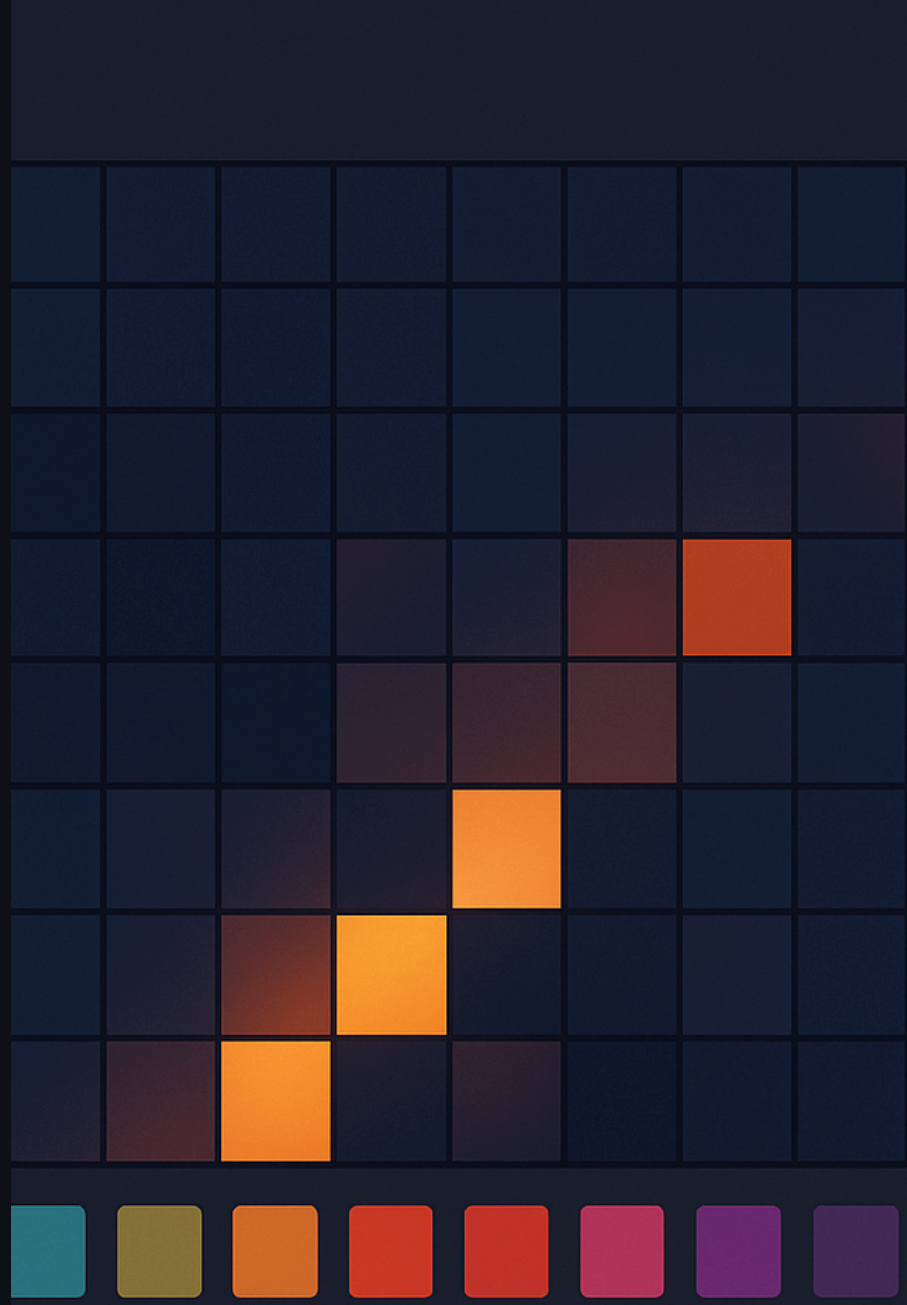
Tokens:	128	1K	8K	128K
Pairs:	16K	1M	67M	16B

n tokens = n^2 pairwise relationships

→ double the context = **4× the compute**

This quadratic cost is the **fundamental reason** context windows have limits.

Ref: Jay Alammar — The Illustrated Transformer · Raschka, Ch 3 — Attention · Barbero et al. (2025)



The Context Window

The central concept in this talk

What it is

- **ALL** the model can see
- No hidden memory, no state
- Everything serialised as tokens

What this means

- Model is **stateless** — each call starts fresh
- Not in the window = **doesn't exist**

> 🎯 **The context window IS the model's entire reality.**

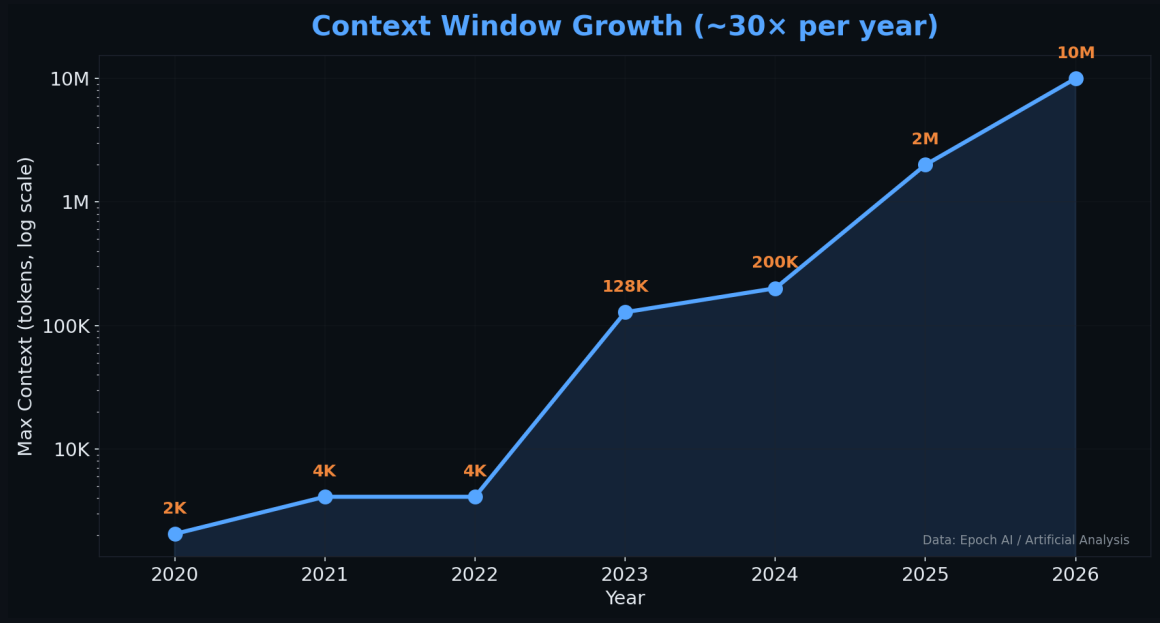
Context Windows Have Grown Fast

But can they *use* it?

- 80% accuracy: ~250× growth/yr
- Real tasks ≠ NIAH benchmarks
- Context rot is real

Scale reference:

- Novel ≈ 100K tokens
- Full codebase ≈ 500K–2M tokens *(~50K–200K lines of code)*
- All of Wikipedia ≈ 4B tokens



Interactive: epoch.ai/data-insights/context-windows · artificialanalysis.ai

Context Rot: Bigger ≠ Better

Chroma tested 18 LLMs — **only input length varies**, task complexity constant:

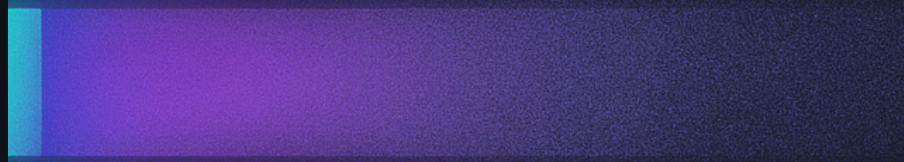
Key findings

- NIAH (lexical retrieval) ≈ solved
- **Semantic** retrieval degrades
- **Distractors** make it worse

Implications

- NIAH scores mislead
- More tokens = more noise
- **Curation > capacity**

> **Working memory, not storage.** *Overwhelm it → performance degrades.*
> *Opus 4.6:* **93%** → **76%** *accuracy — same task, just*



Part 2

Context Is Everything

Building the Context

Layer	Owner	When
Model provider prompt	Provider	Always
System prompt	UI	Always
copilot-instructions.md	User	Always
.instructions.md	User	Path match
AGENTS.md	User	Nearest in tree
Tool definitions	UI	Always

Layer	Owner	When
MCP server schemas	User	Config'd
Loaded SKILL.md	Agent	On demand
Custom agents (.agent.md)	User	Invoked
Conversation history	UI	Compacted
Retrieved context	Agent	Tool calls
User's current message	User	Always

All markdown → tokens → context. The model doesn't distinguish layers.

💡 What's in the System Prompt?

System prompts reveal what's loaded before you even type:

Copilot CLI

- Tool definitions (bash, view, edit...)
- Session context + environment
- Custom instructions from repo
- Behavioural rules + tone
- Skills + agent config

ChatGPT / Claude

- Thousands of tokens each
- Personality + safety rules
- Tool schemas (DALL-E, browsing...)
- Knowledge cutoff info
- Prepended to **every request**

💡 Demo: github.com/asgeirtj/system_prompts_leaks ★ 31k

"Features" Are Just Context Injection

Feature	What it actually does	Example
System prompt	Text prepended to every request	Leaked system prompts
copilot-instructions.md	Repo-wide rules, auto-loaded	Docs
Tools (function calling)	JSON schemas describing actions	OpenAI function calling
MCP servers	Tool schemas from external process	MCP spec
RAG	Retrieves text chunks, injects them	Anthropic RAG guide
Skills	Markdown lazy-loaded when relevant	Copilot skills
Sub-agents	Fresh window + tailored prompt	Agents & sub-agents
Conversation history	Prior turns, often summarised	Context compaction
Few-shot examples	Example pairs pasted into context	Few-shot prompting
Reasoning / thinking	Chain-of-thought tokens generated & read	Extended thinking

The Scale Problem

Real-world overhead from Anthropic's engineering team:

134,000

tokens of tool definitions
in Anthropic's internal setup
before any conversation

5-MCP-server example

Server	Tools	Tokens
GitHub	35	~20,000
Slack	11	~8,000
Sentry	5	~5,000
Grafana	5	~5,000
Splunk	2	~2,000
Total	**58**	**~55,000**

Add Jira → **100k+ overhead**

Ref: Anthropic — "Advanced Tool Use" (2025)

The Tool Search Tool

Instead of loading 58 tools (~55k tokens), give the model **one search tool** (~500 tokens):

Context reduction

85%

fewer tokens

55,000 → 3,000

Accuracy *improved*

Model	Before	After
Opus 4	49%	**74%**
Opus 4.5	79.5%	**88.1%**

Less noise = better decisions

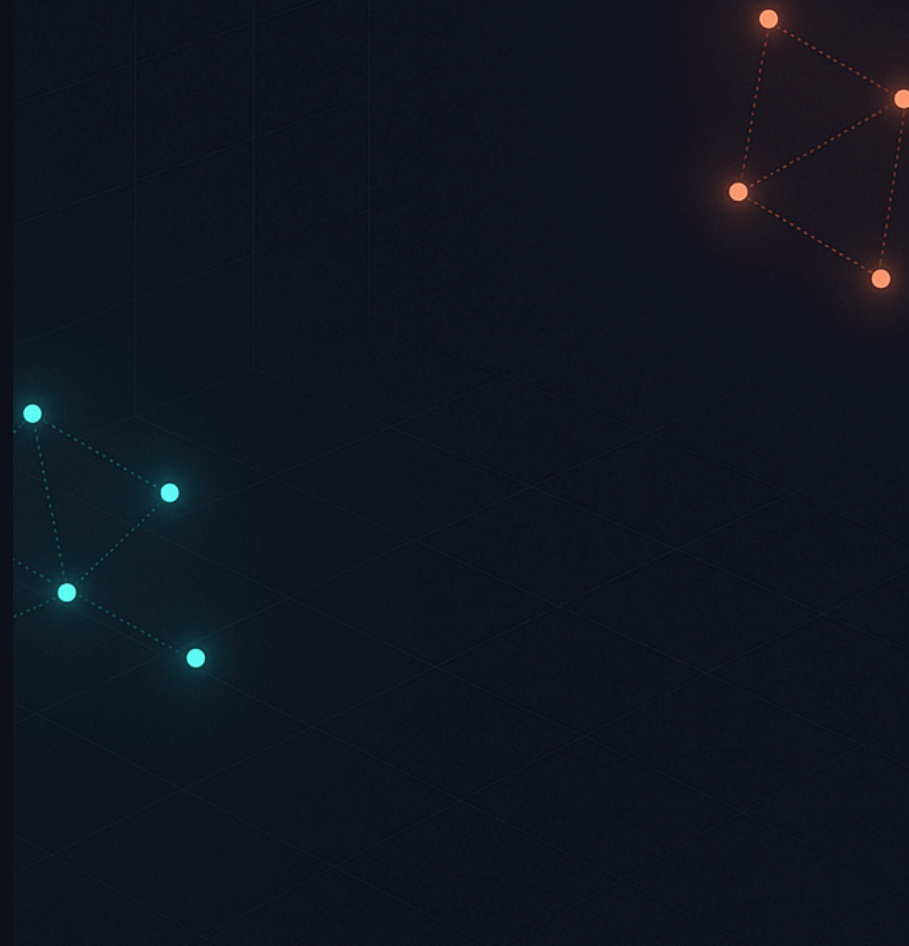
Ref: Anthropic — "Advanced Tool Use" (2025)

Finding the Right Context

Two strategies, same goal: get the right text into the window.

	RAG	Agent tools
How	Pre-indexed vectors → similarity search	`grep`, `glob`, `view` at runtime
Speed	Fast lookup	Slower, multi-step
Freshness	Can go stale	Always current
Who decides	Your search setup	The model itself

Both are just **context injection** — the difference is *when* and *who* picks what goes in.



Context Engineering > Prompt Engineering

Era	Focus
2023	"Write a good prompt"
2024	"Give it examples + tools"
2025–26	**"Engineer the entire context pipeline"**

> *"Context engineering is the delicate art and science of filling the context window with just the right information for the next step."*
> — **Andrej Karpathy**

> *"The art of providing all the context for the task to be plausibly solvable by the LLM."*
> — **Tobi Lütke**, *Shopify CEO*

The Goldilocks problem: 🔻 Too little → hallucinates · 🔺 Too much → diluted attention · ✅ Just right

simonwillison.net/2025/Jun/27/context-engineering · Anthropic — "Effective Context Engineering"

Part 3

Agents: Systems That Manage Context

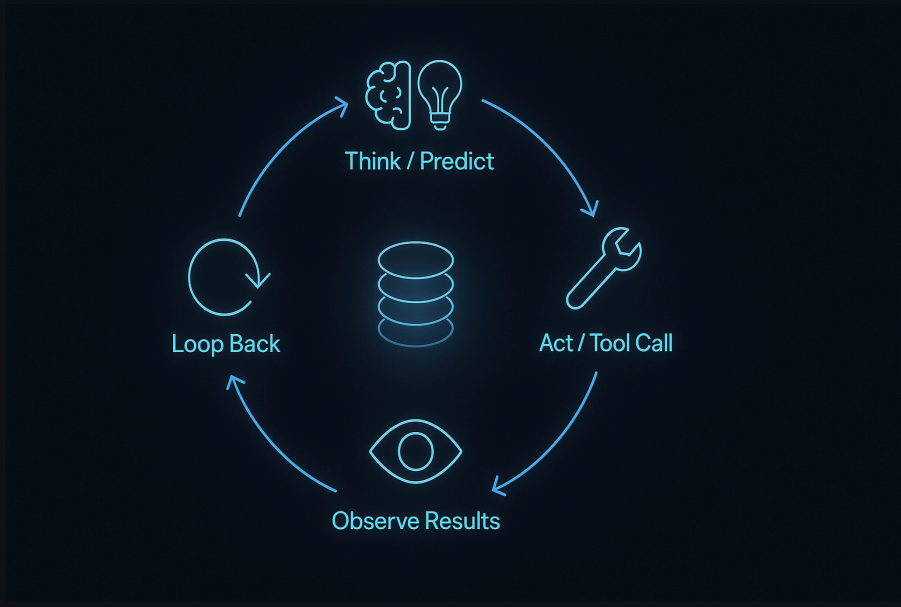
The Agent Loop

> *"An LLM that runs tools in a loop to achieve a goal."*
> — **Simon Willison**

Think → Act → Observe → Loop

Each iteration, the context grows.
Effectiveness depends entirely on what's in that window.

Ref: Willison — "2025: The Year in LLMs"



Managing Context

When agents work 30+ minutes, context fills up. **Three strategies:**

1. **Compaction** — summarise conversation, start fresh
2. **Structured Note-Taking** — external files for persistent memory
3. **Sub-Agents** — fresh window for focused sub-tasks

Ref: Anthropic — "Effective Context Engineering" · Fowler — sub-agents

Strategy 1: Compaction

Summarise conversation → start fresh with summary

- Preserves decisions, discards verbose tool output
- Copilot CLI does this automatically
- Compression ratio: ~10:1 typical

Strategy 2: Structured Note-Taking

Agent writes to external files, reads back later

- Plan.md, to-do lists, decision logs
- Survives context resets
- Copilot CLI: session workspace for persistent notes

Strategy 3: Sub-Agents

Fresh context window for focused sub-task

- `.agent.md` file defines specialised persona + curated tools
- Might use 50k+ tokens exploring
- Returns concise summary to parent agent

The Convergence

All major coding agents → **same pattern**, different names:

Concept	Claude Code	GitHub Copilot	Cursor
Always-on rules	`CLAUDE.md`	`copilot-instructions.md`	`.cursorrules`
Path-scoped rules	Rules (`*.ts`)	`.instructions.md` + `applyTo`	"Apply intelligently"
Lazy-loaded context	Skills (`SKILL.md`)	Agent Skills (`SKILL.md`)	Evolving → Skills
Specialised agents	Sub-agents	Custom Agents (`.agent.md`)	Sub-agents (new)
External tool access	MCP servers	MCP servers	MCP servers
Lifecycle scripts	Hooks	Hooks	Hooks (new)

Same mechanism: curate what goes in, control when it loads.

Calibrating Trust

> *"Think in probabilities and choose the right level of human oversight for the job."*
> — **Martin Fowler**

In practice (Feb 2026), they're remarkably reliable:

- Follow instructions consistently when context is well-engineered
- Mistakes are mostly edge cases, not routine failures
- The better the context, the more you can trust the output

Match oversight to stakes:

- 🟢 Most tasks — let it run, review the diff
- 🟡 Sensitive data — log and spot-check
- 🔴 Irreversible actions — require human approval

Ref: Fowler — "Beware: Illusion of control" · HumanLayer — Function Stakes

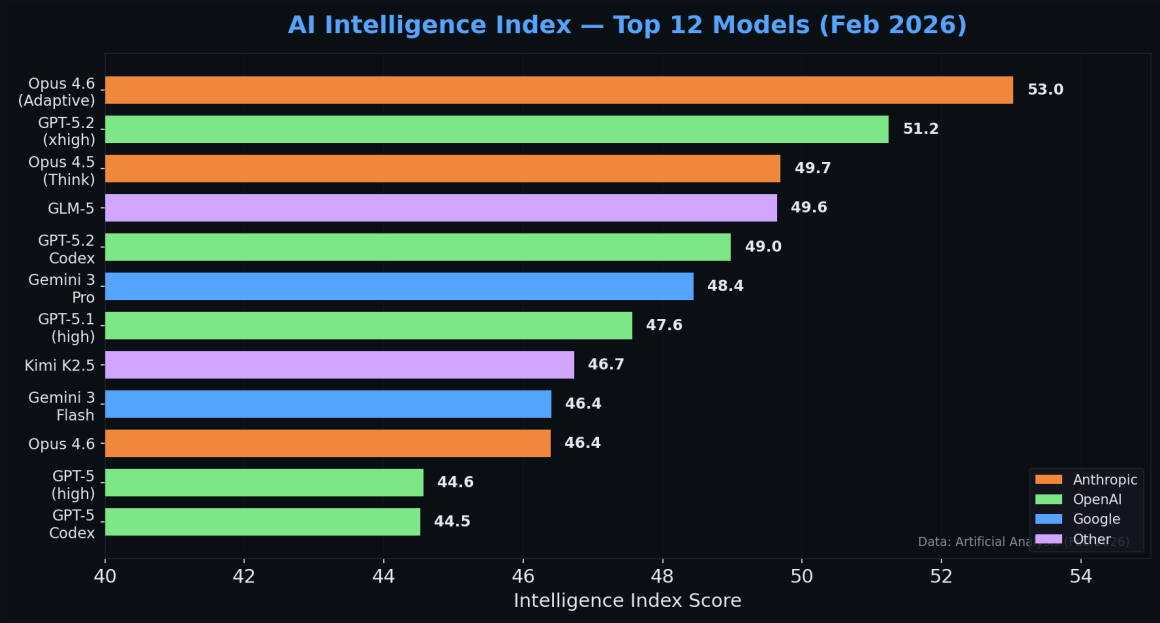
Part 4

Putting It Together

Coding Agent Progression

SWE-bench Full: % of real issues resolved

Model	Score
GPT-4 *(Oct 2023)*	**1.7%**
GPT-4o *(Jun 2024)*	**22%**
Claude 3.5 Sonnet *(Oct 2024)*	**49%**
Opus 4.6 (Thinking)	**79.2%**
Gemini 3 Flash	76.2%
GPT 5.2	75.4%
Opus 4.5	74.6%



The formula: Reasoning + tools + context engineering

Context Engineering Is the New Core Skill

Every tool, framework, product — **all solving the same problem:**

What text goes into the context window?

It transfers

- Copilot → Cursor → Claude → Gemini
- Chat → agents → code review → CI
- Models change; context doesn't

It compounds

- Good instructions → every conversation improves
- Good tool design → every agent loop improves
- Good code structure → every AI interaction improves

HumanLayer's *12 Factor Agents*: "Own your context window" — actively curate, don't leave it to frameworks.

Ref: HumanLayer — 12 Factor Agents · humanlayer.dev

What We Covered *(& what to remember)*

Part 1 — LLMs predict the next token. That's the entire mechanism.

Part 2 — The context window is all they see. Every "feature" is just putting text into it.

Part 3 — Agents are loops that manage context: compaction, notes, sub-agents.

Part 4 — SWE-bench: 1.7% → 79% in two years. Context engineering is the core skill now.

📣 Next Friday: Your Turn

Crowd-sourced content — AI adoption in practice

Anything goes: technical, non-technical, a demo, tips & tricks, a workflow, a war story...

- **1–10 minutes** per slot
- Live demo or pre-recorded video — both welcome
- If oversubscribed, we'll add a second session

Drop me a message on Teams and I'll schedule you in.

Thank You

Context is everything.

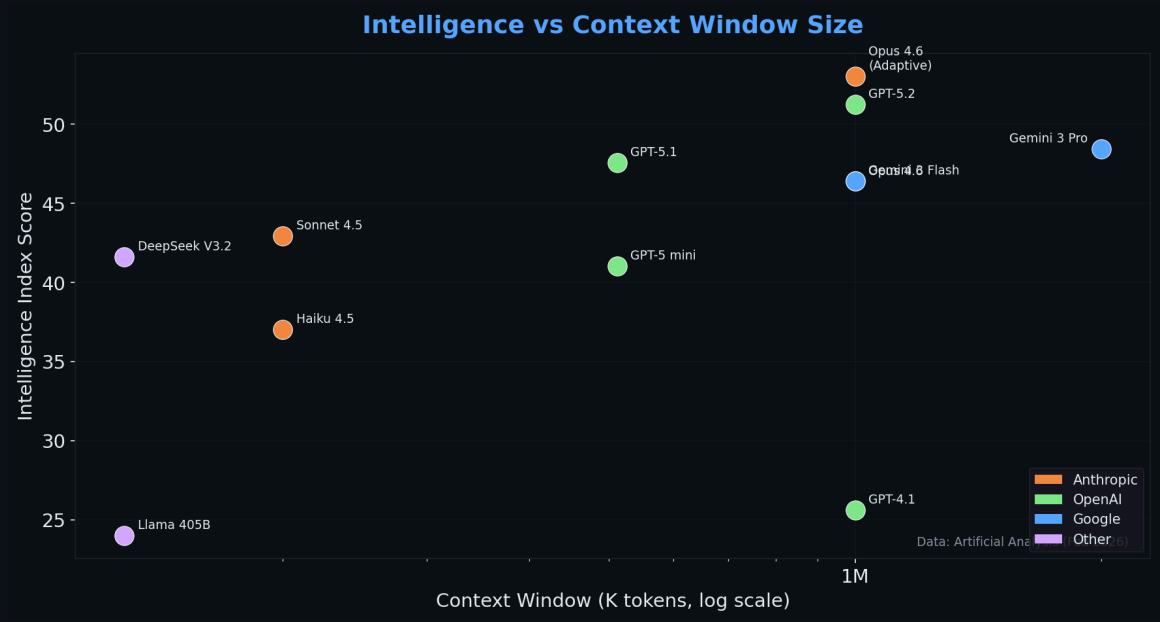
All slides, outline, and references:

`github.com/crmitchelmore/context-presentation`

Key Graphs & Data Sources

Bookmark these — live, interactive data:

Resource	What it shows
epoch.ai	Context window growth (30×/yr)
trychroma.com	Context rot vs input length
artificialanalysis.ai	Quality, speed, price, context
swebench.com	Coding agent leaderboard
attentionviz.com	Attention visualisation
openai.com/tokenizer	Live tokeniser



References & Further Reading

Essential

1. Fowler — "Context Engineering for Coding Agents"
2. Anthropic — "Effective Context Engineering"
3. Willison — "Context Engineering"
4. Anthropic — "Advanced Tool Use"

Fundamentals

5. Raschka — *LLMs-from-scratch*
6. Karpathy — *Neural Networks: Zero to Hero*
7. Karpathy — nanoGPT

Show & Tell

8. system_prompts_leaks
9. anthropics/skills
10. Chroma — Context Rot

GitHub Copilot

11. Custom Instructions
12. Custom Agents

Broader

13. zakirullin/cognitive-load
14. HumanLayer — 12 Factor Agents
15. Epoch AI — Context Windows