How LLMs Work

Context Is Everything

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.

What We'll Cover

1. **First Principles** — tokens, attention, the context window
2. **Context Is Everything** — 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 Simpler Than You Think

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

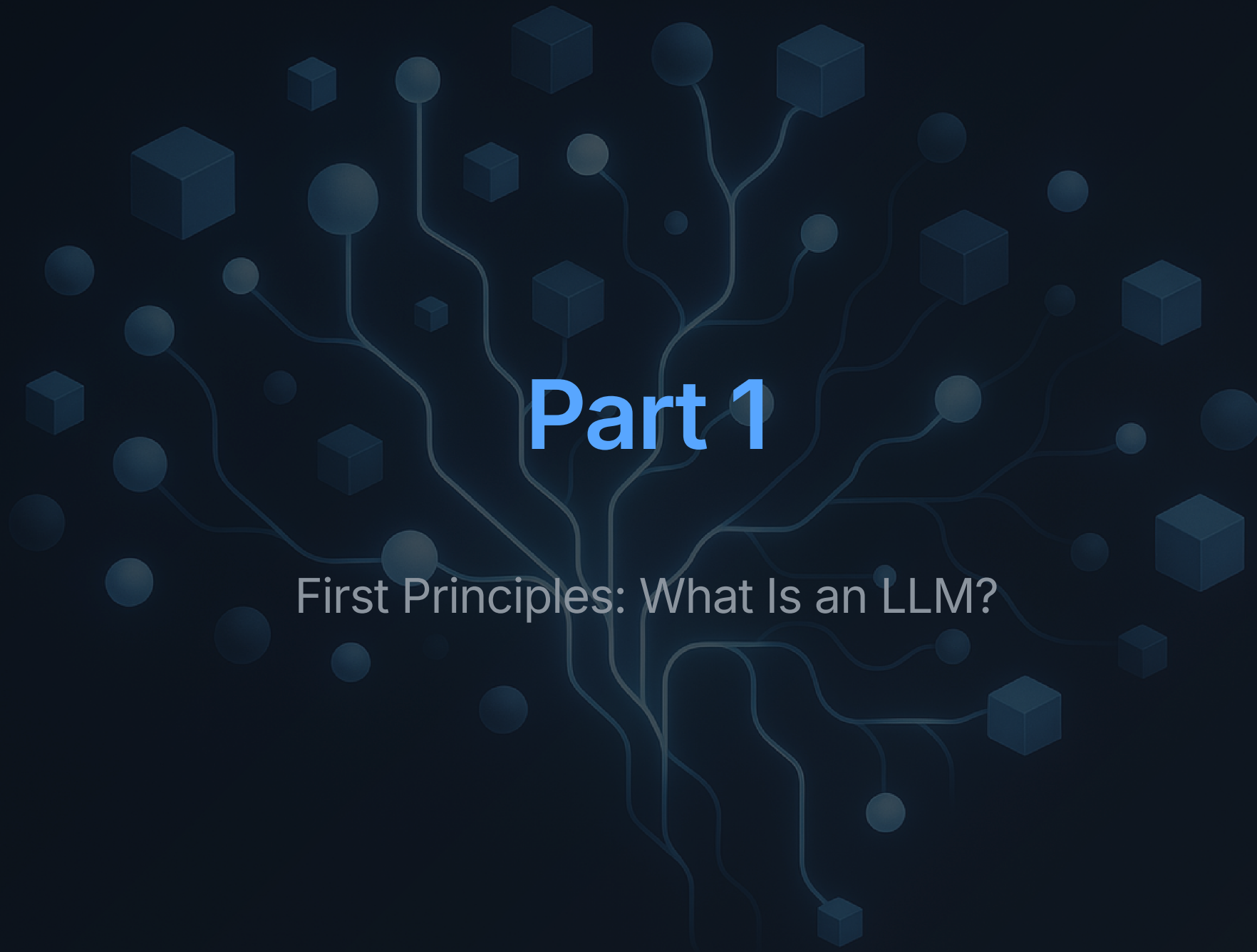
```
# true      = [0, 0, 1, 0]    ← correct answer
# predicted = [0.1, 0.2, 0.6, 0.1] ← model's guess
loss = -(true * log(predicted)).sum()
#          ^ element-wise multiply (×)
```

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

The Jargon Decoder

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

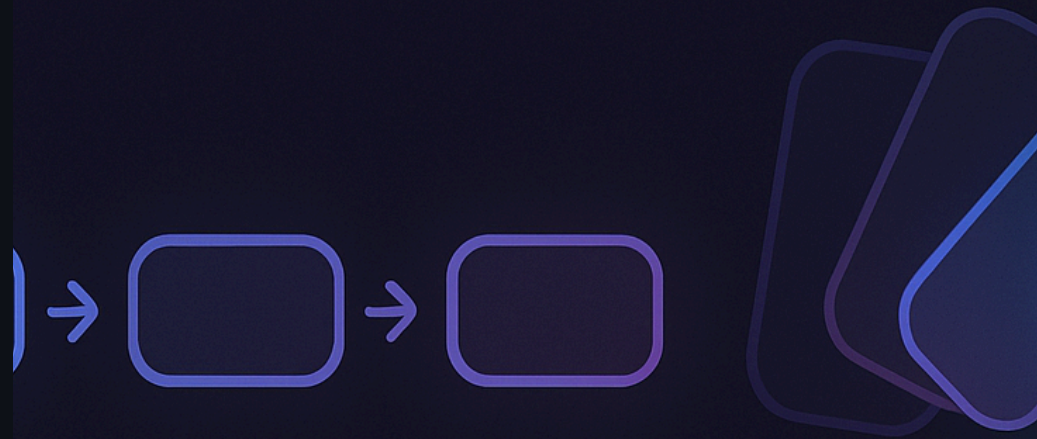
First Principles: What Is an LLM?

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**.

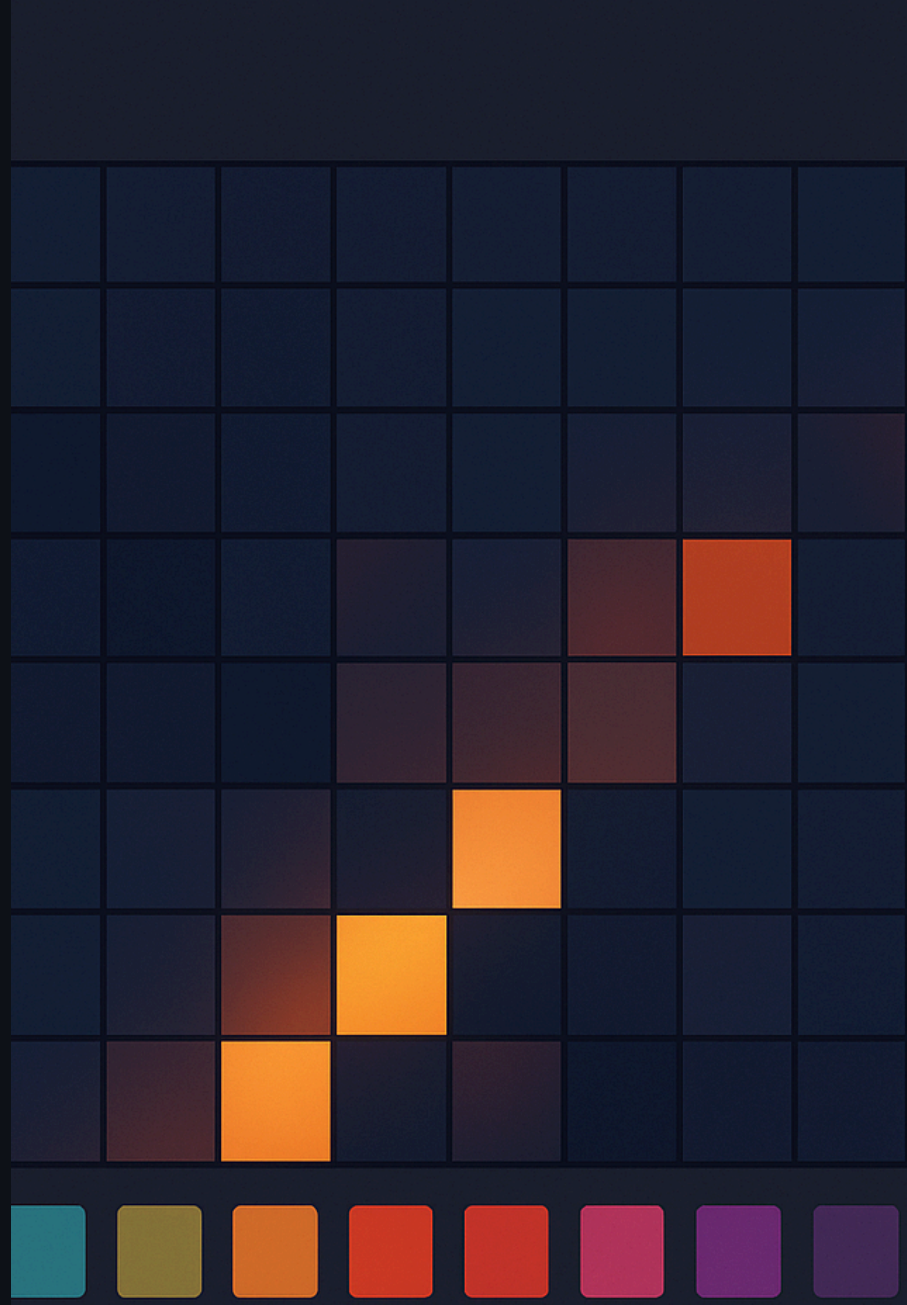
Tokens:	128	1K	8K	128K
Pairs:	16K	1M	67M	16B

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.**

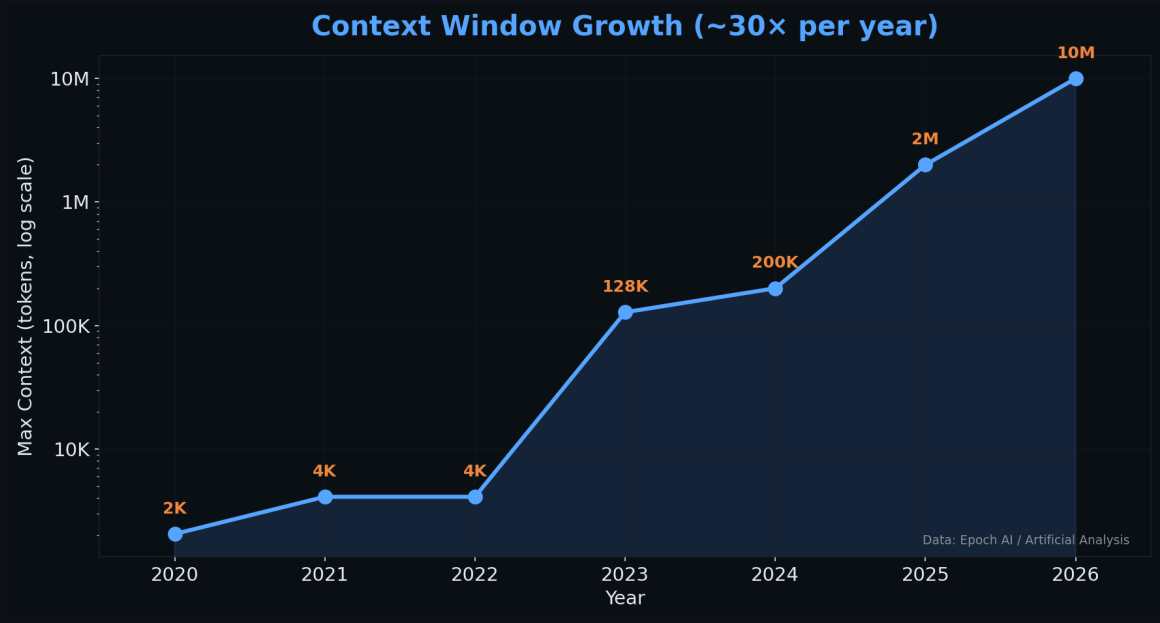
Context Windows Have Grown Fast

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:

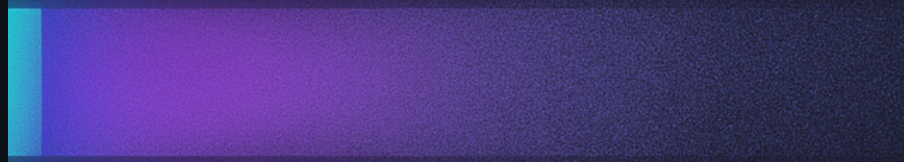
Key findings

- 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*

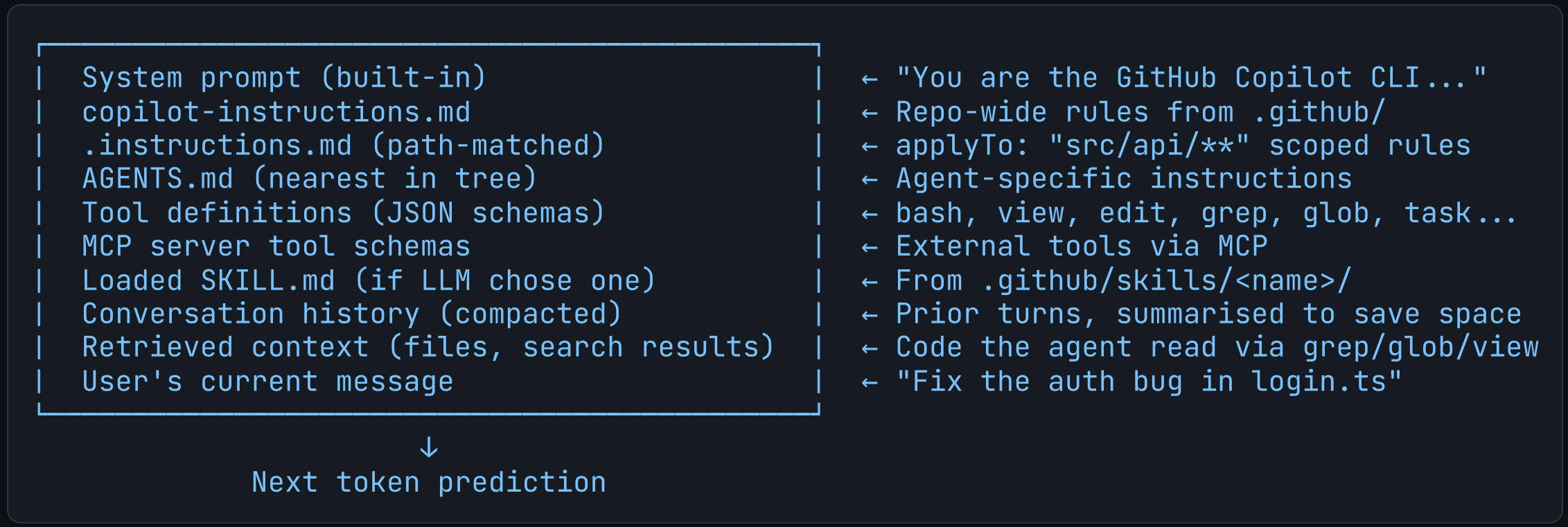


Part 2

Context Is Everything

Anatomy of a Single LLM Call

What goes into one Copilot CLI turn:



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

💡 What's Really In There?

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 message**

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

"Features" Are Just Context Injection

Feature	What it actually does	Token cost
System prompt	Text prepended to every conversation	Always loaded
copilot-instructions.md	Repo-wide rules, auto-loaded	Always or conditional
Tools (function calling)	JSON schemas describing actions	~500–2,000 per tool
MCP servers	Tool schemas from external process	Additive — easily 50k+
RAG	Retrieves text chunks, injects them	Per-query
Skills	Markdown lazy-loaded when relevant	On demand
Sub-agents / custom agents	Fresh window + tailored prompt	Separate window
Conversation history	Prior turns, often summarised	Grows per turn
Few-shot examples	Example pairs pasted into context	Fixed cost
Reasoning / thinking	Chain-of-thought tokens generated & read	Model-generated

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**

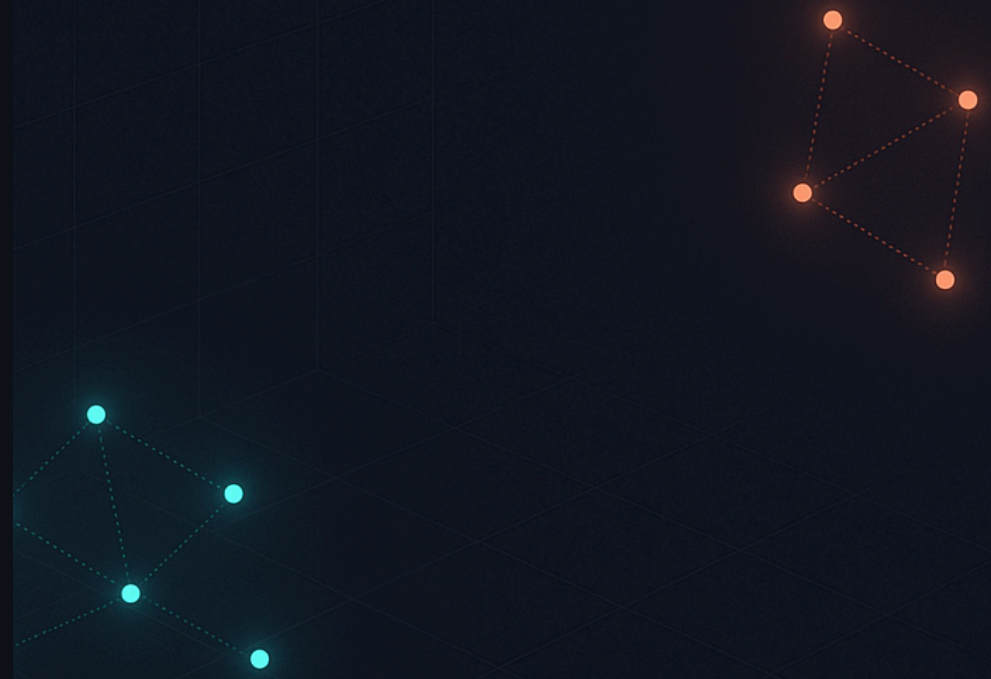
Ref: Anthropic — "Advanced Tool Use" (2025)

RAG: Pre-Load Relevant Context

How it works: text → numeric vectors (embeddings) where similar meaning = close together → search by similarity → inject matches into context

Fast, pre-indexed — but can be stale.

Copilot CLI approach: Just-in-time retrieval via `grep`, `glob`, `view` — always current, contextually relevant.



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)

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

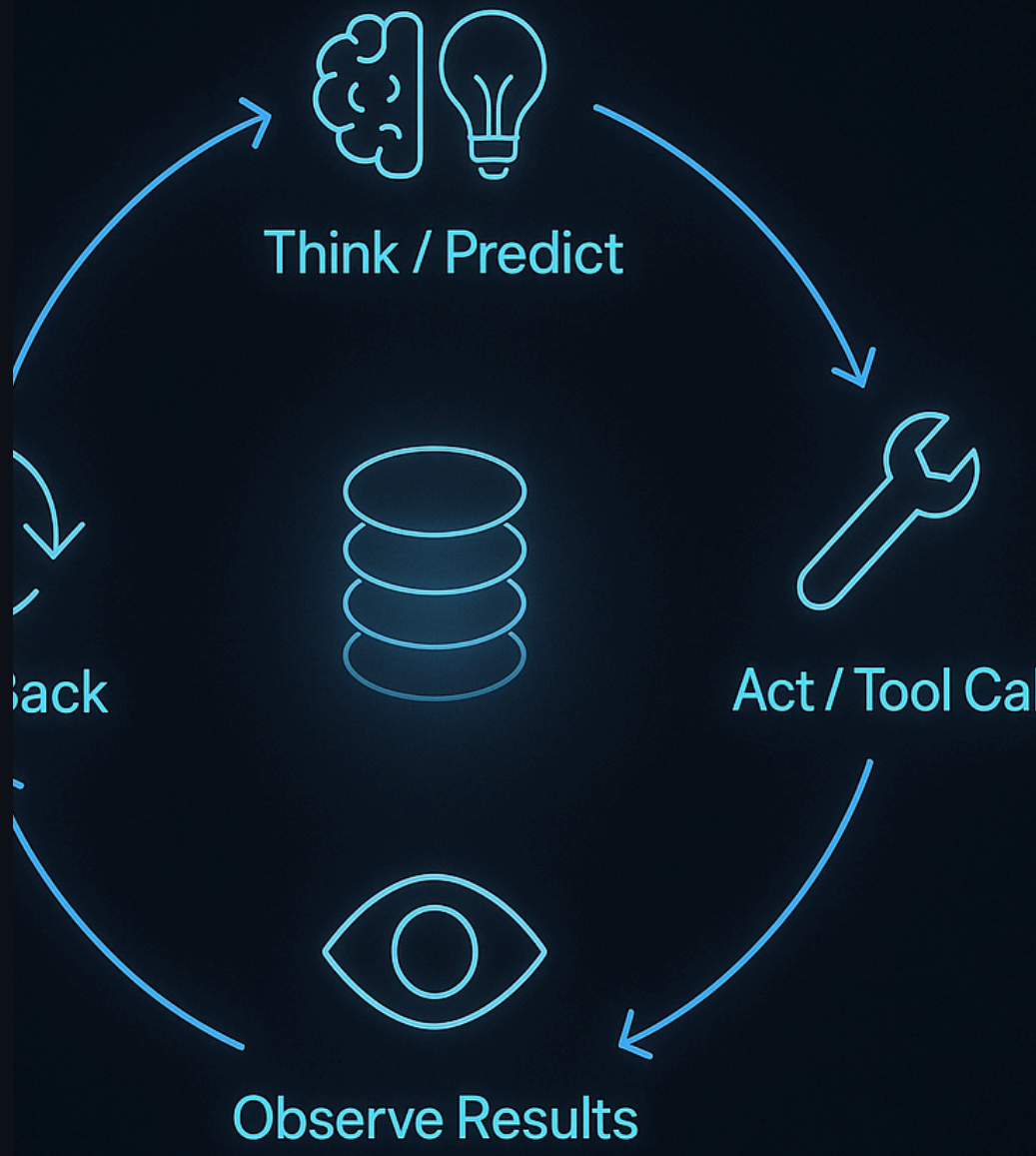
The Agent Loop

> *"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"



How Copilot Fills the Context Window

What gets injected + **who decides** to load it:

Feature	What it injects	Trigger
`copilot-instructions.md`	Repo-wide rules & conventions	**Always** — every request
`.instructions.md`	Path-scoped guidance (`applyTo: "*.ts"`)	**Path-matched**
`AGENTS.md`	Agent-level instructions (nearest wins)	**Always** for agents
Agent Skills (`SKILL.md`)	Specialised workflows & knowledge	**LLM-decided**
Custom Agents (`.agent.md`)	Curated persona, tools, model	**Human-triggered**
Hooks	Lifecycle scripts (pre/post actions)	**Event-driven**
`grep` / `glob` / `view`	Codebase content on demand	**LLM-decided** (JIT)

Every row: markdown or tool output → tokens → context

Ref: Fowler — "Context Engineering" · GitHub Docs — Custom Instructions · GitHub Docs — Custom 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 specialized persona + curated tools
- Might use 50k+ tokens exploring
- Returns concise summary to parent agent

Managing Context: Summary

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

1. **Compaction** — summarize 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

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.

Why This Is Hard

> *"As long as LLMs are involved, we can never be certain of anything. We still need to think in probabilities and choose the right level of human oversight for the job."*
> — **Martin Fowler**

Context engineering increases **probability** of good output — doesn't guarantee it.

- Rules say "always do X" → model might not follow
- Skills describe when to load → model might choose not to
- Hard code for humans = hard for LLMs → **cognitive load matters**

Calibrate oversight to stakes:

- 🟢 Low — read public data → let it run
- 🟡 Medium — read private data → log and review
- 🔴 High — write access, send messages → **require human approval**

Part 4

Putting It Together

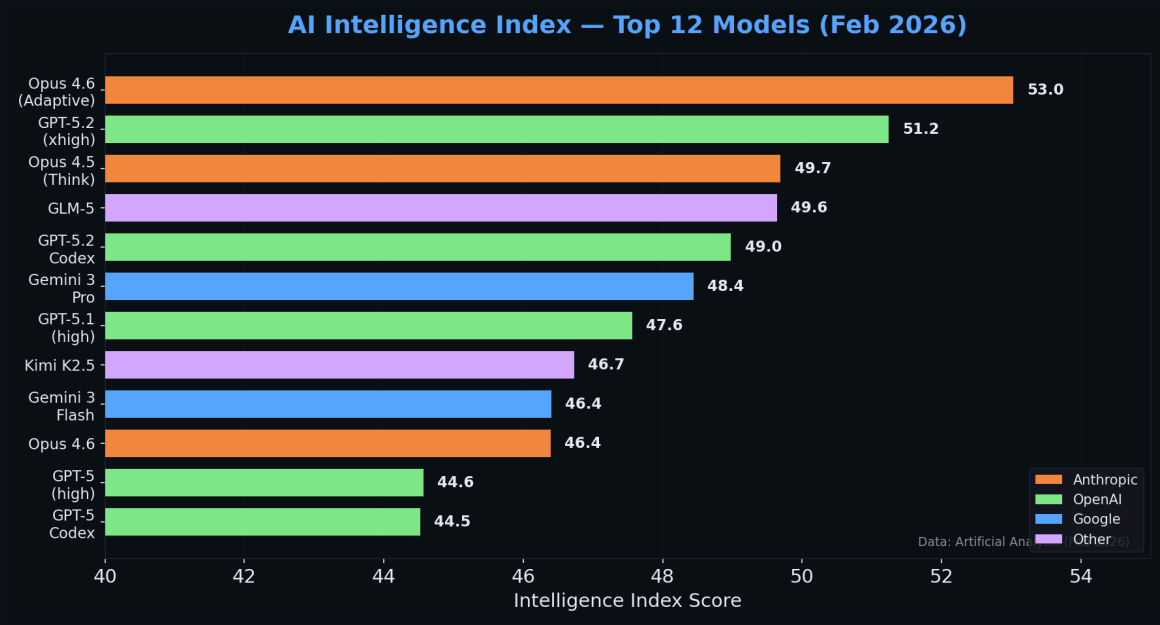
Coding Agents: Where This Lands

SWE-bench Full: % of real issues resolved

Model	Score
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

swebench.com · artificialanalysis.ai



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

The Mental Model

1. An LLM **predicts the next token** based on its context window
2. The context window is **all it has** — no hidden memory
3. Every feature is **context injection** — prompts, tools, MCP, RAG, skills, agents
4. Context engineering is **the new core skill** — curating what the model sees
5. This applies whether you're **building or using** AI

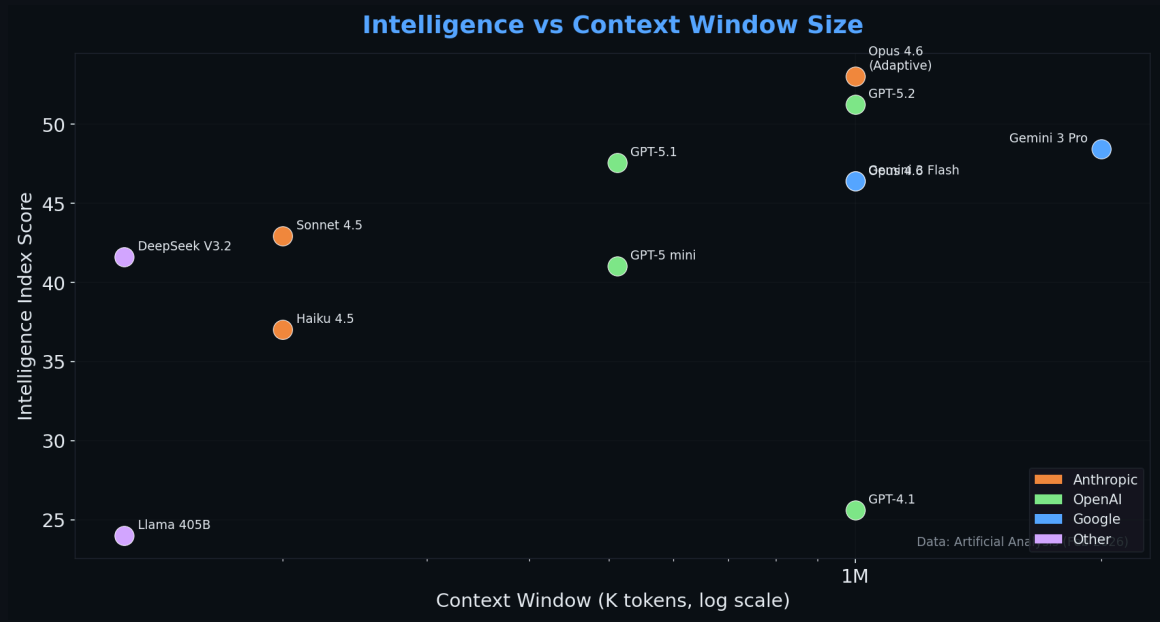
What We Covered

1. ✅ LLMs **guess the next token** — that's the whole mechanism
2. ✅ The **context window** is all they see — no hidden memory
3. ✅ Every feature is just **putting text into that window**
4. ✅ **Context engineering** — curating what goes in — is the core skill
5. ✅ This transfers across tools, models, and use cases

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

Thank You

Context is everything.

All slides, outline, and references:

`github.com/crmitchelmore/context-presentation`