

 TsunamiIA · Sparks

A ONE-DAY WORKSHOP

Introduction to Large Language Models

Building and shipping LLM systems in production

Amadeus · Sparks Dimitri Foucault · TsunamiIA

How we will work today

A few ground rules to get the most out of the day.

Camera on

It is interactive · let us see each other.

Ask anytime

Interrupt me · no question is too basic.

Try, don't just watch

The labs are hands-on · follow along on your machine.

Share real use cases

Your context makes the day useful.

Confidentiality

What is shared here stays here · no company data in the tools.

Notifications off

Stay present and engaged.

Round the table

Quick intros, so I can tailor the day to you.

Your name & team

Who you are, what you work on.

What you build

Your stack, your role.

Your LLM experience

On a scale of 1 to 5.

Your goal today

One thing you want to leave with.



A quick word about me



Late 2022, GPT was a real shock: my security-consulting work was about to change, so I chose to ride the wave. Since then I help companies **integrate generative AI** · over 1,000 people trained and 15 audits last year.

— WHAT I BRING

- ◆ **Generative-AI integration** · from first use case to production
- ◆ **Project & change management** · the manager side, not only hands-on
- ◆ **Security background** · former CISO
- ◆ **TsunamiA** · a weekly AI podcast for 3 years · constant dialogue with the field



TsunamiA · the AI podcast

link.tsunami.a.fr



Objectives for the day

By the end of the day, you should be able to:

1 Understand what an LLM is and how it works, without the heavy maths

2 Recognise real integration patterns: assistants, RAG, agents

3 Grasp the economics: tokens, context and cost across models

4 Know the main risks of LLMs · and how to reduce them

5 Leave with a practical map to scope your own LLM project

Agenda for the day

Six modules, three hands-on labs. A lot of ground to cover, and you keep the deck.

- 1** Fundamentals: how LLMs actually work 60 min
- 2** Prompt engineering & structured output · Lab 1 75 min
- 3** RAG & fine-tuning: choosing the right approach · Lab 2 70 min
- 4** Deployment, cost & token economics 60 min
- 5** LLM security essentials · OWASP Top 10 · Lab 3 60 min
- 6** Trends, evaluation & perspectives 35 min



Same prompt, different answers 10 min

Let's feel the probabilistic nature of an LLM before we explain it.

— STANDARD TRACK

Everyone sends the **same prompt** to their assistant (Copilot). Read a few answers out loud and compare. Suggested prompt: *"In 3 bullet points, what will be the biggest change in software engineering in 2027?"*

— ADVANCED CHALLENGE

Send the same prompt **with vs without web access**. Observe: different outputs, sensitivity to phrasing, and the effect of grounding on factuality.

Environment: GitHub Copilot Chat (any model). No company data.

01

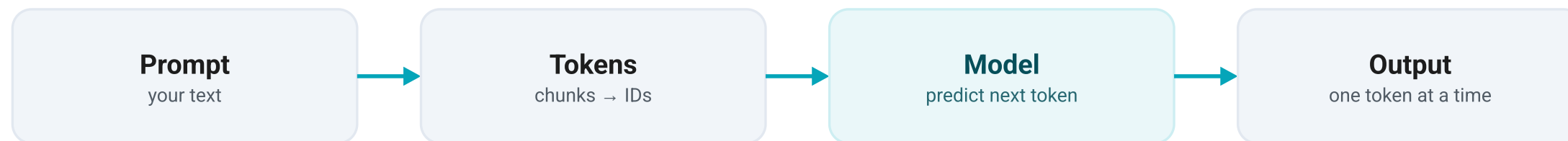
LLM landscape and how they work

A shared vocabulary, without the heavy
maths.



What is an LLM?

A model trained to predict the next token, over and over.



It does not "know" or "think" · it produces the most statistically plausible continuation.

Autoregressive generation

The model outputs a probability for every possible next token, picks one, appends it, and repeats.

“The cat sat on the ___”



Temperature decides how sharply we pick. Generation stops at a stop token or max length.

Token: definition

◆ A token is a piece of text the model actually processes · not a letter, not always a word, often a sub-word.

◆ The model never sees letters, only token IDs (numbers).

◆ Tokens are the unit of context windows, latency and billing.

Counting tokens in practice

Measured with tiktoken · a common word is cheap, random strings are expensive.

Common word

Bonjour = 1 token

Random / rare string

UD **H** **F0** **Y** **382** **D** = 6 tokens

Rule of thumb (English): 1 token \approx 4 characters \approx 0.75 word. Random strings cost many tokens \rightarrow more expensive.

Is "1 token \approx 4 chars" always true?

No · it's an English average for older tokenizers.

◆ It varies by language, casing, leading spaces, digits...

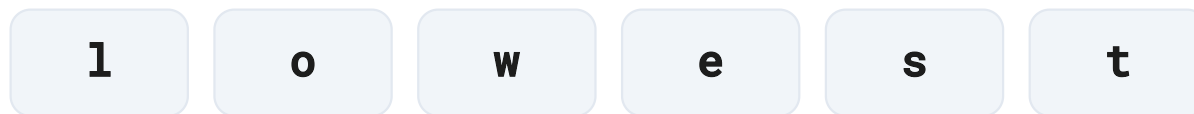
◆ ...and above all the model's tokenizer. The **vocabulary** is the fixed set of tokens it knows · a bigger vocabulary = fewer tokens for the same text.

◆ Big vocabularies: GPT-4o o200k (200k), Llama 3 (128k), Gemini (256k).

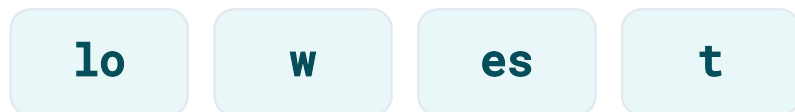
What is a tokenizer? (BPE)

Byte-Pair Encoding: merge the most frequent pair, repeatedly, until the target vocabulary size.

1 · characters



2 · merge frequent pairs



3 · subword tokens

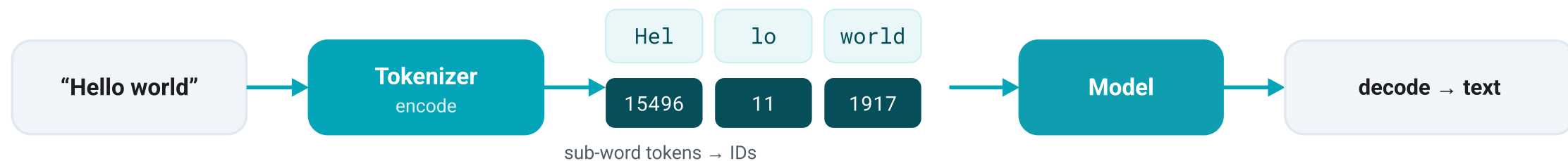


Byte-Pair Encoding: repeatedly merge the most frequent pair into a new symbol, until the target vocabulary size is reached.

Frequent words → 1 token. **Rare words** → split.

Tokenizer: text in, text out

Encode to token IDs, run the model, decode back to text.



The model only ever sees integers, never letters.

Principle redrawn from Hands-On Large Language Models (Alammar & Grootendorst).

Why the tokenizer impacts price, context and latency

Tokens are the unit of billing, of context, and of speed · so the tokenizer touches all three at once.

€ Price

APIs bill **per token**, input and output. A 50-page PDF is already about 30k input tokens, on every call.

📄 Context

The context window is a **token budget** (128k, 200k, 1M...). Prompt, history, RAG and output all count against it.

🕒 Latency

More tokens = more compute. Output tokens dominate (made one by one); a long prompt mainly adds **time-to-first-token**.

- ◆ Fewer tokens for the same meaning = **cheaper, faster, more room in context**.
- ◆ The **same text** costs a different number of tokens on different models · and the tokenizer also shapes how well the model handles **rare words and code**.

— TOKENIZERS IN THE WILD

“cl100k”, “o200k” · what those names mean

A tokenizer has a name and a **vocabulary size** · a bigger, better vocabulary packs the same text into fewer tokens.

- ◆ **cl100k_base** · the tokenizer of GPT-3.5 / GPT-4 · about 100k entries in its vocabulary.
- ◆ **o200k_base** · GPT-4o and the o-series · about 200k entries · fewer tokens for the same text, especially non-English and code.
- ◆ Newer frontier models keep upgrading their tokenizer · swapping models can **silently change your token counts** (and your bill).
- ◆ Fewer tokens = lower cost, more effective context, and a shorter **time-to-first-token** · less to read before it starts answering.

*Real-life: with a long prompt you wait through ****prefill**** (the model reading your input) before the first word appears · that is the time-to-first-token.*

The language "token tax" (FR vs EN)

Non-English text costs more tokens · but the gap depends heavily on the tokenizer.

tokens for the same 10,000 characters

GPT-4 (cl100k)



GPT-4o/5 (o200k)



The FR penalty melts with newer, larger-vocabulary tokenizers. The big tax (2.5–15x) hits non-Latin scripts.

*The tax is relative to the **tokenizer's corpus**, not the language. A Western tokenizer taxes Chinese (GPT-4o ≈ 1.3 char/token vs ≈ 4.7 for English), but a **Chinese-native model** (Qwen, DeepSeek) tokenizes Chinese efficiently · sometimes cheaper than English.*

Embeddings (1/3) · what they are

An embedding turns a text (or image) into a vector of numbers that captures its meaning.

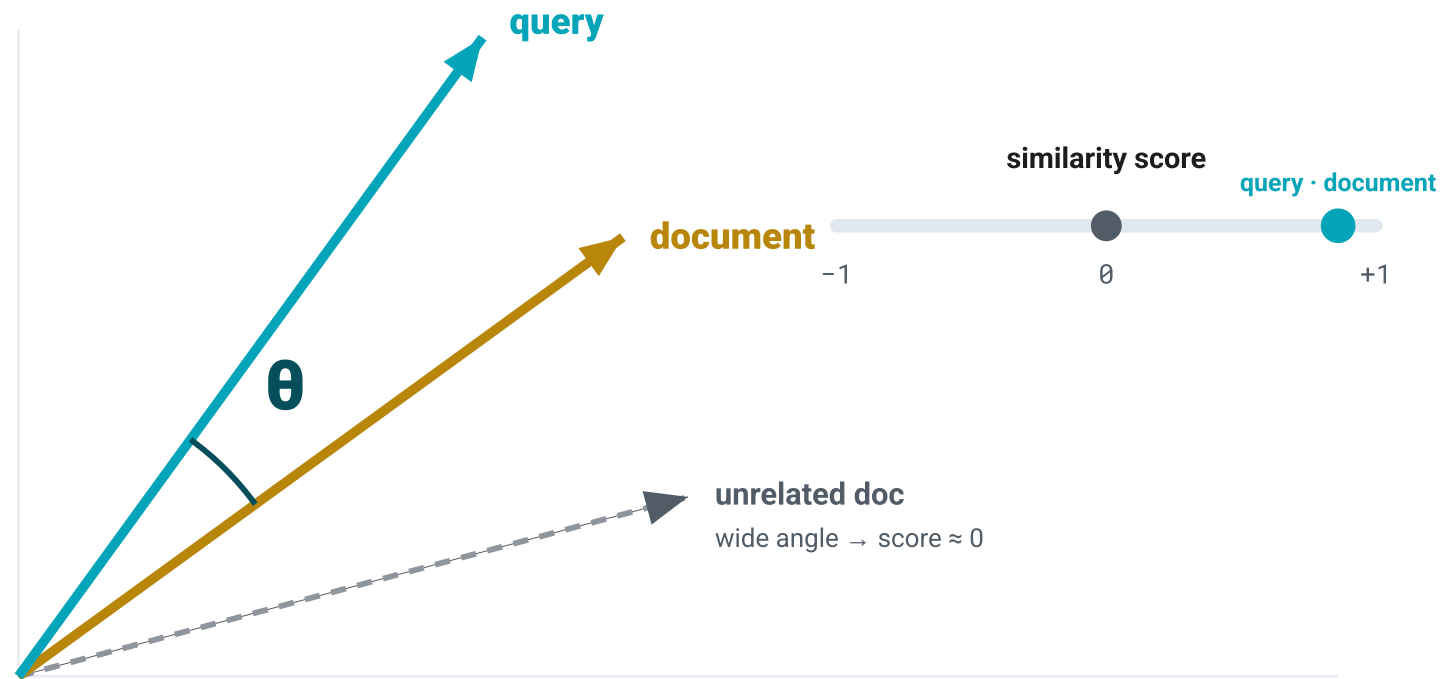


Similar meaning → nearby vectors. Search = return the closest points to the query.

Different from a generative LLM: an embedding model only produces vectors. Typical sizes: 768, 1024, 1536, 3072 dimensions.

Embeddings (2/3) · similarity & vector space

Cosine similarity measures the angle between two vectors. Search = find the vectors closest to my query.

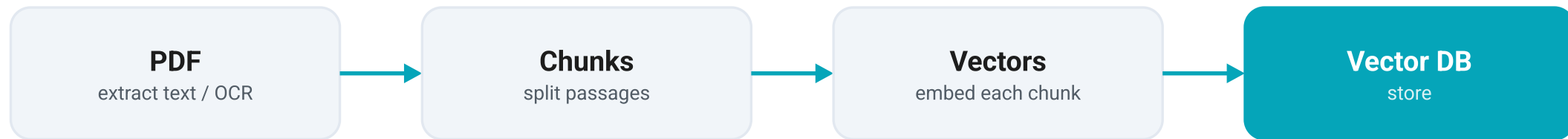


cosine similarity = $\cos \theta$ · from -1 (opposite) to +1 (same meaning); search returns the highest scores

*This powers **semantic search** and **RAG**.*

Embeddings (3/3) · embedding documents (PDF etc.)

“Embedding a PDF” = converting it into vectors so you can retrieve it by meaning.



At query time: embed the question → retrieve the closest chunks. This is the first half of a RAG.

*Two cautions: **chunks are a RAG thing** · if the doc fits in context, skip them and just paste the text (next module). And **most RAG failures start at PDF parsing** (shifted tables, page numbers mid-sentence), not at the embedding · garbage in, garbage out.*

— MULTIMODAL EMBEDDINGS

ColPali · retrieving PDFs without OCR

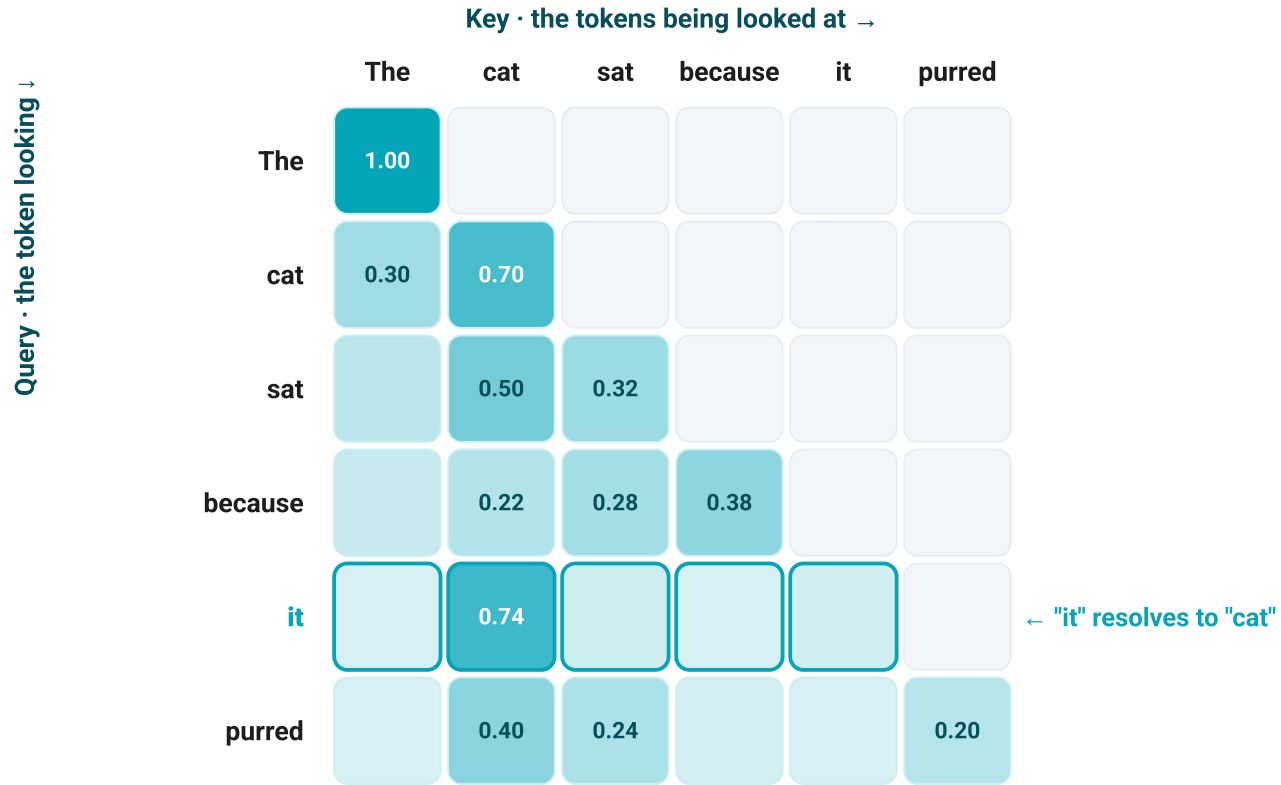
Instead of reading the text, the model **looks at each page as an image** and embeds it directly.

- ◆ **Classic RAG:** PDF → OCR → text → embed. Brittle on tables, columns, scans and figures · the layout is lost.
- ◆ **ColPali:** render each page to an image → a vision-language model embeds the picture into the same space as the query text.
- ◆ No OCR step · the model sees the page (tables, charts, handwriting, logos) the way you do, nothing thrown away.
- ◆ At query time, your text question and the page images share **one vector space** · retrieval = the nearest pages.

For travel-tech: contracts, fare rules, scanned docs and screenshots become searchable as-is.

Attention (1/2) · the intuition

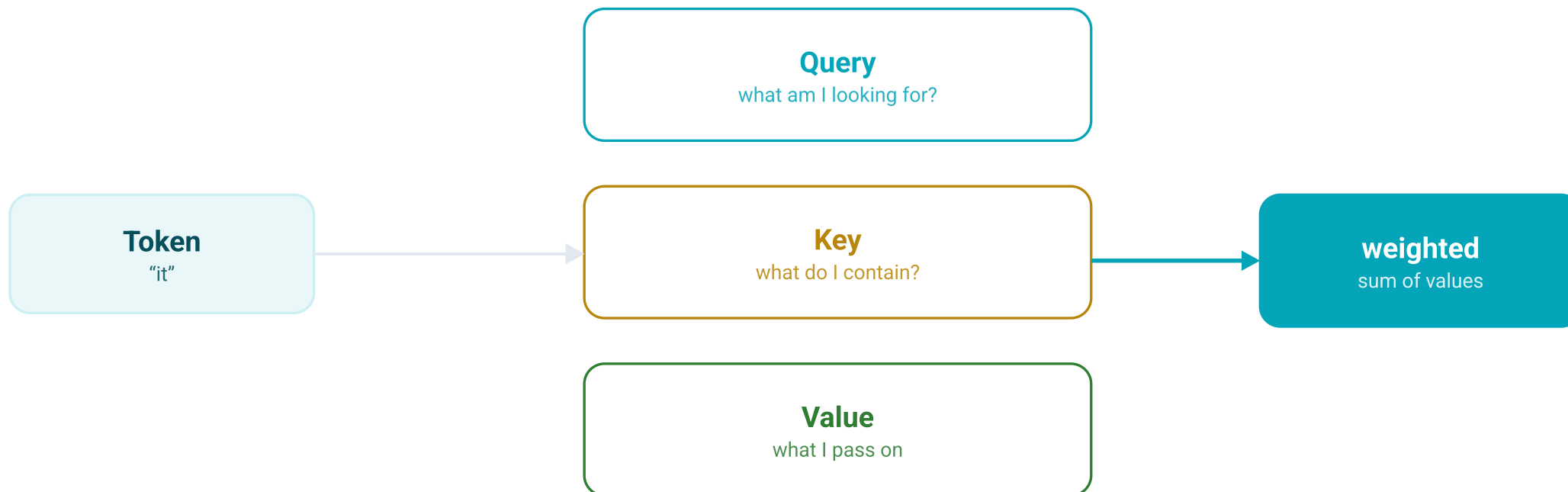
Each token “looks at” all the others and decides which ones matter.



Each **row** is a token asking “what should I look at?”. Darker cell = more attention. The query “**it**” points back to “**cat**” · that is coreference, learned, not coded.

Attention (2/2) · Q/K/V and why it changed everything

Each token emits a Query, every token offers a Key and a Value.



Match query → keys → weights → weighted sum of values. The Transformer (2017) made this parallel and scalable.

*The **context window** is how many tokens can attend to each other at once.*

— NO MATHS, JUST THE IDEA

The KV cache, in plain words

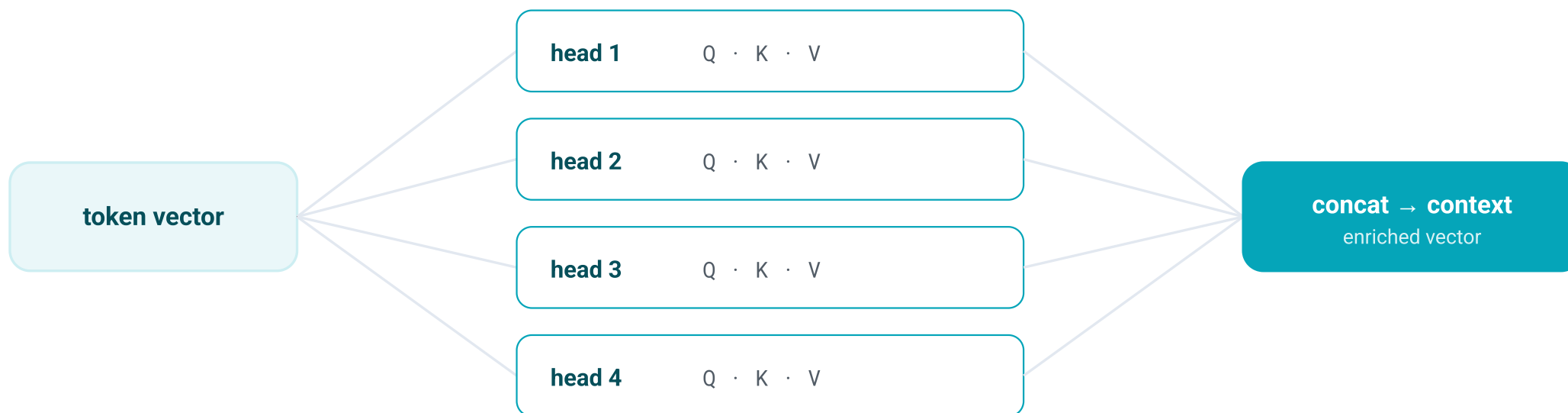
Writing one token at a time, the model would re-read everything it already wrote. The KV cache stops that waste.

- ◆ Each new token must look back at all the **Keys and Values** from attention. Redoing that every step is a huge waste.
- ◆ So the model keeps a **notepad** of what it already computed and just adds the new token · that notepad is the **KV cache**.
- ◆ It is what makes generation fast, but the notepad **grows with the length of the chat**, so long contexts use more memory.
- ◆ **Prompt caching** = reusing this notepad for a shared start (system prompt, big doc) · about **-90% cost**. Same idea, sold as a feature.

Analogy: writing a long email, you don't re-read the whole thread before each word · you keep it in your head and add the next one.

Multi-head attention

Several heads run in parallel, each learns a different relation.

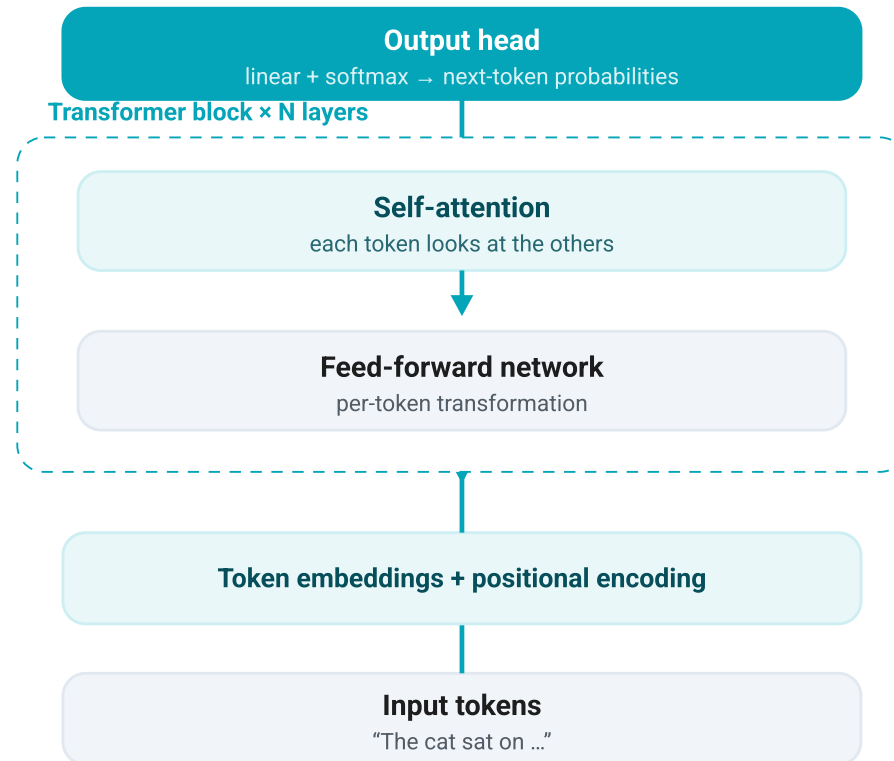


Several attention "heads" run in parallel, each learns a different kind of relation.

Principle redrawn from Hands-On Large Language Models (Alammar & Grootendorst).

The Transformer, in one picture

Embeddings in, N stacked blocks, a next-token distribution out.



One forward pass =
one predicted token.

Stacking blocks lets the model
build richer representations.

*Context window = how many tokens
attend to each other at once.*

Architecture principle redrawn from Hands-On Large Language Models (Alammar & Grootendorst).

Encoder, decoder, encoder-decoder

Same Transformer brick · the difference is whether a token may look at the future.

Encoder-only (BERT)

Bidirectional (sees left and right), trained by masking words. Great for **embeddings and understanding**, not generation.

Decoder-only (GPT)

Causal (sees only the past), trained on next-token prediction. **Today's chat LLMs**, autoregressive.

Encoder-decoder (T5)

Encoder understands the input, decoder writes the output. Translation, summarization.

What is a parameter?

◆ A parameter is a **weight** · a number the model learned during training.

◆ Modern LLMs have **billions** of them. “7B” = 7 billion parameters.

◆ More parameters = **more capacity**, but also **more memory and compute**.

Model naming: total vs active parameters

A model's name ("7B", "671B") is its **total** size. What it actually runs per token can be far smaller.

TOTAL PARAMETERS

- ◆ All the weights the model **owns** · stored on disk, loaded into memory.
- ◆ Sets the **memory footprint**: the VRAM just to load the model.
- ◆ This is the number in the name: 7B, 70B, 671B.
- ◆ Dense models: **total = active** · every weight runs for every token.

DeepSeek-V3 stores **671B** parameters in memory.

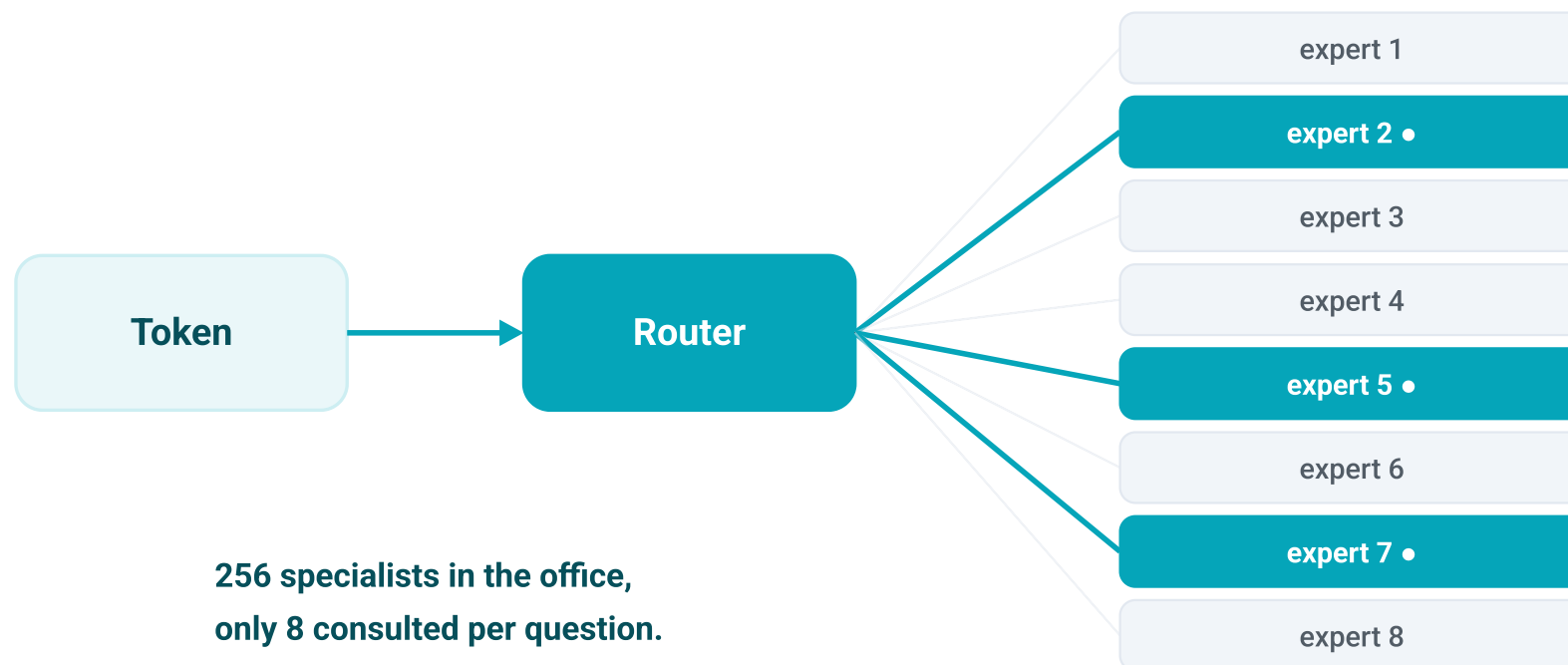
ACTIVE PARAMETERS / TOKEN

- ◆ The weights **actually used** to produce one token.
- ◆ Sets **compute, cost and latency** · what you really pay to run.
- ◆ Mixture-of-Experts: a **router** activates only a few experts per token.
- ◆ So **active** \ll **total** · big in memory, much cheaper to run.

DeepSeek-V3 runs only **37B** per token (about 1 in 18).

Mixture of Experts (MoE) · what it is

A small router picks only a few experts per token, instead of running the whole model.



256 specialists in the office,
only 8 consulted per question.

*Pay **compute** at the price of **active** params, keep the **knowledge** of **total** params. Catch:
you still need enough memory to hold ALL experts.*

MoE · real examples

Total vs active parameters.

MODEL	TOTAL PARAMS	ACTIVE PARAMS	SPARSITY
Mixtral 8x7B	46.7B	12.9B	2 of 8 experts
Qwen3-235B-A22B	235B	22B	8 of 128 experts
DeepSeek-V3	671B	37B	8+1 of 256 experts
Kimi K2	1T	32B	8 of 384 experts

"DeepSeek-V3 = the quality of a near-700B at the compute cost of a ~37B." (DeepSeek blocked at Amadeus, kept as a teaching example.)

Parameters → memory footprint (the rule)

Memory for weights \approx number of parameters \times bytes per parameter.

weights memory \approx parameters \times bytes per parameter

bytes per parameter, by precision

FP32

4 B

rarely used now

FP16 / BF16

2 B

standard · big LLMs

INT8

1 B

local / on-prem

INT4

0.5 B

consumer GPUs

Plus the KV-cache (grows with context) and overhead → rule of thumb VRAM \approx weights \times 1.2

Memory examples + CPU vs GPU

A 7–8B model (weights only):

FP16 → ≈14 GB

Needs a **GPU** (16 GB+).

8-bit → ≈7 GB

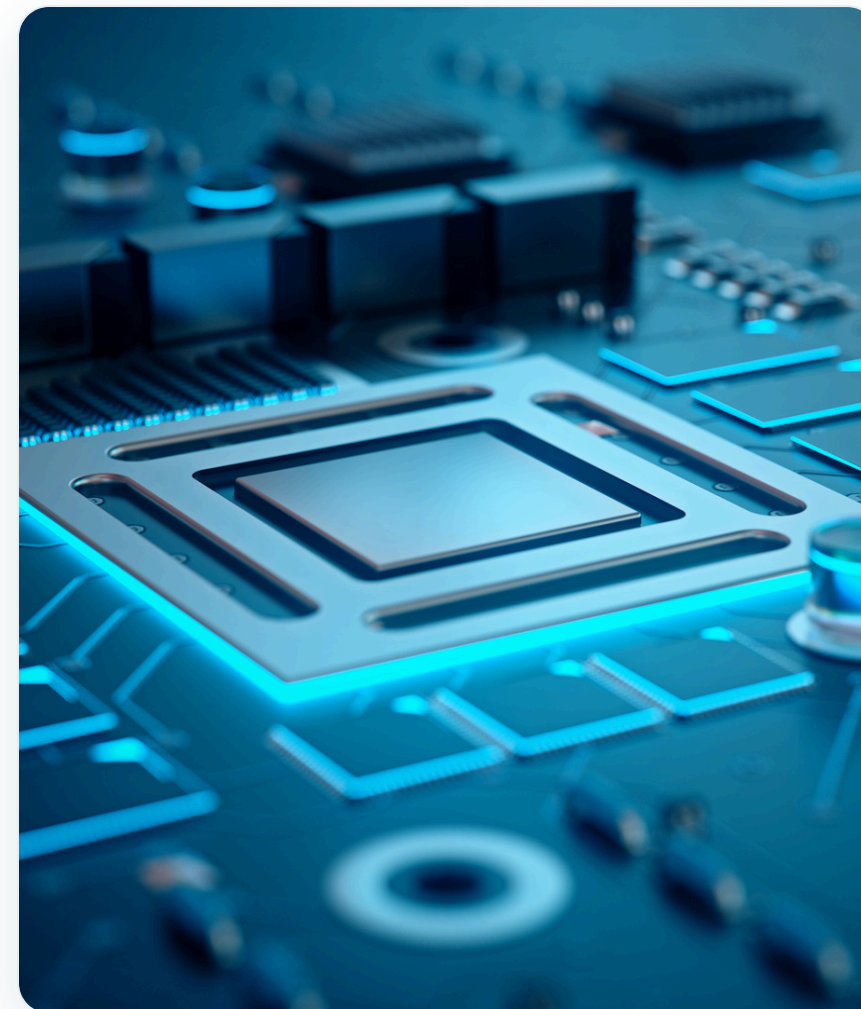
Borderline laptop / small GPU.

4-bit → ≈4–5 GB

Fits a **modern laptop** (CPU / Apple Silicon),
no GPU.

70B in FP16 → ≈140 GB

Multiple GPUs.



Quantization · what & how

Store each weight with fewer bits (FP16 → INT8 → INT4). Lose a little precision, gain a lot of memory and speed.

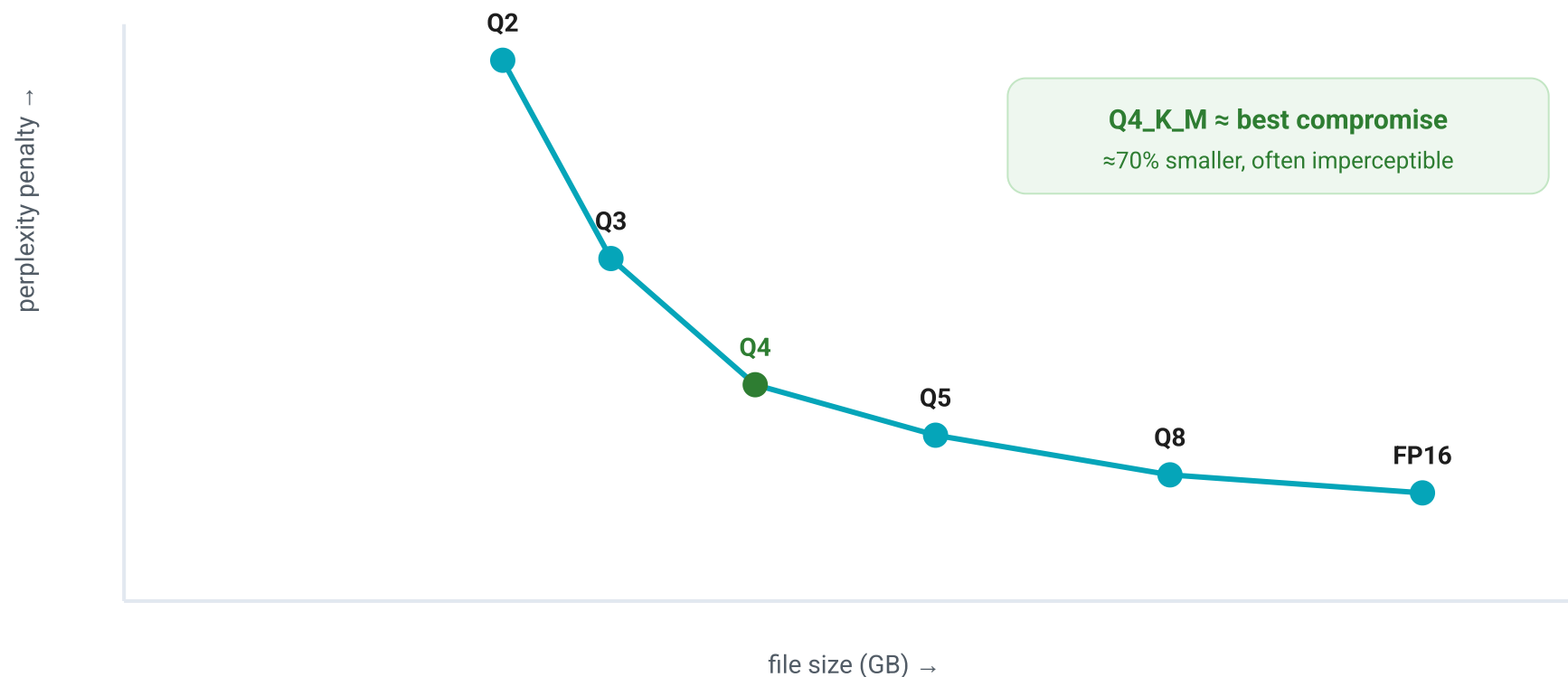
◆ **GGUF** = the file format for llama.cpp / Ollama / LM Studio, runs on CPU.

◆ Variants **Q4_K_M**, **Q5_K_M**, **Q8_0** · the number \approx bits per weight.

◆ **GPTQ** / **AWQ** are GPU-oriented methods.

Quantization · quality vs size

Perplexity = how “surprised” the model is (lower = better). Modern 4-bit adds only $\approx +0.1$ to $+0.3$.



*Golden rule: **the bigger the model, the less quantization hurts** (a 70B Q4 \approx FP16; a 3B Q4 starts to hallucinate).*

1-bit models / BitNet

An extreme idea: train the model so each weight is almost a single bit · just -1 , 0 or $+1$.
Far less memory, much simpler maths.

FP16 weights

$a \times w$

multiplication (costly)

continuous values

1-bit weights $\in \{-1, 0, +1\}$

$a + / - a$

add / subtract (cheap, CPU-friendly)

2.4–6.2× faster · 55–82% less energy

BitNet b1.58: weights natively trained low-precision, not compressed after the fact.

Still early: it works at small sizes, but there is no top-tier 1-bit model yet.

Model landscape mid-2026 · proprietary

PROVIDER	FLAGSHIP	RELEASED
OpenAI	GPT-5.5	Apr 2026
Anthropic	Claude Opus 4.8 · Sonnet 4.6	28 May 2026
Google	Gemini 3.1 Pro · Gemini 3.5 Flash	19 May 2026
xAI	Grok 4.x family	n/a

The frontier moves fast: Opus 4.8 shipped the day before this session.



Model landscape mid-2026 · open-weight

PROVIDER	OPEN-WEIGHT MODELS	LICENSE
Alibaba	Qwen3 (dense + MoE, e.g. 235B / 22B)	Apache 2.0
Meta	Llama 4 Scout / Maverick (Behemoth still training)	Llama license
OpenAI	gpt-oss-120b / gpt-oss-20b	Apache 2.0
Moonshot	Kimi K2.6 (1T / 32B)	Modified MIT
Mistral	Mistral Large 3 (675B / 41B) · Medium 3.5	Apache 2.0
DeepSeek	DeepSeek V4 · blocked at Amadeus	MIT

Open-weight sits at roughly 5% behind the proprietary frontier. Qwen and DeepSeek lead community adoption.



Open weight vs open source + licences

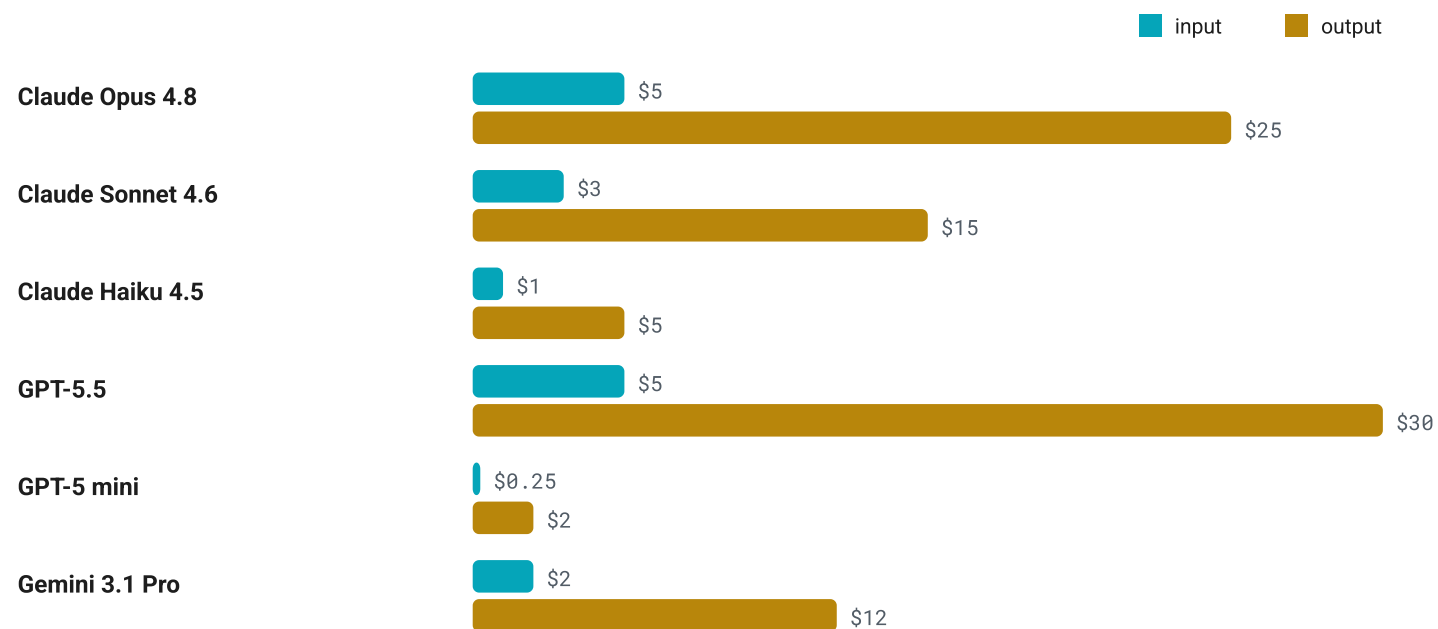
Open weight = you get the weights (run, fine-tune), not the data or recipe. **Open source** (strict) = everything to rebuild it, very rare. So open weight is not open source.

LICENCE	COMMERCIAL USE	THE CATCH	EXAMPLES
MIT	Yes, fully	Most permissive	DeepSeek (blocked at Amadeus)
Apache 2.0	Yes, fully	+ explicit patent grant	gpt-oss, Mistral Large 3, Qwen3
Modified MIT	Yes	Attribution only above huge user thresholds	Kimi K2
Llama Community	Yes, with limits	Not open source; >700M users needs a Meta deal	Llama 4

*Licence ≠ approval: DeepSeek is MIT but **blocked at Amadeus**.*

Model pricing · input vs output

API price per 1M tokens. Output tokens are typically the most expensive.



USD per 1M tokens. Output tokens are typically the most expensive.

*Standard tiers shown · GPT-5.5 is \$5/\$30 here; the deliberative **GPT-5.5 Pro** is far pricier (\$30/\$180). Turning on “thinking” does not raise the per-token price, it just produces more (billed) tokens.*



Strengths (with examples)

 **Summarise / rewrite / translate**


Compress, reformat, localise text.

 **Extract structure from messy text**

Logs, emails, bookings → clean fields.

 **Draft and transform code**

Scaffold, refactor, explain.

 **Answer over your data (grounded)**

RAG on your documents.

- ◆ Amadeus angle: **travel logs are text** → classification, anomaly hints, report drafting.

Weaknesses & limits (with their counter-measures)

Each limit announces a counter-measure seen later · the thread of the day.

THE LIMIT

- ◆ **Hallucination** (confident but wrong)
- ◆ **Knowledge cutoff**
- ◆ **Non-determinism** (same prompt, different output)
- ◆ **Sensitivity to phrasing**
- ◆ **Maths / counting** errors (tokenization)

THE PARADE

- ◆ RAG + evaluation
- ◆ RAG / tools
- ◆ Temperature + tests
- ◆ Prompt engineering
- ◆ Tools / code (e.g. count the r's in “strawberry”)

02

Prompt engineering and structured output

The 2025-2026 shift: from prompt to context engineering.

Has prompting aged?

The skeleton still holds, but its value moved.

- ◆ On modern models, **clever phrasing matters less**; clarity, structure and context selection matter more.
- ◆ The field shifted from “**prompt engineering**” → “**context engineering**” · finding the smallest set of high-signal tokens.
- ◆ Documented fact, not just an impression: sensitivity to **tricks** drops, sensitivity to **ambiguity** stays high.



The classic structure (still a baseline)

Still the baseline · not for magic words, but because it forces you to be clear: who the model is, what it knows, what to do, and in what format.

Role

Who the model should be.

Context

What it needs to know.

Instruction

What to do, precisely.

Format

How the output should look.

Examples

Lock the output format.

Anatomy of a prompt

A good prompt is structured, each part has a job.

one prompt · each part has a job

Role

You are a senior travel data analyst.

Context

Booking text: <booking> ... </booking>

Instruction

Extract the PNR, dates, total and currency.

Output format

Return strict JSON with exactly those keys.

Examples

"BA234, 320 EUR" → { amount: 320, currency: "EUR" }

A concrete example: the same five parts, filled in for a real extraction task.

What actually works (backed by labs / arXiv)

◆ **Explicit, precise instructions beat clever phrasing** · Claude 4.x follows literally; say WHY, not only WHAT.

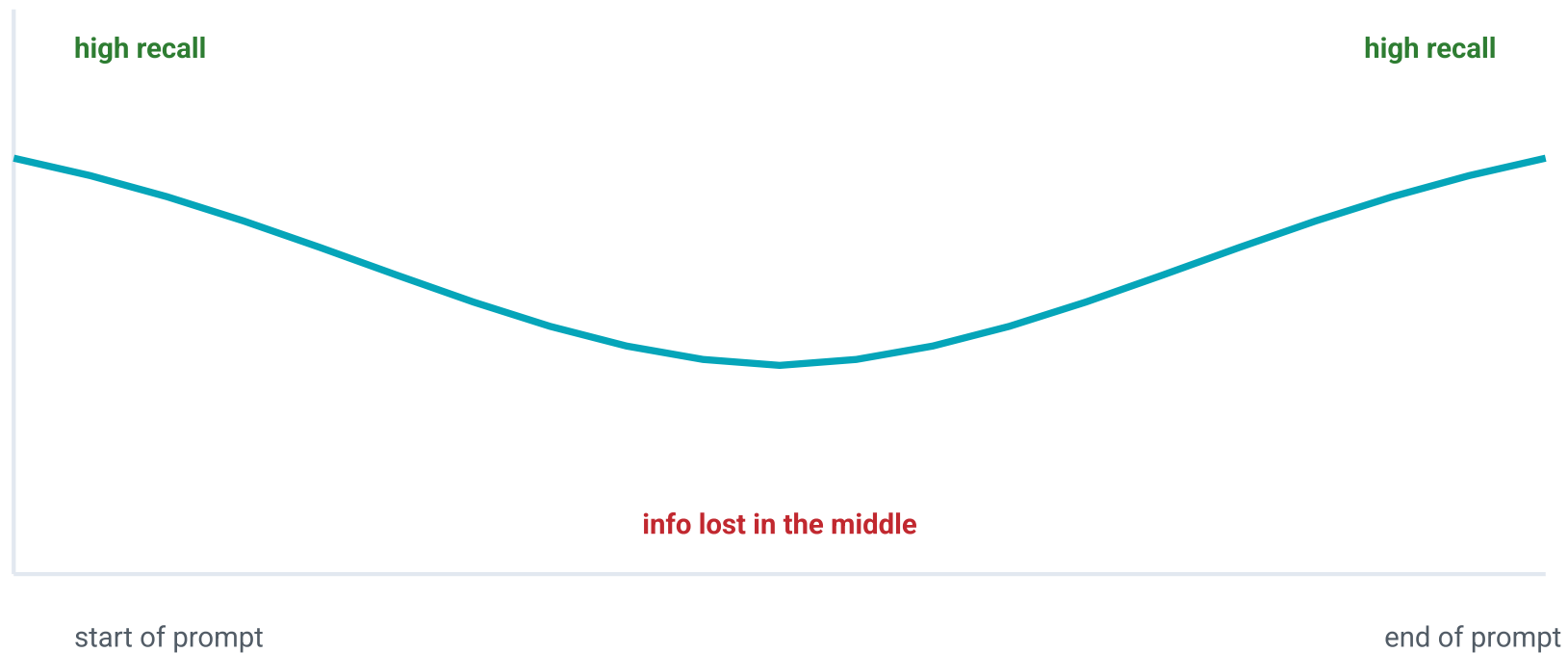
◆ **Structure / delimiters: XML tags (Claude), markdown / headers (OpenAI, Google) to separate instruction vs data.**

◆ **Avoid contradictory instructions** · they cost reasoning.

◆ **Metaprompting: let the model rewrite / optimise your prompt.**

Placement: "lost in the middle"

Models pay most attention to the start and the end of a long prompt, and can miss what is buried in the middle.



Put key instructions at the top or bottom · long docs up top, query at the bottom (up to +30%)

Less severe on modern long-context models, but still real · put your key instructions at the start or the end.

The repetition trick

Repeating the instruction (e.g. at the end) measurably helps NON-reasoning models.

◆ Google Research 2025: 47/70 tests won, 0 losses, no extra latency or output length.

◆ On reasoning models the effect is neutral · they already restate the task.

Few-shot vs zero-shot today

Zero-shot = you just ask. Few-shot = you show a few examples first.

ZERO-SHOT

- ◆ You give **only the instruction**, no examples.
- ◆ Best on **reasoning models** and clear tasks.

“Classify this review as positive or negative: ‘The seat was broken.’”

FEW-SHOT

- ◆ You add **2-3 example pairs** (input → wanted output) first.
- ◆ Mainly to **lock the output format** on non-reasoning models.
- ◆ On reasoning models it can **hurt** · prefer zero-shot.

“‘Great seats’ → positive · ‘Lost my bag’ → negative · ‘Seat was broken’ → ?”

Chain-of-thought, then and now

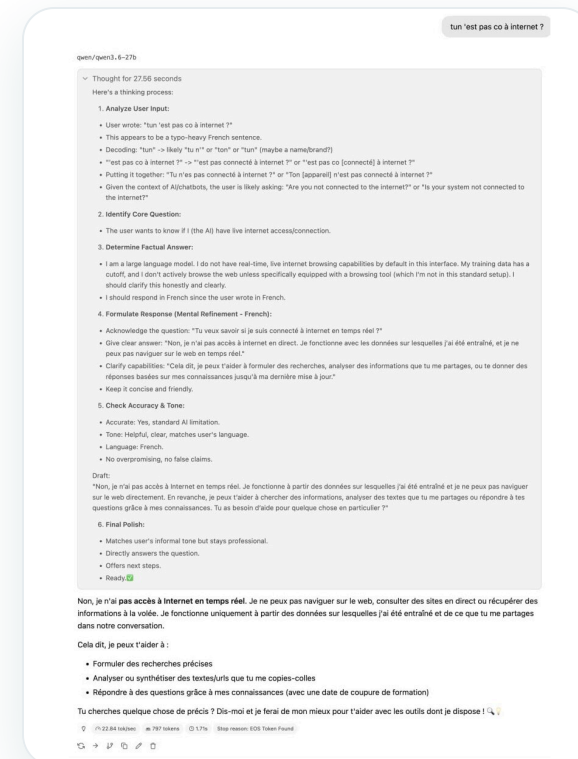
- ◆ CoT (“think step by step”) gives big gains mainly on **math / logic / symbolic** tasks.
- ◆ On reasoning models it is **largely redundant** · they reason internally · and can add **20–80% cost** for **≈3% gain**.
- ◆ OpenAI explicitly says: **avoid CoT prompts on reasoning models**. You pilot reasoning via the API instead.

— LIVE EXAMPLE · LOCAL MODEL

Chain-of-thought in the wild

Qwen3 running **locally** in **LM Studio** · you can watch the model think, then answer.

- ◆ The grey block is the **reasoning** (chain-of-thought): the model analyses a typo-heavy question, decodes it, plans the answer *before* writing it.
- ◆ **Runs on your machine** (LM Studio) · no cloud, nothing leaves the laptop. Useful for sensitive data.
- ◆ Bottom bar = **token economics**, live: 22.8 tok/s, 797 tokens, first token in **1.71s**, stopped on the **EOS** token.
- ◆ Reasoning is **billed output** · those thinking tokens cost money and add latency. Powerful, not free.



qwen/qwen3.6-27b

Thought for 2256 seconds

Here's a thinking process:

- Analyze User Input:**
 - User wrote: "tu n'est pas co à internet ?"
 - This appears to be a typo-heavy French sentence.
 - Decoding: "tu n" -> likely "tu n'" or "ton" or "tu n'" (maybe a name/brand)
 - "est pas co à internet ?" -> "est pas connecté à internet ?" or "est pas co (connecté) à internet ?"
 - Putting it together: "Tu n'es pas connecté à internet ?" or "Ton [appareil] n'est pas connecté à internet ?"
 - Given the context of AI chatbots, the user is likely asking: "Are you not connected to the internet?" or "Is your system not connected to the internet?"
- Identify Core Question:**
 - The user wants to know if I (the AI) have live internet access/connection.
- Determine Factual Answer:**
 - I am a large language model. I do not have real-time, live internet browsing capabilities by default in this interface. My training data has a cutoff, and I don't actively browse the web unless specifically equipped with a browsing tool (which I'm not in this standard setup). I should clarify this honestly and clearly.
 - I should respond in French since the user wrote in French.
- Formulate Response (Mental Refinement - French):**
 - Acknowledge the question: "Tu veux savoir si je suis connecté à internet en temps réel ?"
 - Give clear answer: "Non, je n'ai pas accès à internet en direct. Je fonctionne avec les données sur lesquelles j'ai été entraîné, et je ne peux pas naviguer sur le web en temps réel."
 - Clarify capabilities: "Cela dit, je peux t'aider à formuler des recherches, analyser des informations que tu me partages, ou te donner des réponses basées sur mes connaissances jusqu'à ma dernière mise à jour."
 - Keep it concise and friendly.
- Check Accuracy & Tone:**
 - Accuracy: Yes, standard AI limitation.
 - Tone: Helpful, clear, matches user's language.
 - Language: French.
 - No overpromising, no false claims.

Draft:

"Non, je n'ai pas accès à internet en temps réel. Je fonctionne à partir des données sur lesquelles j'ai été entraîné et je ne peux pas naviguer sur le web directement. En revanche, je peux t'aider à chercher des informations, analyser des textes que tu me partages ou répondre à tes questions grâce à mes connaissances. Tu as besoin d'aide pour quelque chose en particulier ?"

6. Final Polish:

- Matches user's informal tone but stays professional.
- Directly answers the question.
- Offers next steps.
- Ready.

Non, je n'ai pas accès à Internet en temps réel. Je ne peux pas naviguer sur le web, consulter des sites en direct ou récupérer des informations à la volée. Je fonctionne uniquement à partir des données sur lesquelles j'ai été entraîné et de ce que tu me partages dans notre conversation.

Cela dit, je peux t'aider à :

- Formuler des recherches précises
- Analyser ou synthétiser des textes/urls que tu me copies-colles
- Répondre à des questions grâce à mes connaissances (avec une date de coupure de formation)

Tu cherches quelque chose de précis ? Dis-moi et je ferai de mon mieux pour t'aider avec les outils dont je dispose !

0 / 22.84 tok/s | 797 tokens | 1.71s | Stop reason: EOS Token Found

Qwen3 · LM Studio · "thinking" mode · [click to zoom](#)

Controlling reasoning via the API

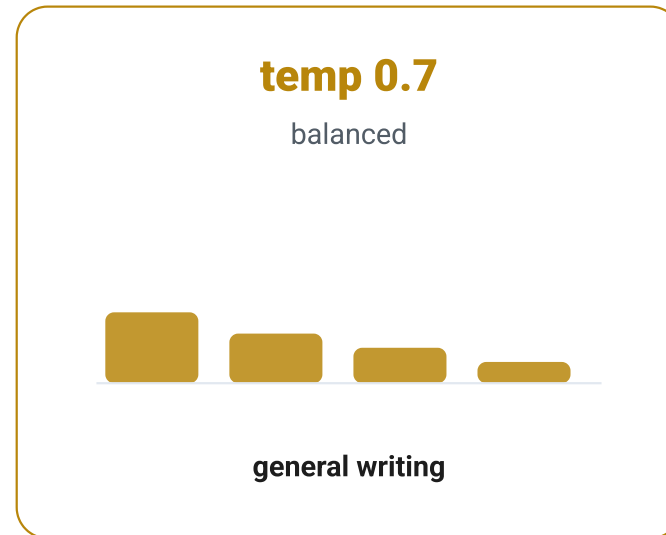
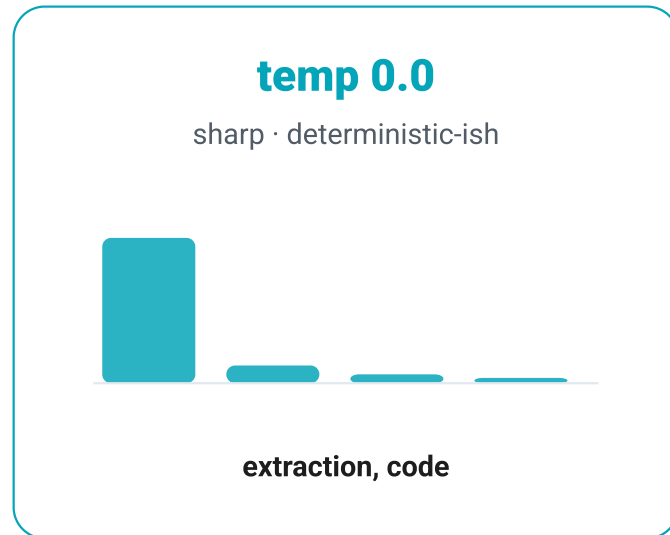
You control reasoning through API parameters, not prompt text.

LAB	PARAMETER	VALUES
OpenAI (GPT-5 / o-series)	reasoning_effort + verbosity	minimal / low / medium / high
Anthropic (Claude)	effort (adaptive thinking)	low / medium / high / xhigh / max
Google (Gemini 3.5 / 3)	thinkingLevel / thinkingBudget	low → high

*More reasoning tokens = better on hard tasks, but a direct **cost** / **latency** trade-off.*

Temperature & sampling

Temperature rescales the probability distribution: low = sharp, high = flat.



Top-p (nucleus): line the next-word candidates up by probability, keep just enough from the top to cover p of the total chance (say 90%), and pick from that shortlist · the unlikely long tail is dropped. Rule: tune temperature OR top-p, not both.

Temperature 0 is not determinism

- ◆ Even at temperature 0, hosted endpoints can return **different outputs**.
- ◆ Main cause: kernels are not **batch-invariant** · server batch size (out of your control) changes the result.
- ◆ Reasoning models (o-series, GPT-5) often **fix temperature / top-p at 1** (not changeable).

Structured output

Ask for strict JSON, not free text.

- ◆ Use JSON mode / schema enforcement.
- ◆ Function calling: the model fills typed arguments of a function you define.
- ◆ This is what turns an LLM into a software component.

Always validate the output (bridge to security)

- ◆ Treat LLM output as untrusted input.

- ◆ Validate / parse before using it (DB, shell, email, another API).

- ◆ This is OWASP LLM05 · Improper Output Handling, seen in module 5.



Prompting & structured output ≈25 min

From a fake travel booking, extract clean structured data.

— STANDARD TRACK

From a fake travel booking text, write a prompt that extracts clean **JSON** (PNR, dates, amount, currency). Compare **zero-shot vs few-shot**; vary the **temperature** and observe.

Environment: GitHub Copilot Chat (any model) or Google AI Studio. No company data.

— ADVANCED CHALLENGE

Enforce a **strict schema** via JSON mode / function calling; handle an **ambiguous input**; measure the **token cost** of each approach.



Quiz time!

Quick recap before lunch: tokens, models and prompting.

kahoot.it

game PIN on screen

Fundamentals · Tokenizers & cost · Prompt engineering

03

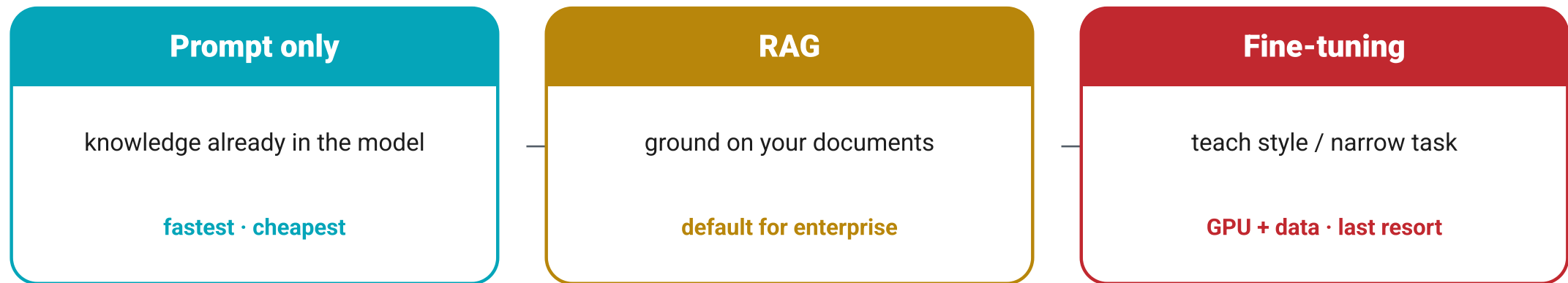
RAG and fine-tuning: choosing the right approach

Travel logs are text → RAG is very relevant. You won't put a 500 GB model on a server.



Prompt vs RAG vs fine-tuning

Three options · the single most useful slide to scope a project.



Start with a prompt · add RAG for grounding · fine-tune only when both fall short.

RAG vs long context (just paste the doc?)

Huge context windows make “paste the whole document” a real alternative to RAG.

CONTEXT STUFFING

- ◆ Paste the whole doc into the prompt · **no retrieval, no chunks.**
- ◆ Simple, no pipeline to build.
- ◆ Windows mid-2026: **1M common** (Claude, Gemini, GPT-5), up to **10M** (Gemini 3 Pro, Llama 4 Scout).

Best for a **few docs** and simplicity.

RAG

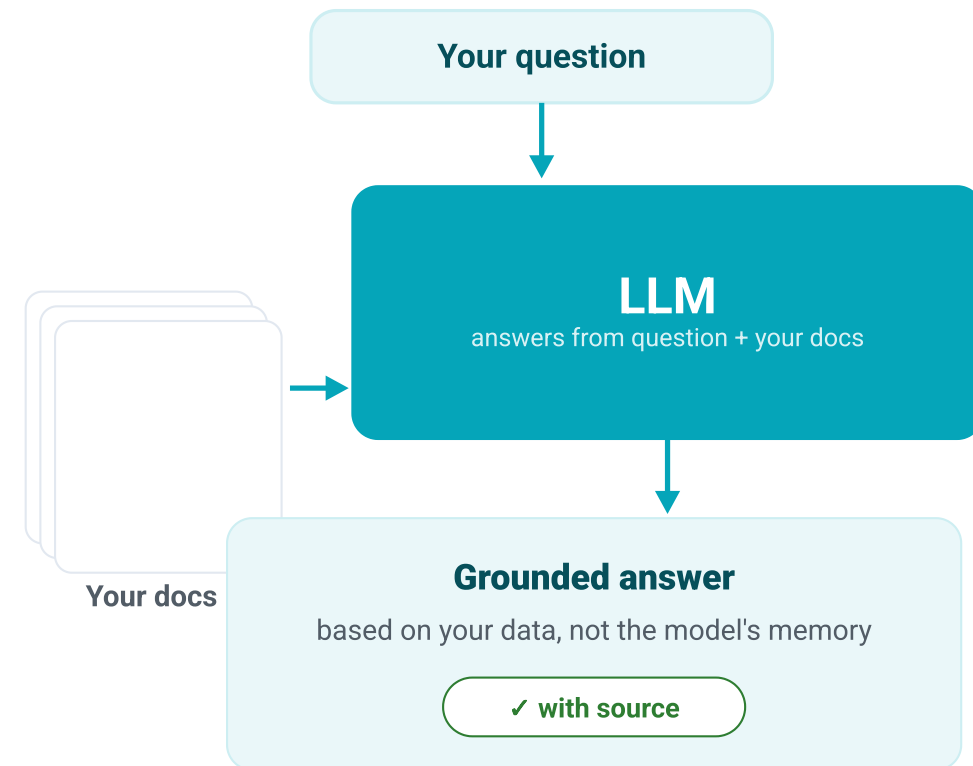
- ◆ Retrieve only the relevant passages, then answer.
- ◆ Stuffing is not free: you pay **all** input tokens **every call** (a 1M-token doc ≈ \$5 on Opus), and recall drops (lost in the middle).
- ◆ Scales to a **large, changing corpus**, with cost control and traceability.

Best for a **big or changing corpus.**

RAG · why

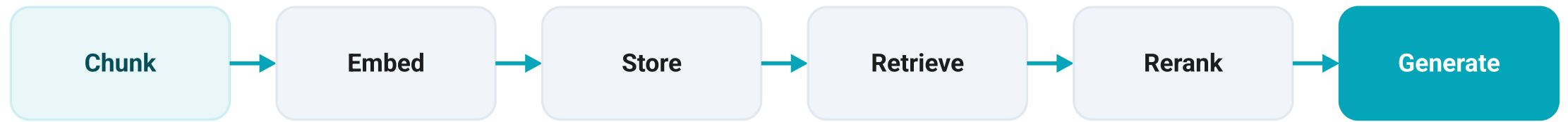
Retrieval-Augmented Generation: retrieve the right passages from your documents, then give them to the LLM to answer.

- ◆ **Grounded** answers → reduces hallucination.
- ◆ **Fresh / private** data without retraining.
- ◆ **Traceable** source citations.
- ◆ **No GPU** needed.



RAG · how it works (pipeline)

Chunk → Embed → Store → Retrieve → Rerank → Generate.



The query is embedded too, then matched against stored chunks · garbage in → confident wrong answers.

RAG · chunking & retrieval choices

◆ **Chunk size and overlap matter.**

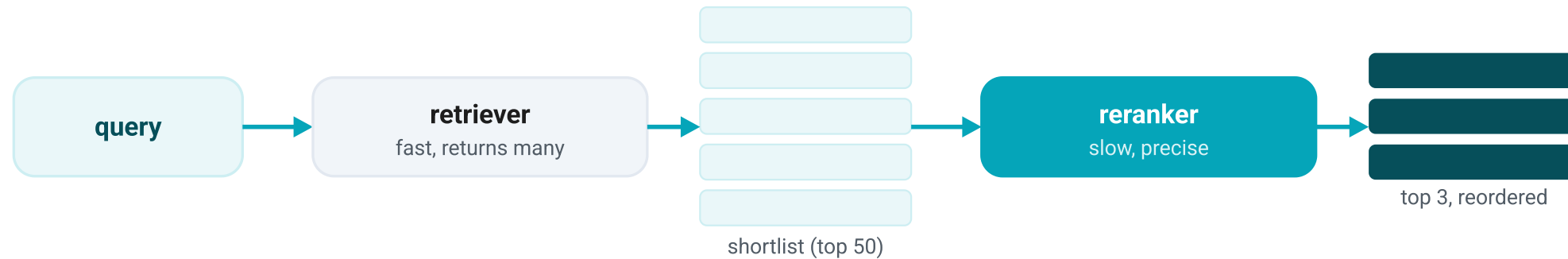
◆ **top-k controls how many passages you retrieve.**

◆ **Reranking reorders results for relevance · a big quality lever.**

◆ **Semantic chunking groups related sentences. Garbage in → confident wrong answers.**

Two-stage retrieval: search then rerank

A cheap retriever casts a wide net, a reranker keeps what is truly relevant.



Cheap search casts a wide net, the reranker picks what is truly relevant.

Principle redrawn from Hands-On Large Language Models (Alammar & Grootendorst).

RAG · pitfalls (security bridge)

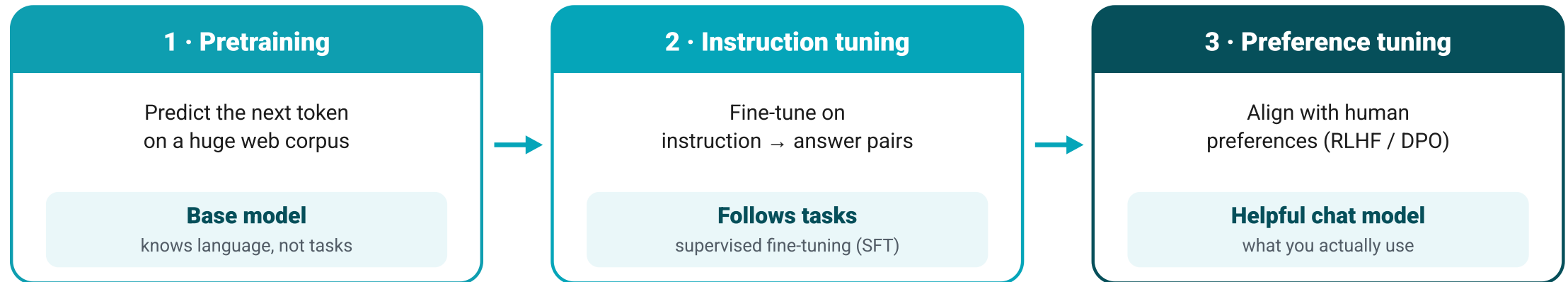
◆ **Bad chunking / retrieval → wrong context.**

◆ **Vector & embedding weaknesses (OWASP LLM08): RAG poisoning, cross-tenant leakage, embedding inversion.**

◆ **RAG is also a path for indirect prompt injection.**

How an LLM is actually built

Three stages turn raw text into a helpful assistant.



Most enterprise work happens AFTER stage 1: you consume a chat model, or lightly tune it.

*RLHF = Reinforcement Learning from Human Feedback · DPO = Direct Preference Optimization. On Hugging Face: **-base** just autocompletes, **-instruct** / **-chat** is the assistant · in production use instruct/chat, never base.*

RLHF: who actually gives the feedback?

The “human feedback” behind aligned models is a real, and expensive, human industry.

\$14.3B

Meta → Scale AI
for a 49% stake (2025)

\$25B

Surge AI valuation
≈\$1.4B revenue, almost all from AI labs

\$150+/hr

Expert annotators
far more for medical / legal specialists

- ◆ Annotation **quality is the real moat** · models are aligned by armies of paid experts, not by magic.

Fine-tuning · when it helps

- ◆ Lock a consistent style / format.
- ◆ Specialise on a narrow, repetitive task.
- ◆ Reduce prompt length once behaviour is learned.
- ◆ LoRA / QLoRA = lightweight tuning · train small adapters, not the whole model.

Lightweight fine-tuning: LoRA

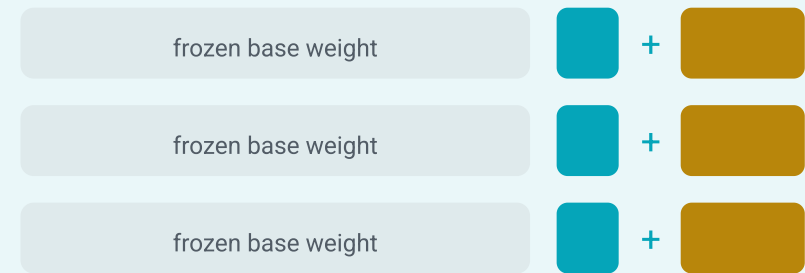
Freeze the base model, train a small low-rank adapter.

Full fine-tuning



update ALL weights · big GPU · one full copy per task

LoRA / QLoRA



freeze base, train a small low-rank adapter (AxB) · cheap, swappable

LoRA principle redrawn from Hands-On Large Language Models (Alammar & Grootendorst).

Fine-tuning · the real cost

- ◆ Needs GPU + a curated, labelled dataset.
- ◆ Must be redone when the base model changes.
- ◆ Often beaten by a good RAG + prompt.
- ◆ At Amadeus: rarely the first step.

Fine-tuning · real-world examples

Code models

Code Llama, Qwen2.5-Coder, DeepSeek-Coder · fine-tuned for code.

Domain models

Finance (BloombergGPT), medical (Meditron), legal (SaulLM).

“Uncensored” community fine-tunes

Guardrails removed, e.g. **Dolphin** (Llama / Mistral) or Nous-Hermes · shows fine-tuning can change behaviour, and why governance matters.

Instruction tuning + RLHF

How a base model becomes a chat model (e.g. Llama → Llama-Instruct).



A small local RAG (Ollama) ≈20 min

100% local, CPU-friendly, no Hugging Face download.

— STANDARD TRACK

Run a provided Python script (**LlamaIndex + ChromaDB**) that indexes 3–4 text files and answers from them. Change the docs and watch grounding; trigger a hallucination by removing the answer.

— ADVANCED CHALLENGE

Tune **chunk size and top-k**, add a **reranking** step; swap models.

Environment: Python + Ollama (llama3.2:3b + nomic-embed-text) + ChromaDB. Force embeddings via Ollama · Hugging Face is blocked. For non-tech: AnythingLLM.

04

Deployment, cost and token economics

—
The hot topic of the day · and the part
that hits your daily work most.



Choosing a model

- ◆ **Capability** for the task vs size / speed.
- ◆ **Cost per token** (input / output / cached).
- ◆ **Latency**.
- ◆ **Context window** needed.
- ◆ Match the model to the task · **don't default to the biggest.**



Deployment options

 **Managed API**

Fastest, pay per token.

 **Azure ML / cloud hosting**

More control.

 **Self-host / local SLM**

Sovereignty.



Tokens: input vs output vs cached

Every API call is billed in tokens · and the three kinds are not priced the same.

€ Input tokens

Everything you send: system prompt, history, retrieved docs, the user message.

⚡ Output tokens

What the model generates, one by one · usually the **most expensive** (often 4-5x input).

🕒 Cached tokens

A repeated prompt prefix can be cached · billed at a fraction of the input price.

- ◆ Knowing the split is the **basis of cost control**: shrink the input, cache the stable part, watch the output length.

Token monitoring

You cannot optimise what you don't measure.

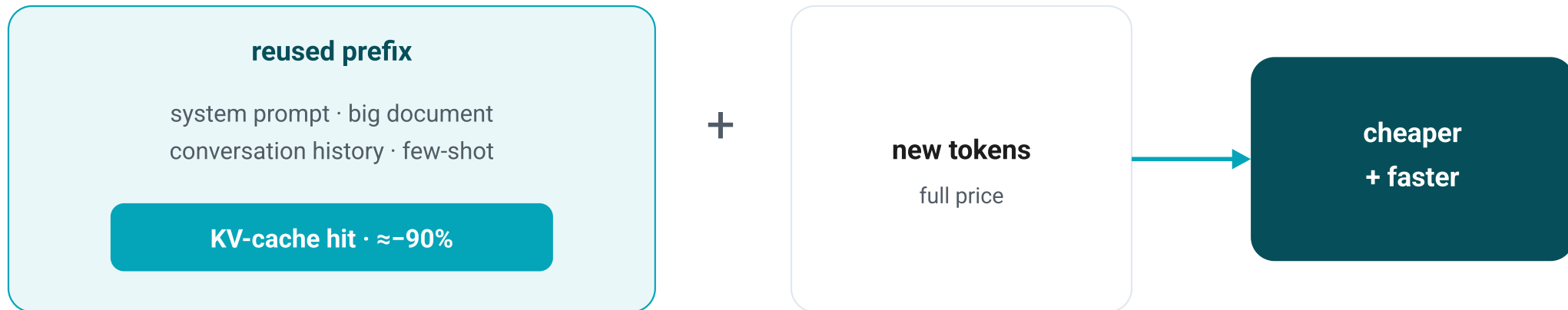
◆ VS Code now exposes token tracking.

◆ Provider dashboards.

◆ Per-model multipliers in Copilot.

Prompt caching · what & why

Reuse the same prompt prefix → the provider reuses the already-computed KV-cache instead of recomputing it.



Cached input ≈ 10% of input price across the big three · ideal for agents, chatbots, RAG.

Prompt caching · pricing

Cached input tokens cost $\approx -90\%$ across the big three.

PROVIDER	MECHANISM	CACHE READ	CACHE WRITE
OpenAI	automatic (≥ 1024 tokens)	10% of input	—
Anthropic	explicit (cache_control)	0.1× input (-90%)	1.25× (5 min) / 2× (1 h)
Google Gemini	implicit + explicit	-90% on 2.5+	—

A major FinOps lever. Anthropic charges for the cache write.



The 2026 memory crisis (why tokens got expensive)

+90–95%

DRAM, Q1 2026 QoQ
PC DRAM >+100%

≈+20%

HBM for 2026
now 23% of DRAM wafer capacity

3x

wafer for 1 bit of HBM
vs DDR5

- ◆ Memory makers shifted capacity to AI → a structural **“memory supercycle”**.
- ◆ HBM is the **critical component of inference GPUs** → hardware cost per token went up.

The invisible hike (tokenizer change)

A provider can keep the headline price unchanged but change the tokenizer so the same text produces more tokens.

◆ **Claude Opus 4.7 (Apr 2026) shipped a new tokenizer → reportedly +12–35% tokens for the same text.**

◆ **A higher bill that never appears on the pricing page.**

◆ **Lesson: $\text{cost} = \text{price-per-token} \times \text{number of tokens}$ · the tokenizer can move silently.**

GitHub Copilot goes usage-based (1/2) · the change

The angle for Amadeus · a heavy Copilot user.

- ◆ **GitHub Copilot is moving ALL plans to usage-based billing on 1 June 2026.**
- ◆ **From a flat “effectively unlimited” subscription to a model where every token counts.**
- ◆ **Announced 27 Apr 2026 by GitHub's CPO.**

GitHub Copilot goes usage-based (2/2) · how it works

New unit: AI Credits

1 credit = \$0.01. Metered per token at per-model API rates.

Plans unchanged

Each includes credits = its price: Pro \$10, Pro+ \$39, Business \$19, Enterprise \$39. Completions stay included.

Model multipliers matter

Cheap 0–1×, expensive much higher. Annual plans: Opus 15× → 27×, GPT-5.5 → 57× on 1 June.

For a big org

Real cost = underlying inference cost → same FinOps levers as raw API.



Case study: Uber

4 months

to burn the 2026 AI budget
exhausted by April

≈5,000

engineers on Claude Code
adoption 32% → 84%

≈70%

of committed code is AI-generated
\$500–\$2,000 / eng / month

- ◆ The COO publicly questioned the ROI: *“can't link tokens to features”*. Ties to OWASP LLM10 · **Unbounded Consumption.**

Case study: runaway bills

- ◆ Reported: a company accidentally ran up **≈\$500M in Claude charges in a single month** after failing to set usage limits (told to Axios; company unnamed).
- ◆ Anthropic added **rate limits** in 2025 to curb power users burning thousands on \$20 plans.
- ◆ Lesson: **no governance = denial of wallet.**



FinOps playbook

Three forces in 2026: vendors pass through inference cost, HBM raises it, APIs go up (or up “silently”).

 **Prompt caching**

-90% on reused context.

 **Model routing**

Small model for easy tasks.

 **Shorter prompts / batching**

Cut tokens, group calls.

 **Right model, right job**

Capability vs cost.

 **Budgets, quotas, alerts**

Governance by default.

 **English for mass automation**

Beat the token tax.

05

LLM security essentials

—
OWASP Top 10 for LLM Applications
(2025) · 1 concept + 1 real example per
risk.



“

**Treat every LLM output as
untrusted input.**

· The one principle to remember

OWASP Top 10 for LLMs (2025)

LLM07 and LLM08 are new in 2025; LLM09 = ex-“Overreliance”; LLM10 = ex-“Model DoS”, broadened.

LLM01 Prompt Injection

LLM03 Supply Chain

LLM05 Improper Output Handling

LLM07 System Prompt Leakage

new 2025

LLM09 Misinformation

LLM02 Sensitive Information Disclosure

LLM04 Data & Model Poisoning

LLM06 Excessive Agency

LLM08 Vector & Embedding Weaknesses

new 2025

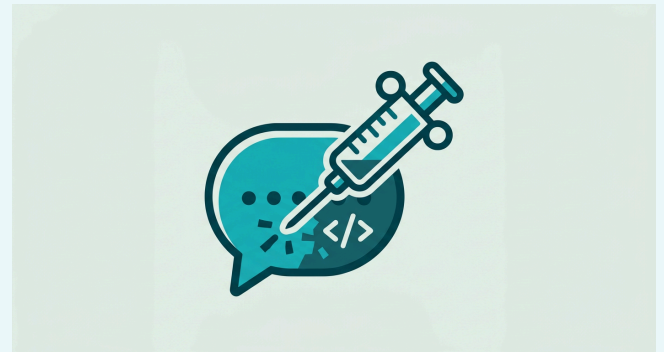
LLM10 Unbounded Consumption

LLM01 Prompt Injection

concept

An attacker slips instructions (visible or hidden) into input the model processes, changing its behaviour.

- ◆ **Direct** (user types it) or **indirect** (hidden in a document / email / web page read via RAG).
- ◆ #1 risk, hardest to fully fix: the model **doesn't natively separate “instruction” from “data”**.
- ◆ Mitigations: constrain behaviour, validate output formats, **least privilege on tools**, human approval for high-impact actions, tag untrusted content.



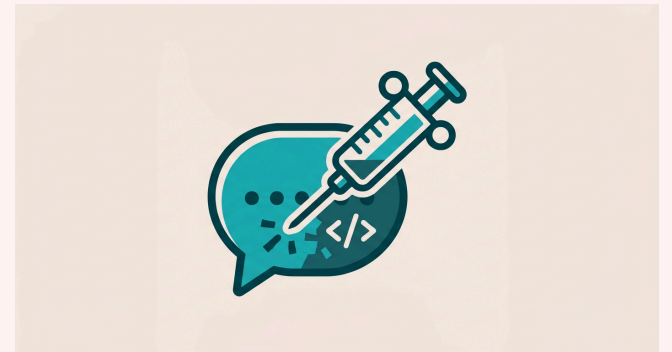
LLM01

LLM01 Prompt Injection

real-world example

EchoLeak (CVE-2025-32711, CVSS 9.3) · the first zero-click prompt injection on Microsoft 365 Copilot (Aim Labs, June 2025).

- ◆ A **booby-trapped email** (no user click) exfiltrates data Copilot can access.
- ◆ Lighter example: a **Chevrolet dealership chatbot** tricked into “agreeing” to sell a car for \$1.

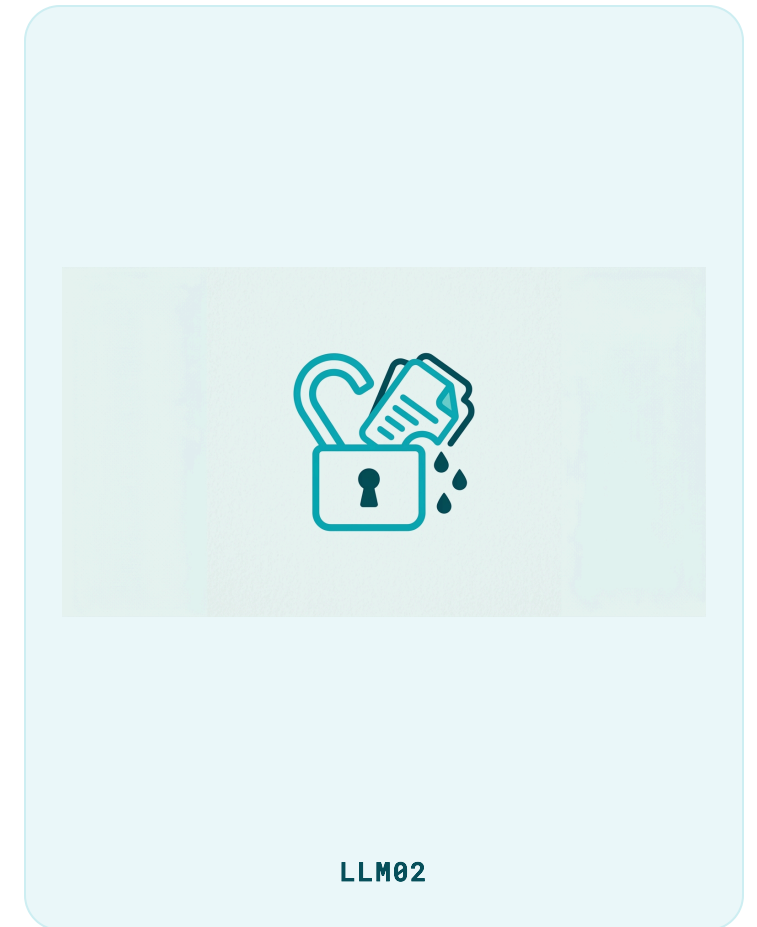
**LLM01**

LLM02 Sensitive Information Disclosure

concept

The model reveals sensitive data (PII, secrets, proprietary code) from its training data or from context users provided.

- ◆ Mitigations: **data sanitisation / scrubbing**, output filtering, strict access control, user awareness, privacy techniques.

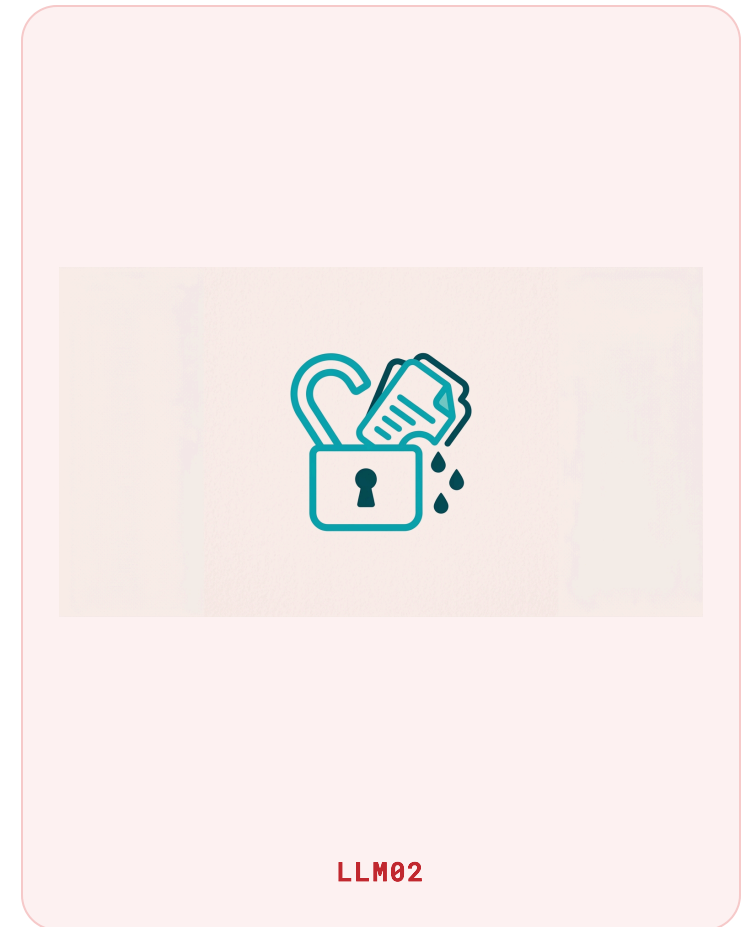


LLM02 Sensitive Information Disclosure

real-world example

Samsung (March 2023): engineers pasted proprietary semiconductor source code and internal meeting notes into ChatGPT.

- ◆ **3 incidents in under 20 days** → Samsung banned public generative AI.

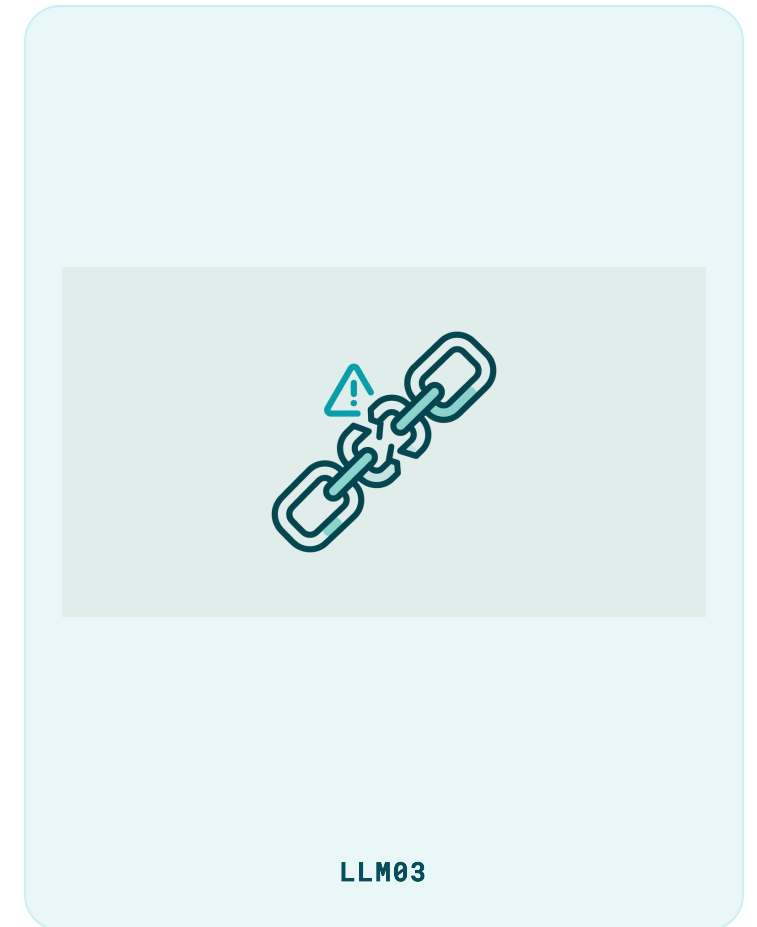


LLM03 Supply Chain

concept

Vulnerabilities from the chain: compromised pre-trained models, poisoned datasets, booby-trapped LoRA adapters, malicious dependencies, third-party models from hubs.

- ◆ Mitigations: verify **provenance / integrity**, AI-BOM / SBOM, trusted sources, component scanning.



LLM03 Supply Chain · slopsquatting

real-world example

LLMs **hallucinate package names** that don't exist; attackers register them with malware; a developer trusting the LLM installs it.

USENIX SECURITY 2025

- ◆ **19.7%** average package hallucination rate
- ◆ 5.2% commercial · 21.7% open-source models

STABLE, SQUATTABLE TARGETS

- ◆ **205,474** unique hallucinated names
- ◆ **43%** reproducible

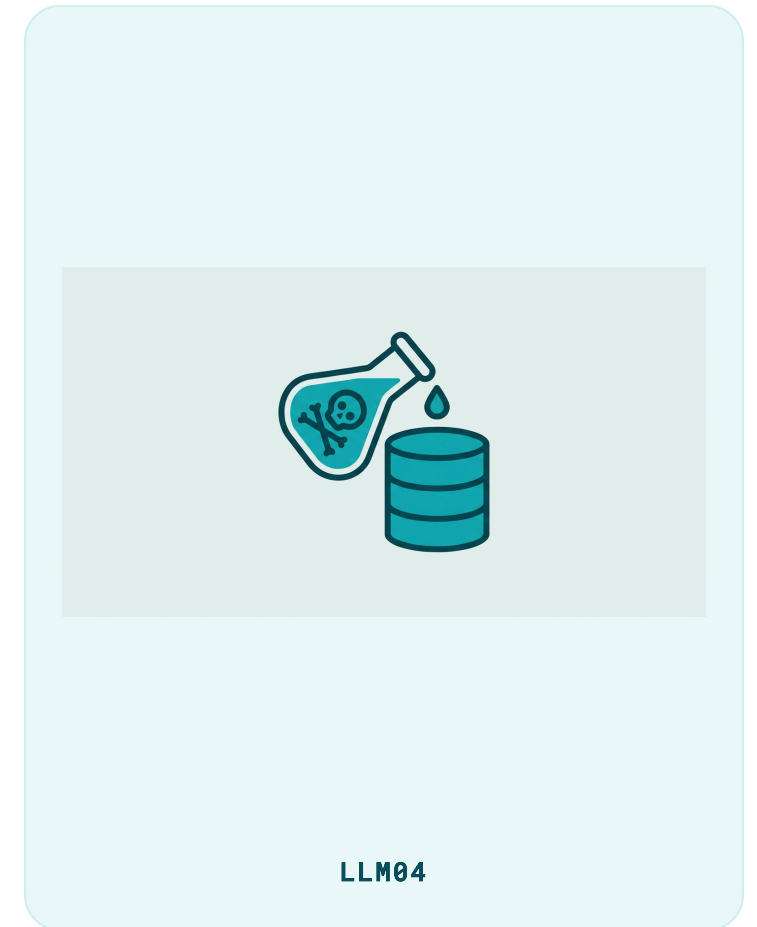
**LLM03**

LLM04 Data and Model Poisoning

concept

Malicious data injected during pre-training, fine-tuning or embedding to plant backdoors, bias or degraded behaviour · sometimes triggered by a hidden “trigger”.

- ◆ Mitigations: data provenance / integrity, **anomaly detection**, red-teaming, data lineage.



LLM04 Data and Model Poisoning

real-world example

PoisonGPT (Mithril Security, 2023, PoC): a model surgically edited to spread targeted misinformation while passing standard benchmarks.

- ◆ Shows you **can't trust a model just because it scores well**. (Documented proof-of-concept.)

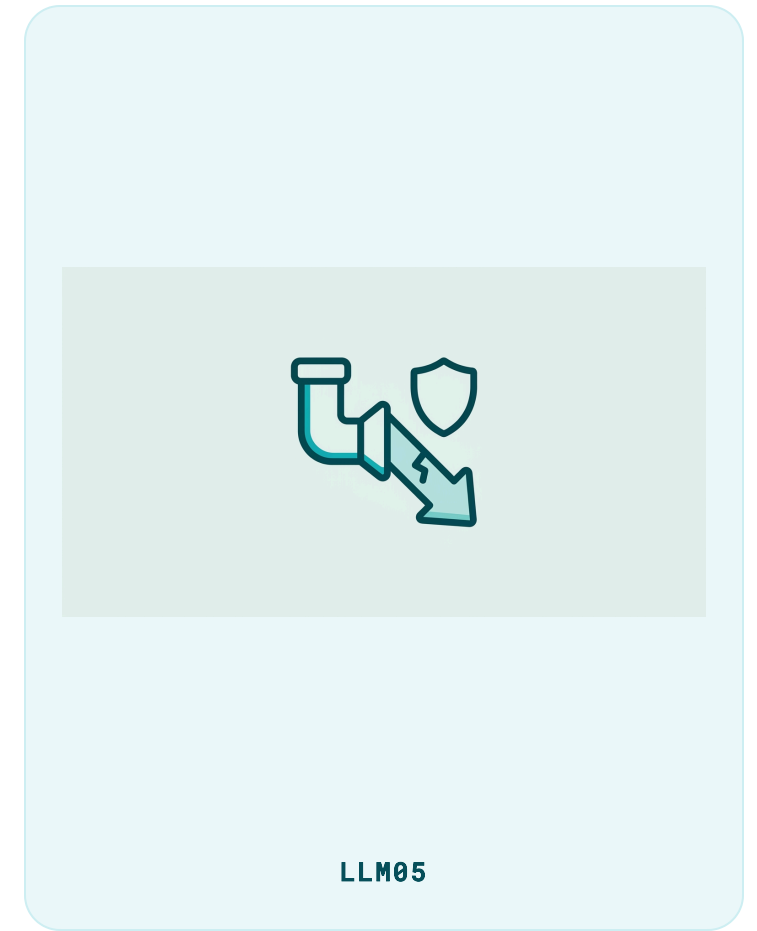
**LLM04**

LLM05 Improper Output Handling

concept

LLM output passed to a downstream component (browser, shell, DB, eval) without validation → XSS, SSRF, SQL injection, RCE.

- ◆ The **bridge between LLM risks and the classic web Top 10**.
- ◆ Mitigation: zero-trust output, contextual encoding, parameterised queries, CSP.



LLM05 Improper Output Handling

real-world example

An LLM emits Markdown with an `` pointing to an attacker server, with data in the URL → exfiltrates context.

- ◆ Exactly the mechanism inside **EchoLeak**.



LLM06 Excessive Agency

concept

Giving an LLM / agent too much functionality, permission or autonomy; a wrong or manipulated output triggers real-world actions.

- ◆ Major risk of 2025–2026 **agentic** systems (delete data, send money / emails).
- ◆ Mitigations: minimal tools, fine-grained permissions, **least privilege**, **human-in-the-loop**, audit trail.



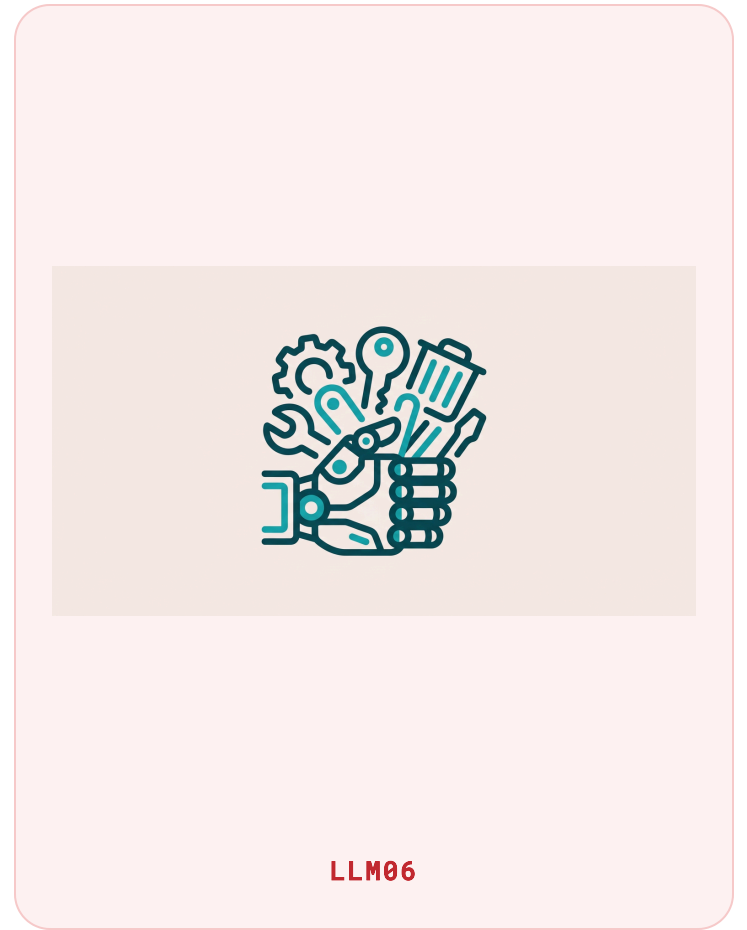
LLM06

LLM06 Excessive Agency

real-world example

An email assistant allowed to “delete”, hit by a prompt injection inside an incoming mail, **empties the inbox**.

- ◆ Ties directly to Unbounded Consumption (Uber-style runaway agents).

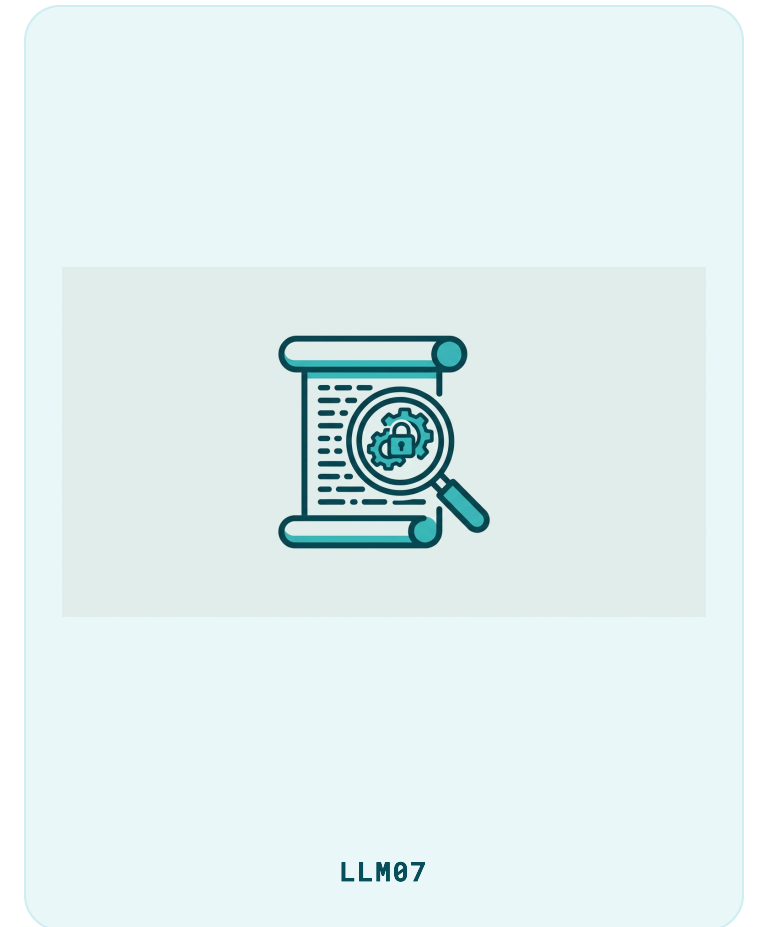


LLM07 System Prompt Leakage (new 2025)

concept

The system prompt (rules, sometimes secrets) is extracted by users.

- ◆ The real problem: putting things in the system prompt that **should never be there** (keys, security rules, business logic).
- ◆ Mitigation: never put secrets in the prompt, **enforce security OUTSIDE the LLM**, don't rely on the prompt for security.

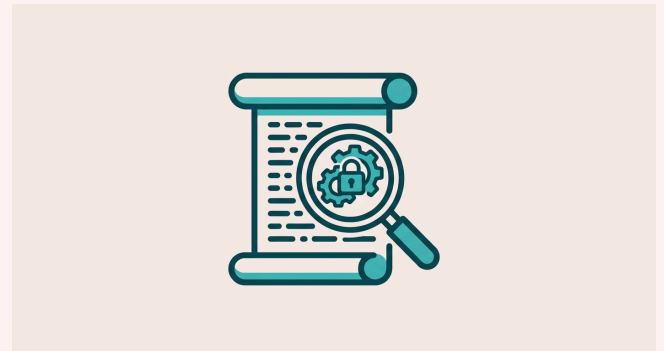


LLM07 System Prompt Leakage

real-world example

Throughout 2024, many commercial assistants' system prompts were extracted publicly via simple “**repeat your instructions**” requests.

- ◆ Exposing guardrail logic and making bypass easier. (Connect to the Gandalf lab.)

**LLM07**

LLM08 Vector and Embedding Weaknesses (new 2025)

concept

Weaknesses specific to RAG.

- ◆ Poisoning the **vector store**, **cross-tenant leakage** in a shared DB, **embedding inversion**, indirect injection via indexed documents.
- ◆ Mitigations: access control & **partitioning** of the vector DB, validate ingested data, per-tenant permissions, integrity monitoring.



LLM08 Vector and Embedding Weaknesses

real-world example

RAG poisoning: an attacker drops a booby-trapped document into an indexed SharePoint.

- ◆ Later it is retrieved and its hidden instructions run (indirect prompt injection).
EchoLeak also abuses this RAG surface.

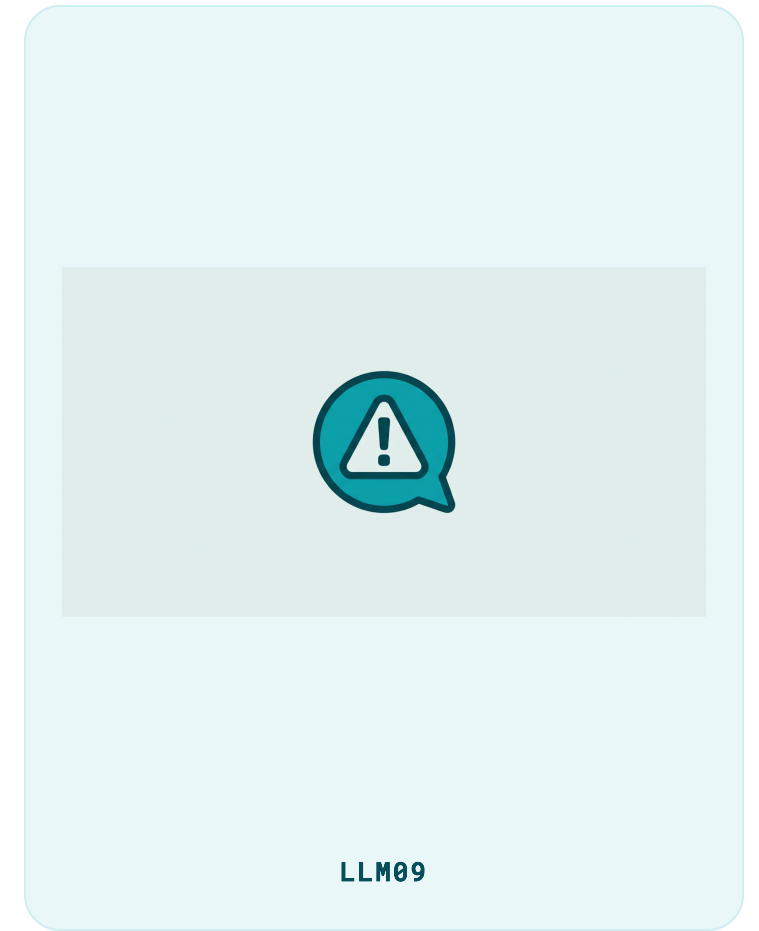
**LLM08**

LLM09 Misinformation (ex-Overreliance)

concept

The model produces false-but-credible information (hallucinations) that users rely on without checking.

- ◆ Includes **hallucinated code** (packages / APIs → slopsquatting).
- ◆ Mitigations: RAG with trusted sources, cross-checking, human oversight, citations, clear limits.

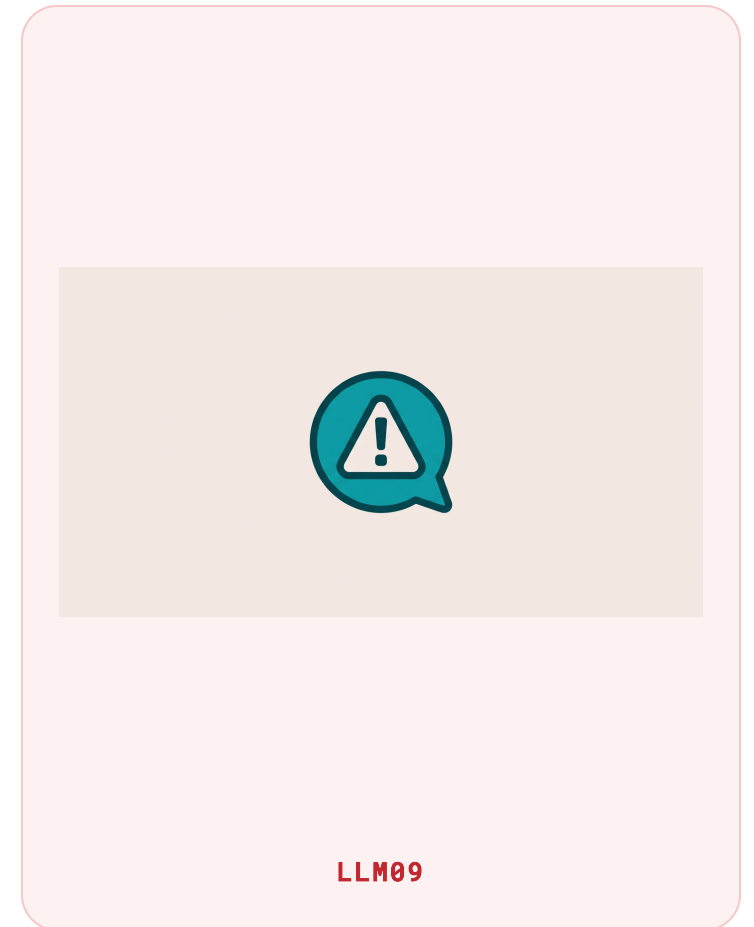


LLM09 Misinformation

real-world example

Air Canada (2024): the chatbot invented a bereavement-fare refund policy.

- ◆ The tribunal held the airline **liable**, rejecting “the chatbot is a separate legal entity”.
- ◆ First well-known “**the company owns its bot's words**” ruling.

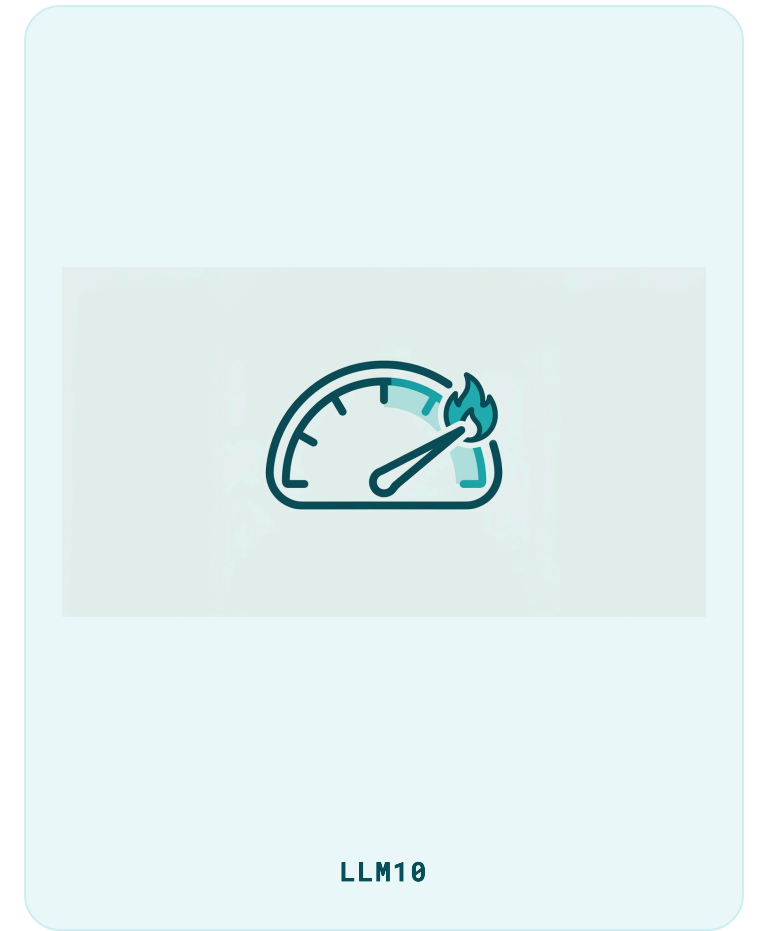


LLM10 Unbounded Consumption (ex-Model DoS)

concept

Uncontrolled resource use: massive / expensive requests, denial of service, “denial of wallet”, and model extraction.

- ◆ Mitigations: **rate limiting, quotas, input size limits, budget caps + alerts, timeouts, monitoring.**



LLM10 Unbounded Consumption

real-world example

The token-cost cases from module 4 ARE this risk in production.

- ◆ Uber's 4-month budget burn and the reported \$500M single-month bill = **denial of wallet at scale**.
- ◆ Governance (budgets, quotas) is the fix.



The flip side: AI as a defender

AI is both the threat and the counter-fire.

Artemis (OOPSLA 2025)

Static taint analysis + LLM to find SSRF: **106 real SSRF on 250 PHP apps, 35 new, 24 CVEs.**

Claude Mythos (Project Glasswing)

Cyber preview: **23,019 vulnerabilities** flagged across 1,000+ open-source projects, **~6,200 critical/high.**



The new dev reality: audit AI code before prod

≈2.7×

more XSS in AI-assisted PRs

1 in 3

suggested snippets has an exploitable flaw

42–48%

real bugs caught by AI review

vs <20% for traditional SAST

- ◆ 100% of orgs have AI code in prod, only ≈19% have full visibility. Response: **AI code review before deployment.**
- ◆ Tools: Copilot Autofix, Snyk, Semgrep Assistant, Cursor Bugbot, CodeQL. AI **augments**, doesn't replace · keep the human in the loop.



Prompt injection in practice (Gandalf) ≈20 min

Online, no install · make the LLM leak a secret password across rising defence levels.

— STANDARD TRACK

Gandalf (Lakera): levels 1–4 solo, then a group debrief on the 3 defence layers (system prompt, input guard, output guard).

Environment: gandalf.lakera.ai · hackmerlin.io · portswigger.net/web-security/llm-attacks

— ADVANCED CHALLENGE

HackMerlin or a **PortSwigger** LLM lab (indirect injection).



Quiz time!

Afternoon recap: RAG, cost, security and agents.

kahoot.it

game PIN on screen

RAG & fine-tuning · Token economics · OWASP & agents

06

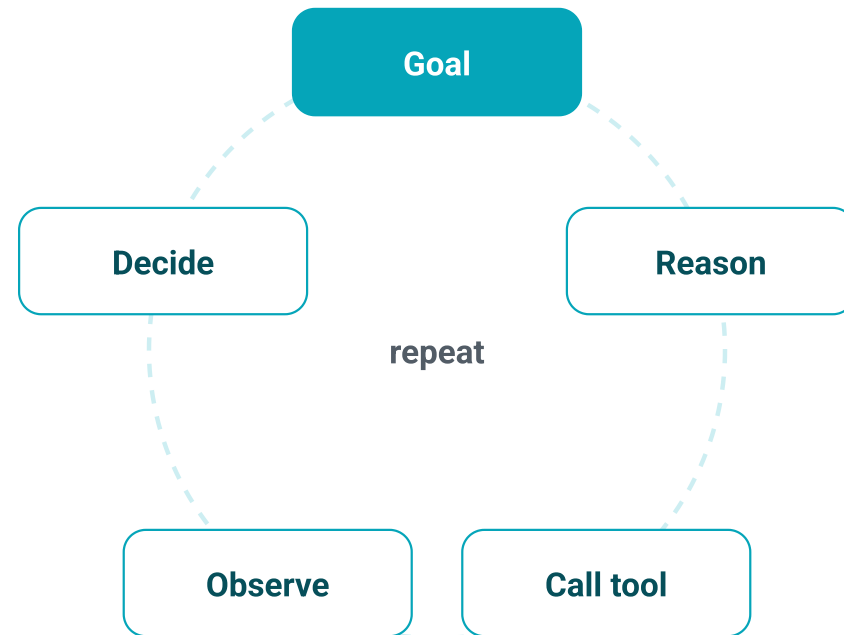
Trends and perspectives

Where this is heading, and how to keep up.



Agentic AI and tool use

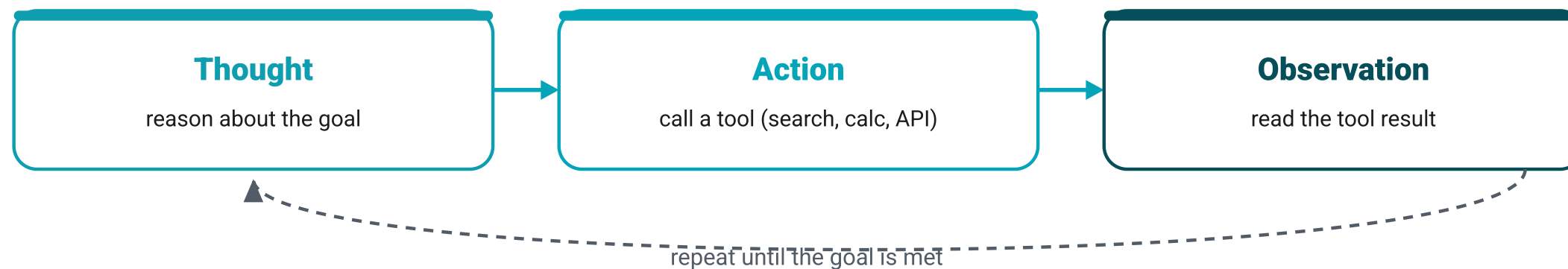
An agent is an LLM that can call tools, observe the result, and decide the next step.



An agent calls tools, observes the result, and decides the next step · more power = more risk (LLM06).

The ReAct loop

Thought, Action, Observation, repeated until the goal is met.

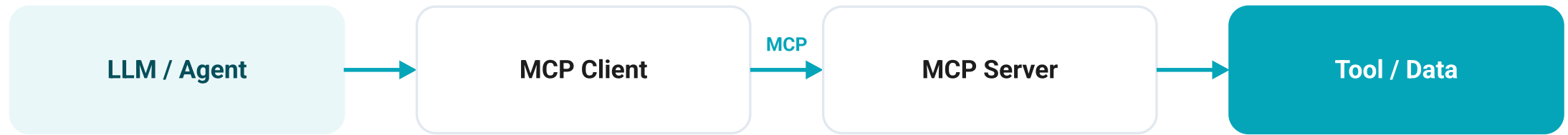


ReAct = Reasoning + Acting, the core loop of a tool-using agent.

Principle redrawn from Hands-On Large Language Models (Alammar & Grootendorst).

MCP · Model Context Protocol

An open standard (Anthropic, Nov 2024) · “a USB-C port for AI apps”.



“A USB-C port for AI apps” · primitives: Tools (actions) · Resources (data) · Prompts (workflows)

Open standard (Anthropic, Nov 2024) · now supported by OpenAI, Microsoft, Google



MCP · concrete example

An agent with an MCP “Calendar” server + an MCP “hotels DB” server.

- ◆ *“What are my free slots next week, and find a hotel?”*
- ◆ It reads the calendar (**resource**) and calls hotel search (**tool**).
- ◆ MCP is now the **de-facto standard** for agentic integration.



UCP · agentic commerce

Universal Commerce Protocol (Google + Shopify, 20+ partners) · “the MCP of commerce”.

- ◆ A common language for AI agents to **discover products, manage a cart and check out at any merchant, without per-merchant integration.**
- ◆ **Interoperates with MCP, A2A and AP2.**
- ◆ **Extended to lodging / hotels in May 2026.**



UCP · the Amadeus angle (nuanced) + AP2

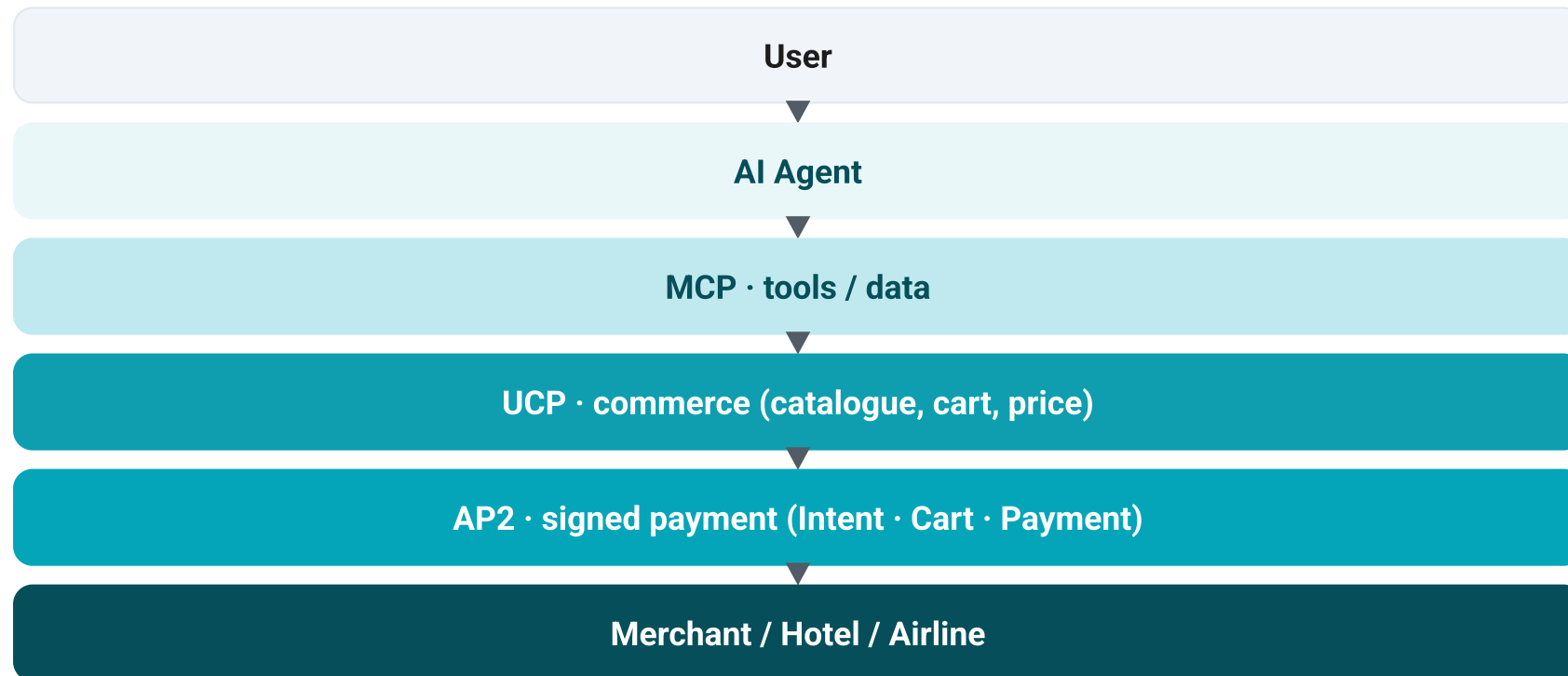
“Book me a hotel in Nice for the first weekend of November, budget 700 EUR.”

- ◆ The agent queries inventory / prices via **UCP**, builds the cart, pays via **AP2** (3 signed mandates: Intent, Cart, Payment).
- ◆ **Amadeus position** (CTO, 21 May 2026): MCP is “only a first step”; current protocols are “retail-native, not yet travel-ready”.
- ◆ Amadeus positions itself as a **neutral execution / trust layer**.
Rival: **ACP** (OpenAI / Stripe, Instant Checkout in ChatGPT).



The agentic stack (layers)

MCP = tools · UCP = commerce · AP2 = payments.



Rival commerce standards: UCP (Google / Shopify) vs ACP (OpenAI / Stripe)

Small Language Models (SLM)

Tiny models (sub-4B) that run on a laptop / phone, no GPU.

◆ Via Ollama: llama3.2:1b/3b, qwen2.5:0.5b-3b, gemma2:2b, phi4-mini.

◆ Rule: pick a model whose size < available RAM - 2 GB.

◆ Models pulled from the Ollama registry, not Hugging Face.



SLM use cases

Anonymise before a big model

PII masking before sending data upstream.
(Microsoft Presidio for industrial use.)

Guardrails

Around a bigger model: Llama Guard 3,
granite3-guardian.

On-device / sovereignty

Keep sensitive data local.

Quantization at the edge (recap)

The path to near-free, on-device AI without a GPU.

- ◆ Quantization (module 1) + **1-bit BitNet**.
- ◆ A **4-bit 7B** fits a laptop.
- ◆ A **1.58-bit BitNet** runs a big model on a CPU.



Evaluation · why

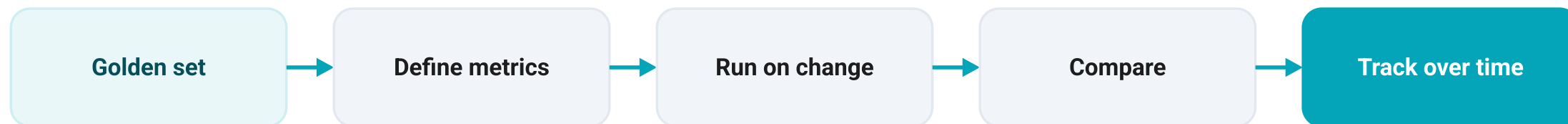
“How do I know an upgrade (5.4 → 5.5) didn't break my agent?”

◆ You can't ship LLM features without **evaluation**.

◆ Outputs are **non-deterministic** and models **change under you**.

Evaluation · how

Build a small golden set, define metrics, re-run on every change.



Regression testing for LLMs · correctness + cost + latency · LLM-as-judge for open-ended outputs.

*LLM-as-judge for open-ended outputs · track quality, cost and latency. Use eval tooling (W&B Weave, LangSmith, Langfuse, MLflow) to log runs and **compare versions** on every upgrade (5.4 → 5.5).*

Key benchmarks: read them critically

The Sparks program names these · know what each one really measures.

✓ MMLU

Multiple-choice, 57 subjects. **Now saturated** (top ≈90%), no longer separates the best. Successors: MMLU-Pro, **GPQA Diamond** (graduate-level).

</> SWE-bench Verified

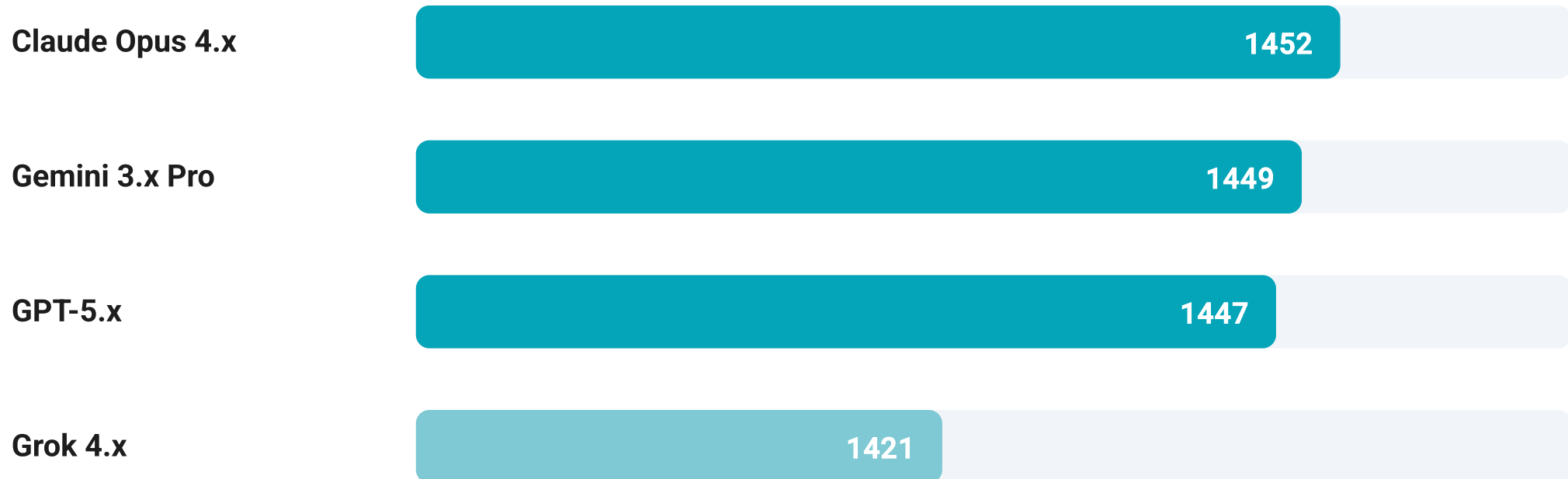
Real GitHub issues · the model must ship a **patch that passes the tests**. The “can it actually code” test. Mid-2026 top ≈**85-89%**, varies a lot by harness.

🎯 Read critically

Watch **contamination, saturation, gaming**, and the gap to **your** task. Build your own golden set.

Benchmarks & LM Arena

LM Arena ranks models by blind human votes (two anonymous answers → Elo).



Mid-2026 top 3 are statistically tied · blind human votes → Elo · always validate on YOUR use case.

*Limits: vote gaming, models “optimised for the arena”. **Show the live leaderboard on the day** rather than a frozen number.*

Where the research actually lives

Module 6 says “keep up” · here is where to look.

- ◆ **arXiv** · the preprint server for AI papers (cs.CL / cs.LG). **Caveat: not peer-reviewed, read it critically.**
- ◆ **Hugging Face** (models, datasets, leaderboards, Daily Papers) and **Papers with Code** (paper + code + benchmark in one place).
- ◆ **GitHub** for the code, and the **lab blogs** (Anthropic, OpenAI, Google DeepMind, Meta) for what is actually shipping.

The signal is high, but so is the noise · a preprint is a claim, not a proven result.

Multimodal models

Modern frontier models handle text + images (+ audio / video).

- ◆ Read a screenshot, a chart, a scanned PDF; describe or extract.
- ◆ Relevant for **travel docs** (tickets, receipts).
- ◆ Detection angle: **watermarking** (SynthID, C2PA) to tell AI-generated content apart.



Key takeaways

- 1** An LLM **predicts the next token** · its power and its limits.
- 2** **Tokens are money**: tokenizer, caching, the 2026 cost surge.
- 3** **Prompt** → **RAG** → **fine-tune**, in that order.
- 4** Modern prompting = **clarity + structure + control the reasoning via API**.
- 5** Treat every output as **untrusted**; OWASP Top 10 is your checklist; AI is **both threat and defender**.
- 6** **Agentic AI + MCP / UCP** are the next wave; **evaluate everything**.

Going further: internal Amadeus trainings

The topics we deliberately kept out of today, covered by your internal programmes.

Ethics & societal impact

Internal Amadeus training (link to be added).

Regulation & compliance

GDPR, EU AI Act · internal Amadeus training (link to be added).

thank you.

Questions and discussion · Dimitri Foucault

