



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

SoSe 2026

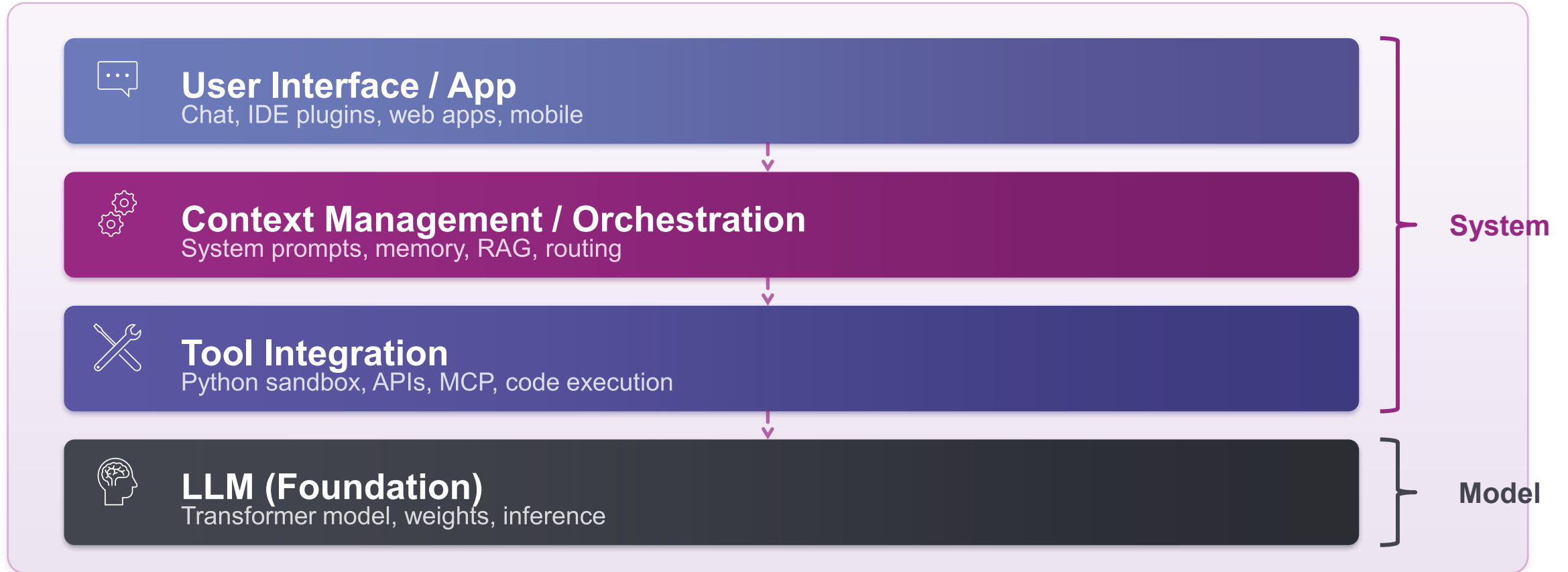
AI Supported Software Development

Steering the Distribution

Hallucination, Compliance and
Prompting

AI Systems in a nutshell

The model is only a (small) part



The LLM provides the **intelligence**, the system provides the **capability**

Four-Act Structure

Act I Experience Weeks 1–2

“The Wow and the Oops”

We USE AI tools before understanding them. We experience both the power and the failures firsthand, creating the cognitive tension that motivates everything that follows.

Act II Understand Weeks 3–7

“How Does This Actually Work?”

We learn foundational concepts: tokenization, embeddings, attention, context windows, inference, hallucination mechanisms. Prompting is taught as the applied side of these fundamentals.

Act III Master Weeks 8–10

“Wielding It Professionally”

We learn to evaluate AI output quality rigorously, integrate AI into real development workflows (IDE assistants, agentic pipelines, CI/CD), and measure productivity impact.

Act IV Govern Weeks 11–13

“The Bigger Picture”

We confront risks, security, organizational impact, and the economics of AI adoption. The course ends with you defending your own AI integration strategy.

AI AUTONOMY

Chat Assistant

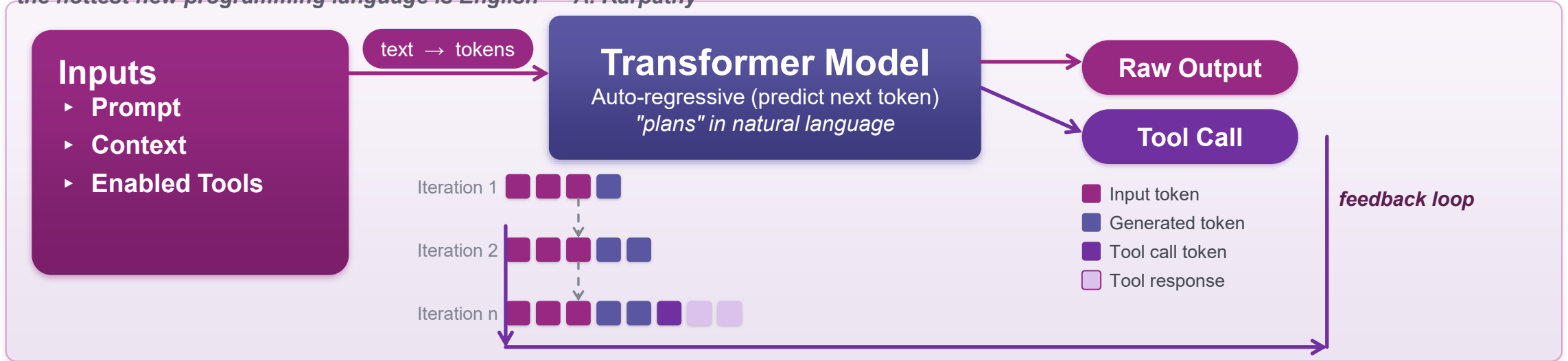
Knowledge Partner

Workflow Integrator

Autonomous Agent

LLM in one picture

“the hottest new programming language is English” – A. Karpathy



Glossary

- **Prompt** (user provided input instructions), **Context** (extra text the model sees)
- **Token** (sub-word unit, model operates on tokens)
- **MCP** (generic interface / protocol to make any external tool available to an LLM)

Tools / Connectors

- External actions (HTTP, email, git, database, ...) **exposed to the model**
- On the fly **code generation & execution**

Two questions, two blocks

Block A

Why the mechanism fails

§2–§8: pre-training, post-training, hallucination, CLAUDE.md compliance, lost-in-the-middle, three-layer model.

Block B

Engineering the distribution

§9–§16: temperature, in-context learning, few-shot, chain-of-thought, self-consistency, role priming.

Five named patterns

Zero-shot » few-shot » chain-of-thought » self-consistency » role priming.

Every pattern is a named use of L5

- Every prompting pattern that appears in this lecture — zero-shot, few-shot, chain-of-thought, self-consistency, role priming — is a **deliberate, named use** of the mechanism we built last week.
- No folklore. Mechanism + citation only.

Why the mechanism fails

One training objective

Pre-training in one sentence

Minimise cross-entropy loss on next-token prediction, trillions of tokens, billions of parameters. The result: a model that emits a **probability distribution over the next token**.

Everything that follows — hallucination, compliance failure, prompting patterns — is a consequence of this single objective.

What the Model does NOT have

No truth

Training data mixes correct and incorrect text. No flag marks “wrong.”

No goals

Not trying to help. Emitting a distribution shaped by training statistics.

No grounding

Cannot check if an API exists. Cannot run code. Cannot consult docs.

No memory

Each forward pass sees only the current context window. Nothing persists.

Base Model: Two Continuations

Prompt: ‘‘What is the capital of France?’’

Base model

Continuation A

“Paris.” — Wikipedia pattern.

Base model

Continuation B

“What is the capital of Germany?
What is ...” — quiz-list pattern.

Both have non-trivial probability. The base model does not prefer the helpful answer.

SFT → RLHF: The Post – Training Pipeline



Pre-training

Next-token prediction on trillions of tokens. Result: pattern completion.

Supervised Fine-Tuning

SFT

Train on curated conversations. Teaches the **format** of an assistant.

Reinforcement Learning from Human Feedback

RLHF

Optimise against human-preference scores. Teaches which responses humans **prefer**.

Supervised Fine-Tuning (SFT)

What it buys

Instruction following
Confident, structured style
Role hierarchy ([system] > [user])
Refusal of harmful requests

What it does NOT buy

Truth or factual accuracy
Hallucination elimination
Reliable refusal
Alignment with truth

User: Write a function that validates an email.

[System]Assistant:

```
def validate_email(address |
```

SFT is the reason the conversation really feels like one – Chat Bot

Reinforcement Learning from Human Feedback (RLHF)

- It is a refinement on top of SFT, not a replacement for it.
- SFT teaches the model what an assistant looks like.
- RLHF teaches the model which of several plausible assistant-style responses humans prefer.

Common misconception

RLHF shifts **style and behaviour**, not knowledge. A post-trained model hallucinates **more politely**, not less often.

DeepSeek-R1 and the GRPO frontier

The frontier has moved

DeepSeek-R1 (Guo et al., 2025): pure RL against a correctness signal can elicit complex reasoning. GRPO replaces PPO. SFT can be skipped entirely (R1-Zero).

The principles of §3 hold: post-training shapes behaviour, not knowledge. The algorithms evolve; the operational truth does not.

*PPO – Proximal Policy Optimization

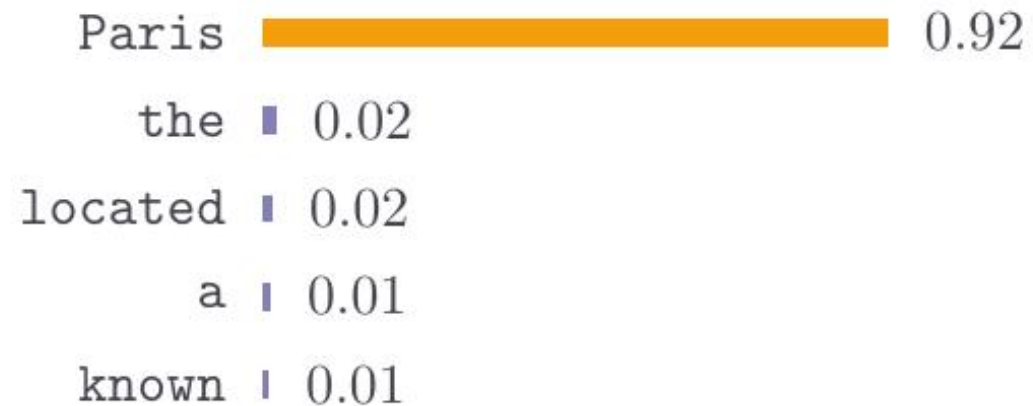
*GRPO – Group Relative Policy Optimization

Hallucination is structural, not a bug

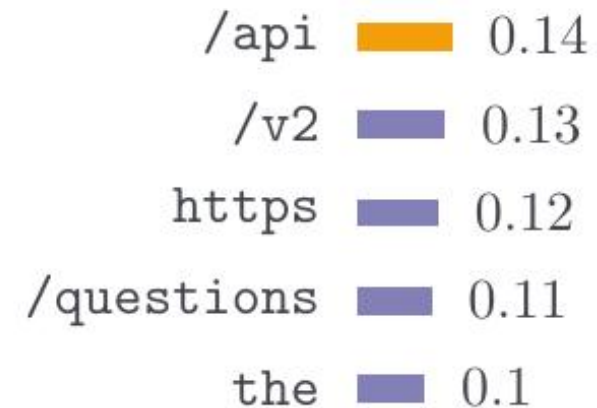
- Hallucination is not a bug. It is the **expected behaviour** of next-token prediction when the context is insufficient to peak the distribution onto a correct answer.

Peaked vs flat: the shape of the distribution

Peaked — “capital of France is ...”



Flat — “SE API endpoint is ...”



The model **always** picks a token — even from a flat distribution.

Factual / structural / confident-nonsense

Not all hallucinations are alike. They share a mechanism, but the way they manifest — and the way you should engineer against them — depends on what kind of structure the model is fabricating.

Factual

Wrong facts

Invented API endpoint:
`/api/v2/questions/monthly`

Structural

Wrong API surface

Fabricated function:
`locust.stats.get_monthly_report`

Confident-nonsense

Wrong explanation

Plausible prose about requests.
`DEFAULT_POOL_SIZE`
— none of it real.

Per-category examples and mitigations

Category	Example	Root cause (L5)	Mitigation
Factual	Invented URL	Flat dist. over rare entities	Retrieval (docs in context)
Structural	Fabricated function	Pattern matches shape, not surface	Grounding (API stubs, types)
Confident-nonsense	Plausible wrong explanation	RLHF style on flat dist.	Verification (run code, review)

Checkpoint: diagnostic foundation

Recap

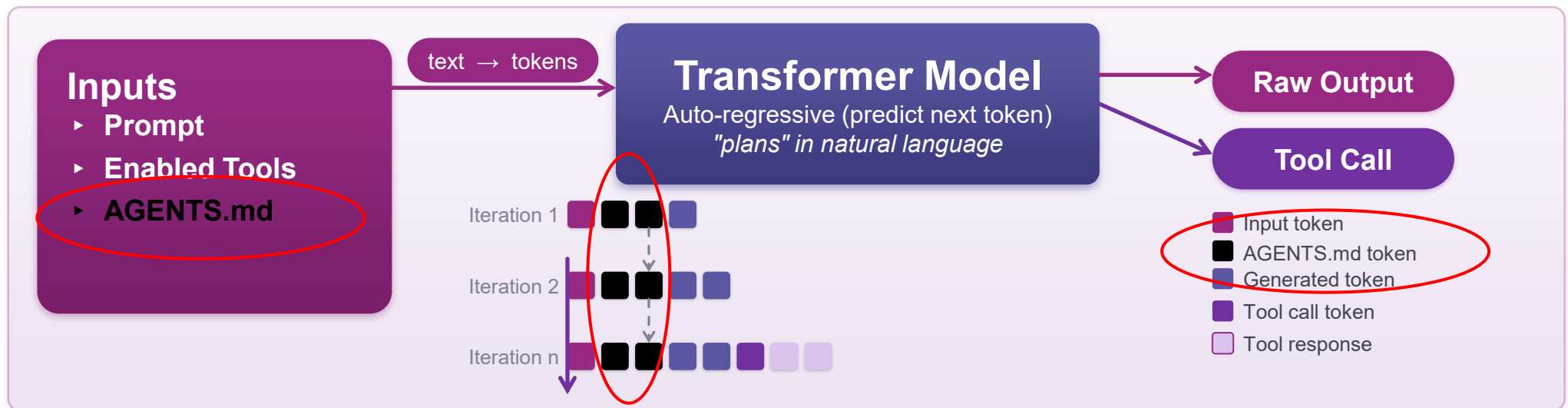
1. One objective » no truth, no goals.
2. Post-training adds **style**, not knowledge.
3. Hallucination = flat distribution + forced sampling.
4. Three categories, three mitigations.

AGENTS.md: the cross-tool standard

What instruction files are

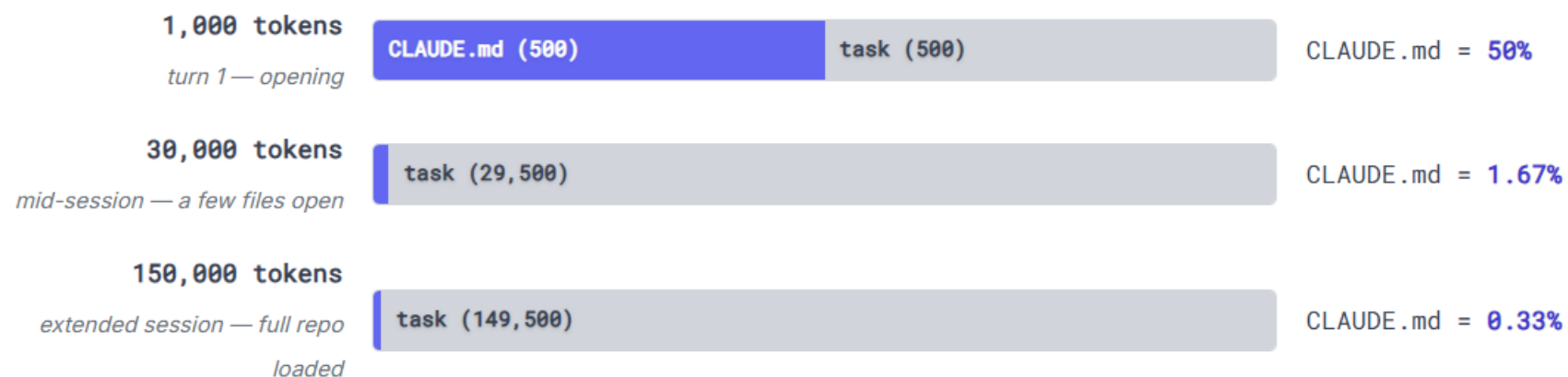
Every modern AI coding tool reads a project-root file before answering. AGENTS.md is the multi-vendor, foundation-governed standard. CLAUDE.md is Anthropic's variant.

Same kind of object across tools: a chunk of text injected as tokens into context. Understanding compliance failure requires L5 attention.



CLAUDE.md tokens compete for attention

Once the conversation starts, more tokens accumulate. The user's first prompt. The model's first response. A tool call to read a file. The file's contents. Another response. Another tool call.

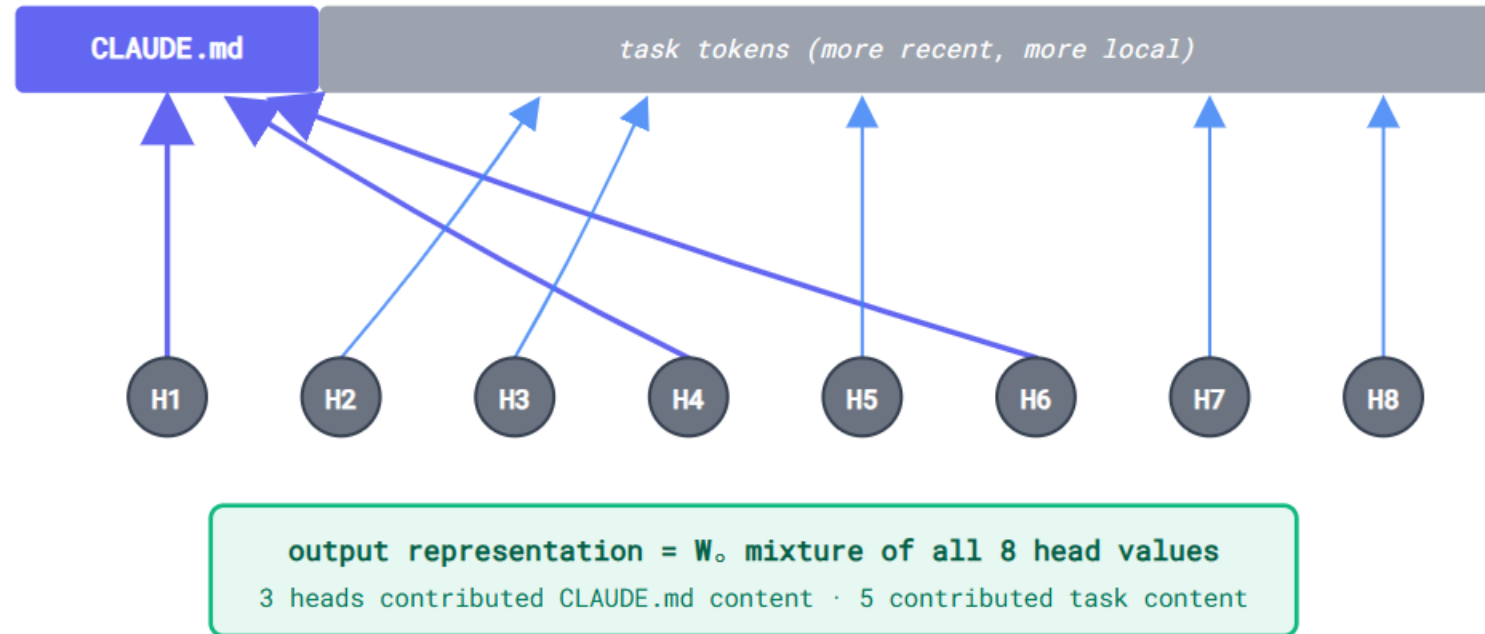


Same 500-token CLAUDE.md across all three rows — same file, same instructions, same disk content. Only the surrounding task tokens have grown. The shrinking indigo sliver is the proportional weight CLAUDE.md now carries when attention rolls out for the next-token decision.

Multi-head dilution

We **do not need to know exactly** which heads do what inside a production model to make the engineering point.

The intuition is enough: the model's final representation at each generation step is a mixture of many heads' outputs, combined by W_o . Heads that happen to route over CLAUDE.md tokens contribute their values to that mixture; heads that do not route there contribute other values.



CLAUDE.md is context, and context competes with other context.

Influenced, not commanded

- CLAUDE.md does not **command** the model. It **influences** the distribution.
- The model is influenced, not commanded.

KV and prompt caching: economics

	Uncached	Cached
Recomputed	All prefix K,V	Only new-turn tokens
Cost / turn	~30K forward passes	~100 forward passes
CLAUDE.md processed	Full prefill	K,V read from cache
Influence on out- put	Identical	Identical

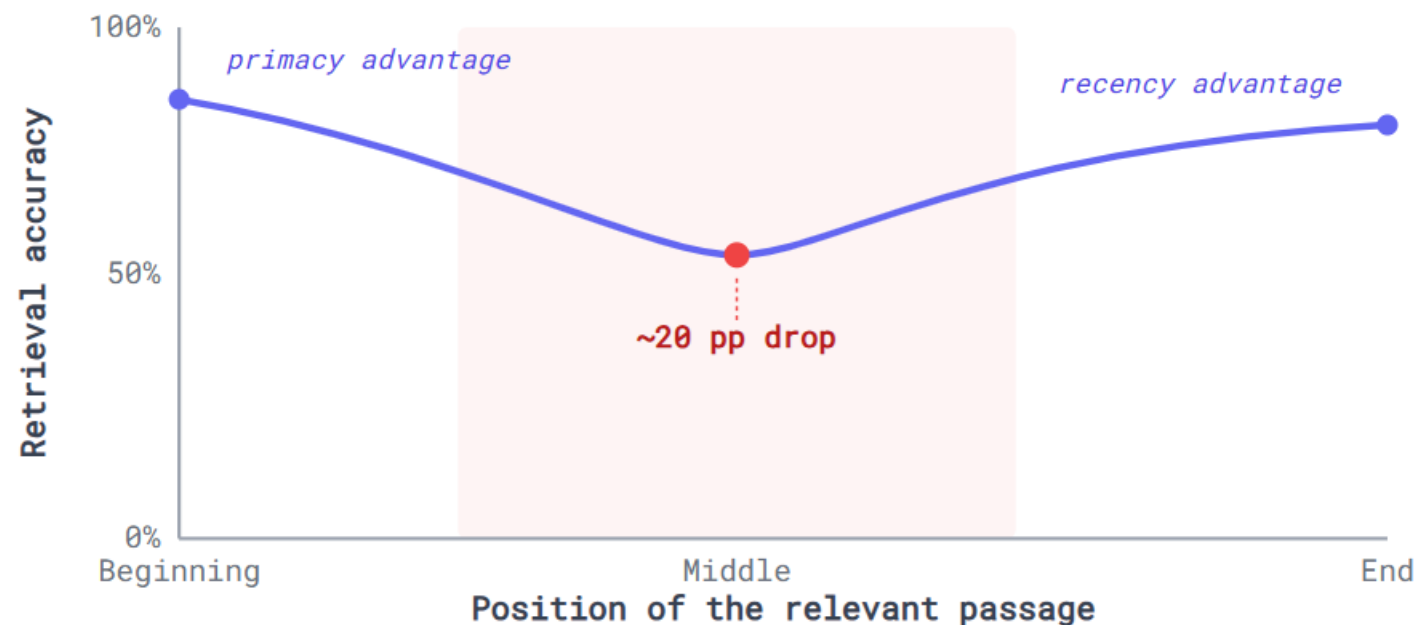
Prompt caching changes the **cost**, not the **attention**. The instruction at position 500 is just as easy to miss.

Lost in the middle: the U-shape

Take a question whose answer is in some long document.

Vary the position of the relevant passage within the document — beginning, middle, end.

Measure the model's accuracy as a function of where the passage sits.



Liu et al. (2023): retrieval accuracy is high at the start (primacy) and end (recency), and drops sharply in the middle — a U-shape rather than the flat line naive attention theory would predict.

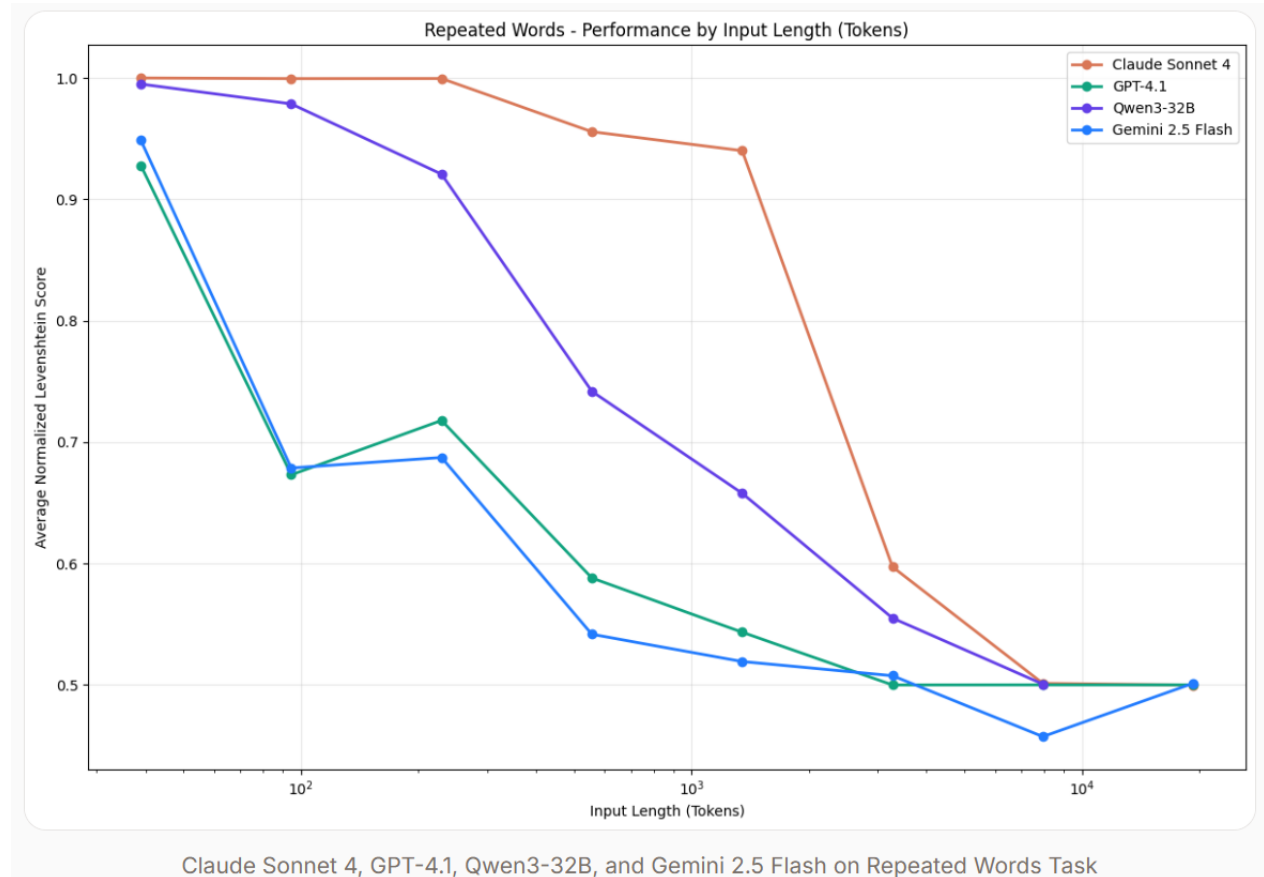
The Chroma “Context Rot” finding

As **the input grows**, the model’s effective **accuracy** on any specific piece of buried information **decays**.

The honest synthesis:

Positional effects on long-context retrieval are real, they are model-dependent, they are hard to predict exactly, and they get worse as contexts get longer.

[Chroma Research’s *Context Rot* report](#) (Hong, Troynikov & Huber, 2025)



Engineering rules for instruction files

Rule 1

Front-load critical rules. The first lines of CLAUDE.md have primacy advantage.

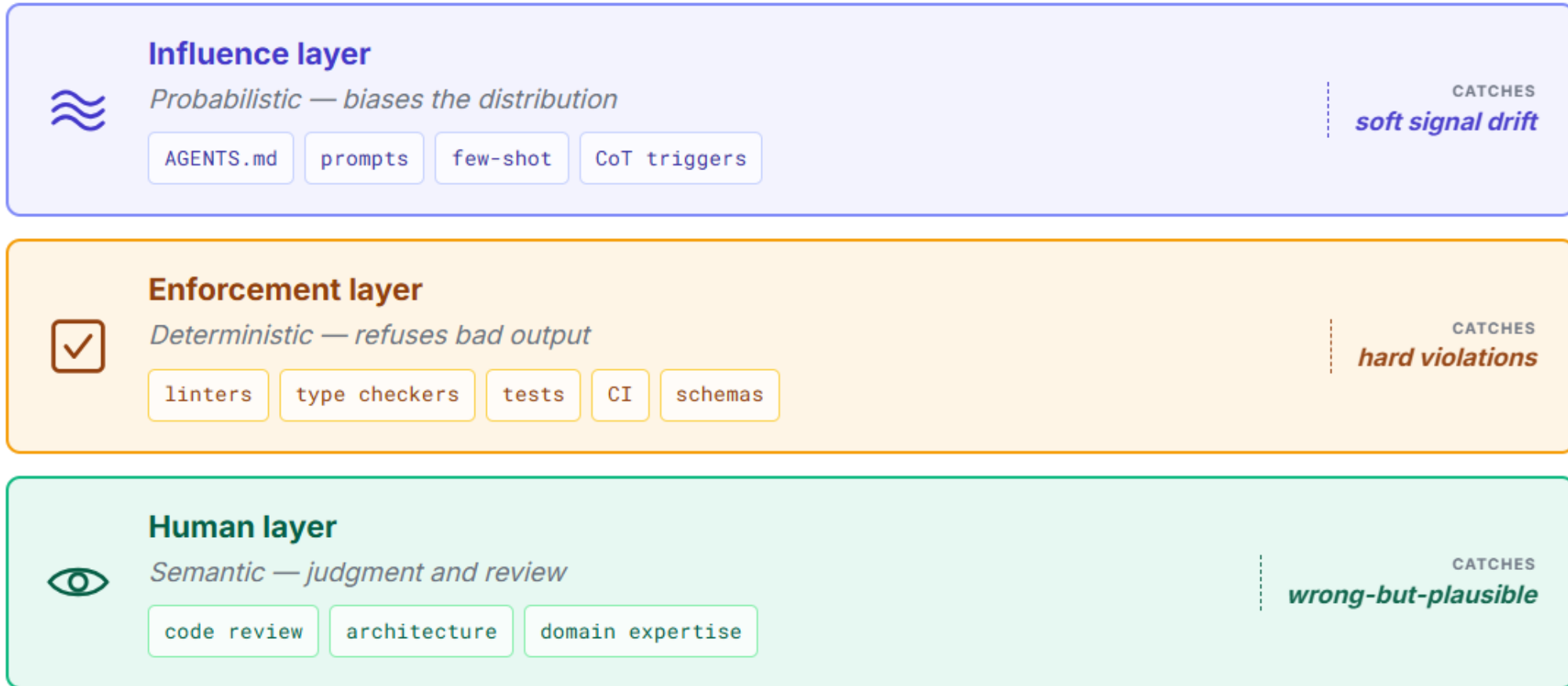
Rule 2

Longer $\neq$ more reliable. The middle decays. A 100-line file is more reliable line-for-line than a 600-line file.

Rule 3

Repeat critical rules near the end of the user prompt to buy recency advantage.

The three-layer model



Three parallel safety nets. Each layer catches what the other two miss — remove any one and the gaps line up.

Why all three layers matter

Layer	What it does	What it catches	Limitation
Influence	Biases the distribution	Soft signal drift	Probabilistic, position-dependent
Enforcement	Refuses bad output	Hard violations	Cannot judge semantics
Human	Reviews with expertise	Wrong-but-plausible	Does not scale

Teaching-repo note: enforcement is thin by design — the influence layer carries extra weight. In production, the balance inverts.

From consequences to controls

Block B begins

Block A named the failures. Block B names the controls. Every pattern ahead is a deliberate use of the L5 mechanism to shift the distribution toward the correct continuation.

Checkpoint: Block A synthesis

Recap

1. Pre-training = one objective, no truth.
2. Post-training = style, not knowledge.
3. Hallucination = flat distribution.
4. CLAUDE.md = influence, not command.
5. Three layers: influence + enforcement + human.

Engineering the distribution

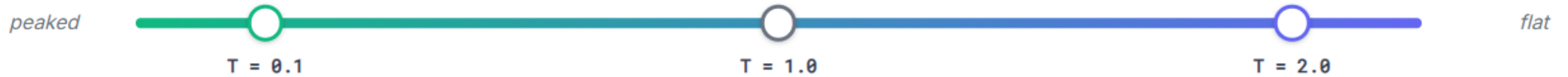
Temperature, Revisited as Engineering

$$p_i = \frac{e^{\ell_i/T}}{\sum_j e^{\ell_j/T}}$$

- $T < 1$: **sharpens** — winner takes more.
- $T > 1$: **flattens** — more variation.
- $T \rightarrow 0$: argmax (deterministic).

L5 softmax as baseline; temperature adds one parameter. This is the first engineering control.

Temperature, Revisited as Engineering



$T = 0.1$ – binary correctness

SHARP

$T = 1.0$ – graded correctness

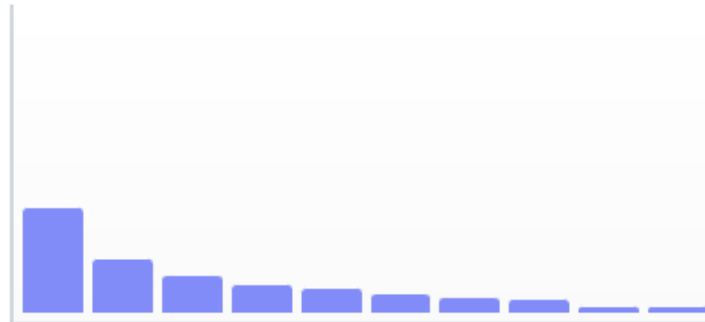
DEFAULT

$T = 2.0$ – variation desired

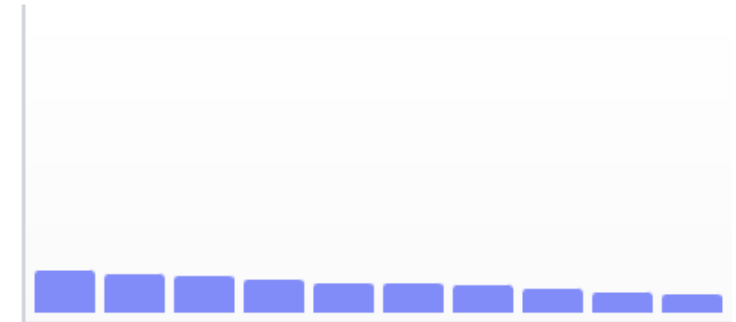
FLAT



Argmax dominates; effectively deterministic.



Top tokens dominate, but the seconds occasionally win.



Sampling explores the long tail; correctness traded for diversity.

Temperature is the dial that turns the same logits into different distributions. Low T concentrates mass on the argmax; high T flattens the mass across the vocabulary. The model and its parameters are unchanged — only how its softmax is shaped at sampling time.

Temperature decision tree

Task type	Recommended T	Reason
Binary correctness	≈ 0	Commit to argmax
Graded quality	0.3–0.7	Top tokens dominate
Variation / brainstorming	0.8–1.2	Explore the tail

Match temperature to the task. Code that must compile » low T . Test-case variations » higher T .

Seems already
(almost) outdated

Worked example: 10-token logit table

Token	Logit ℓ	$p(T=0.1)$	$p(T=1.0)$	$p(T=2.0)$
<BOS>	-1.2	≈ 0	0.003	0.024
wizard	0.5	≈ 0	0.016	0.056
glowing	1.8	≈ 0	0.058	0.107
potion	4.2	≈ 1.000	0.637	0.354
fire	3.1	1.7×10^{-5}	0.212	0.204
troll	0.9	≈ 0	0.024	0.068
smashes	1.1	≈ 0	0.029	0.075

First-token cascade

Greedy $\neq$ correct

At $T \approx 0$, the first generated token is nearly deterministic. But if that first token is wrong, every subsequent token conditions on the wrong start. Low temperature **amplifies** both correct and incorrect first tokens.

This is why $T=0$ code generation sometimes produces spectacularly wrong output with total confidence. CoT (§13) is the antidote.

What temperature does not solve

Limit 1

Temperature does not solve hallucination. A flat distribution from insufficient context stays flat at any T .

Limit 2

Temperature controls the **shape of sampling**, not the **quality of the distribution**. The patterns in the next four chapters shape the distribution itself.

Boltzmann: where the name comes from

$$\text{Softmax-}T: \quad p_i = \frac{e^{\ell_i/T}}{\sum_j e^{\ell_j/T}} \quad \text{logits } \ell_i, \text{ temperature } T$$

$$\text{Boltzmann:} \quad p_i = \frac{e^{-E_i/kT}}{Z} \quad \text{energy } E_i, \text{ Boltzmann const. } k$$

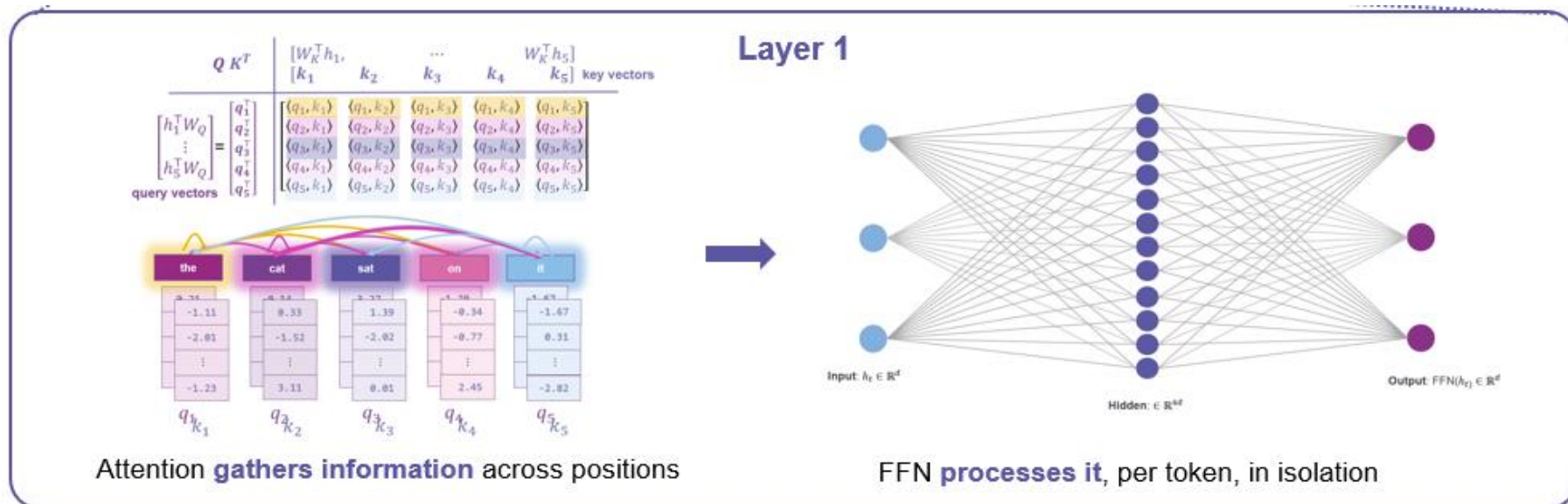
Identify $\ell_i \leftrightarrow -E_i/k$. Same math, same limits. The name is literal.

What is in-context learning?

Definition

The model adapts its behaviour based on examples in the prompt, **with no weight update**. Same parameters. Same forward pass. Only the context changed.

ICL is the substrate of few-shot prompting. The model does not change — the context changes, and attention routes differently.



GPT-3: the ICL headline

The headline

A 175B-parameter model performs tasks it was never explicitly trained on, simply by seeing a few examples in the prompt. Few-shot performance scales sharply with model size.

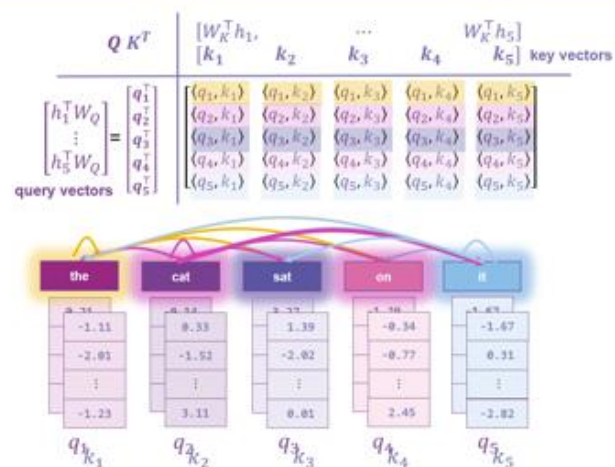
2020: the moment few-shot prompting became a serious engineering tool.

Brown et al. (2020), [*Language Models are Few-Shot Learners*](#), was the paper that introduced GPT-3 and named in-context learning as its headline capability.

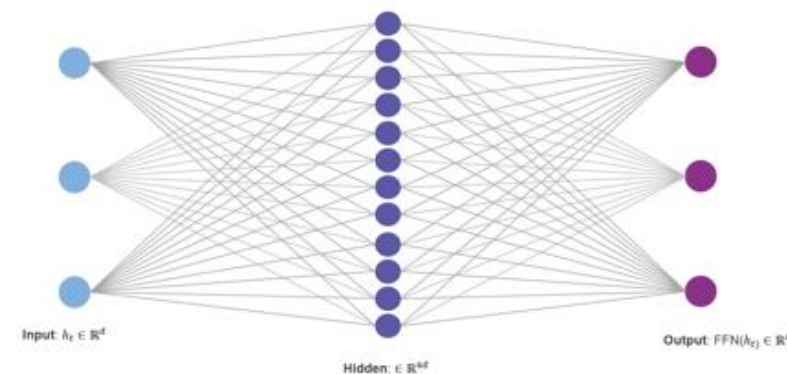
ICL mechanism in transformer vocabulary

Mechanism

Examples in the prompt create additional k, v pairs at every layer. The query from the new input attends to these examples. The mixed-in values shift the final representation and sharpen the unembedding logits toward the demonstrated pattern.



Attention **gathers information** across positions



FFN **processes it**, per token, in isolation

Induction heads: the routing pattern

Olsson et al. (2022), [*In-context Learning and Induction Heads*](#) (Anthropic), is the paper that gave ICL a specific mechanistic name.



The pattern

If you saw [A] [B] earlier, and now you see [A], predict [B]. Induction heads route from the query back to the matching earlier output.

Extended induction head examples

Few-shot prompts are **sequences of (input, output) pairs** followed by a new input.

The pattern is

**[in1][out1][in2][out2]...[in_k][out_k]
[in_query].**

To produce the right out-query, the model has to look at how previous inputs were followed by outputs. That is induction-head routing, on the granularity of multi-token inputs and outputs.

Instruction-style:

*"Write a function that handles HTTP errors. Use our project's standard logging. Raise our **ApiError** exception with the status code. Always include the request ID in the error message. Use snake_case. Keep functions under 30 lines."*

Few-shot style:

*"Here are two examples of how we handle HTTP errors in this codebase:
[example 1: existing handle_http_error function from your repo]
[example 2: another existing error handler from your repo]
Now write a similar handler for the new endpoint **[X]**."*

Format matters more than content

Key finding

Few-shot examples with **random labels** still improve performance over zero-shot. The format of input→output demonstrations transfers even when the content is wrong.

The labels are not primarily teaching the model what answer to give. The **format**, the **label space**, and the **task identity** are what transfer.

Logit trace: zero-shot baseline

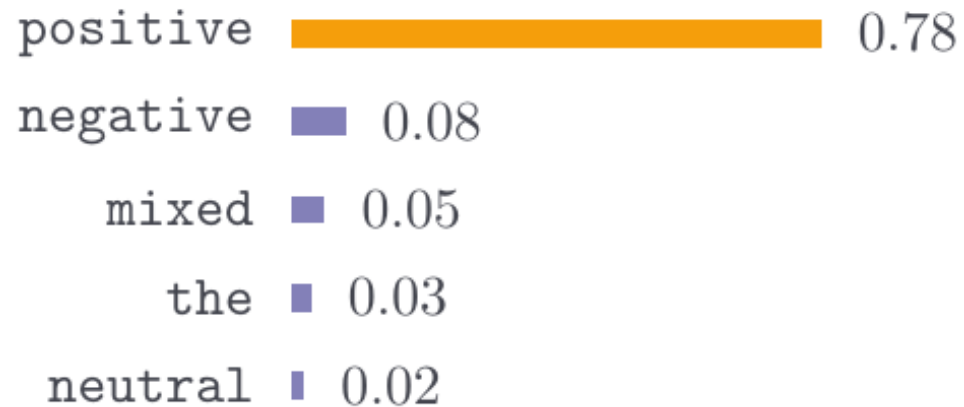
Zero-shot — no demonstrations

positive		0.18
negative		0.16
mixed		0.14
neutral		0.12
the		0.1
it		0.09
I		0.08

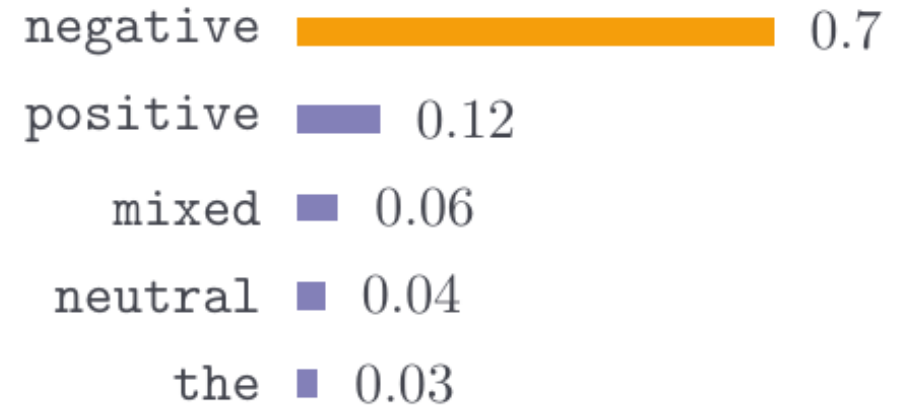
Flat — “positive” and “negative” compete with noise tokens. The model has no format cue for the task.

Logit trace: few-shot correct + random

Correct labels



Random labels



Both are sharper than zero-shot. Format transferred; correctness is secondary.

Format transfers, not content

- For project-specific conventions, **examples beat instructions**. Format transfers; content is secondary.

Checkpoint: ICL and induction heads

Recap

1. ICL = attention on demonstrations (no weight update).
2. Induction heads: $[A][B] \dots [A] \rightarrow [B]$.
3. Format $>$ content (Min et al. 2022).
4. This is why few-shot works — and why it works even with imperfect examples.

TECHNIQUE CARD: Zero-shot

Zero-shot	(baseline)
<i>What it is</i>	Task description + input, no demos, no role.
<i>Mechanism (L5)</i>	Relies on RLHF instruction-following + pretrained priors; distribution shaped by training statistics alone.
<i>SE example</i>	<code>normalize_email</code> (exercise Card 1): direct instruction, model applies general Python knowledge.
<i>Evidence cue</i>	A direct instruction with no examples, no role, no reasoning request.
<i>When / how it fails</i>	Fine for small, familiar, low-risk tasks; wrong tool when conventions or coverage matter.

Weak context » flatter distribution.

Few-shot engineering rules

Rule	Detail	Citation
$k = 2-4$ examples	Sweet spot for most tasks	Brown 2020
Order matters	Best example last (recency bias)	Lu et al. 2022
Format > content	Random labels still lift	Min 2022
Diminishing past ~ 8	Context cost outweighs lift	Empirical
Failure: conflicting	Wrong / off-distribution examples mislead	Min 2022

TECHNIQUE CARD: Few-shot

Few-shot

<i>What it is</i>	$k=2-4$ complete input→output examples; the model continues the demonstrated pattern. No weight updates.
<i>Mechanism (L5)</i>	Induction heads route from query to earlier outputs (Olsson 2022); mixed-in values sharpen logits.
<i>SE example</i>	Project-convention code gen: <code>get_user_by_id</code> (exercise Card 2). Paste 1–2 repo functions; model matches naming, style, types.
<i>Evidence cue</i>	2–4 demonstration pairs; “follow this pattern.”
<i>When / how it fails</i>	Project-specific style/format. Fails with wrong/conflicting examples; diminishing past ~ 8 .

Order matters — recency bias.

Format beats content

Match the input shape

- Make your examples the same **type** as the query — a Python function with a docstring, a 3-key JSON object. Induction heads look for prior occurrences of that shape; give them some.

Match the output shape

- Show the exact **format** you want — markdown table, JSON, bullet list — in the demonstrations. It transfers reliably.

Match the delimiters

- Use the **same separator** for the query as in your examples (newline, "Answer:", ---). The heads route across delimiter positions, so consistency keeps routing reliable.

Examples beat instructions for style

The principle

For project-specific conventions, paste 1–2 existing repo functions before asking for a new one. The model matches naming, SQLAlchemy usage, return style, and type hints **in one shot**.

Instructions alone rarely achieve this consistency. Forward-link to L7: skills = codified few-shot patterns.

Worked example: Stack Exchange (Act 1)

```
‘Write code to fetch monthly question  
counts from the Stack Exchange API’
```

Zero-shot

Hallucinated

Model in-
vents `/api/v2/
questions/monthly`.

Few-shot

Correct

2 examples of real API
calls » correct end-
point.

+CoT

Caught early

“First list the APIs” »
hallucination surfaces
in the plan, not the
code.

CoT: headline and mechanism

Chain-of-thought

Ask for intermediate reasoning » intermediate tokens become attended-to context » the model trades fixed depth for length. Accuracy improves 10–30 points on reasoning-heavy tasks.

Wei 2022: few-shot CoT. Kojima 2022: zero-shot CoT (“let’s think step by step”). Both work because intermediate tokens — the “thinking” — become context for later tokens.

*Wei et al. (2022), [*Chain-of-Thought Prompting Elicits Reasoning in Large Language Models*](#)

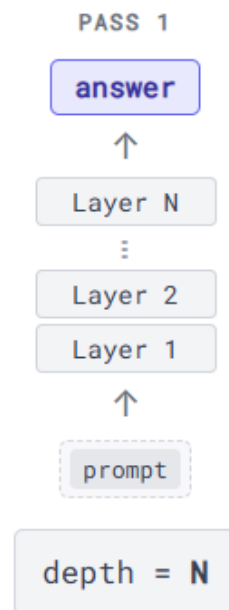
*Kojima et al. (2022), [*Large Language Models are Zero-Shot Reasoners*](#)

Depth for length: direct vs CoT

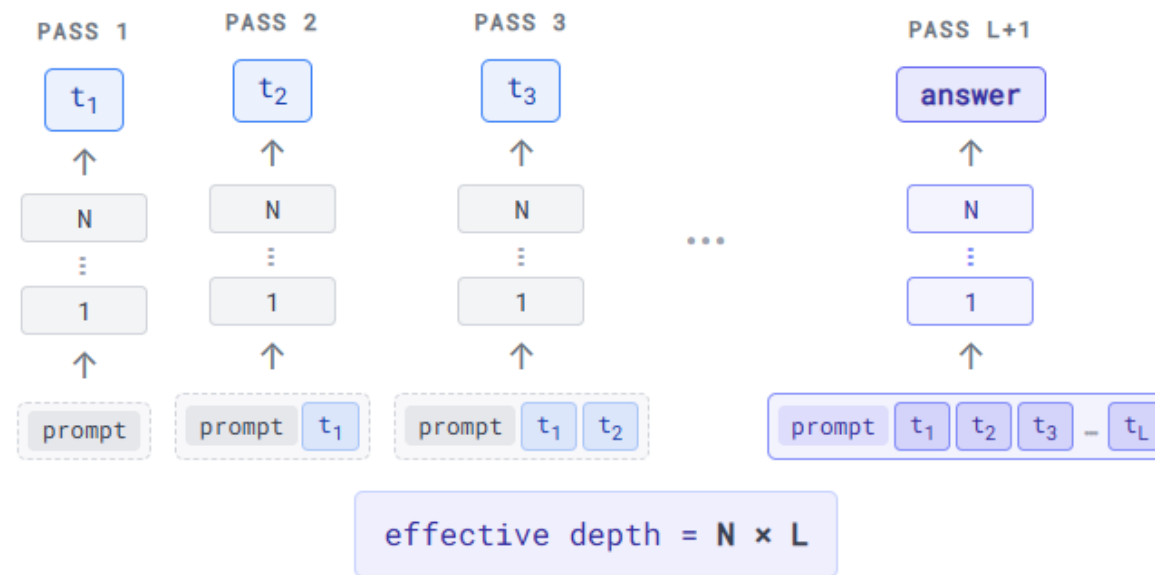
Scratchpad analogy

CoT is a scratchpad — the model writes notes to itself. Each note becomes context for the next step.

Direct generation — one forward pass



Chain-of-thought — L+1 forward passes; each reads the growing context



TECHNIQUE CARD: Chain-of-thought

Chain-of-thought

<i>What it is</i>	Ask for intermediate reasoning (“think step by step” / “plan, then code”).
<i>Mechanism (L5)</i>	Auto-regressive loop: intermediate tokens become context; trades depth for length (Wei 2022, Kojima 2022).
<i>SE example</i>	“First list the APIs you’ll call, then write the code” (§13.6) — catches API hallucinations before code.
<i>Evidence cue</i>	“think step by step,” “plan first,” “explain your reasoning.”
<i>When / how it fails</i>	Multi-step reasoning, planning. Fails on simple lookups (overhead); cascades wrong steps (Card 12).

Visible CoT is not an interpretability tool.

TECHNIQUE CARD: Chain-of-thought

Chain-of-thought

<i>What it is</i>	Ask for intermediate reasoning (“think step by step” / “plan, then code”).
<i>Mechanism (L5)</i>	Auto-regressive loop: intermediate tokens become context; trades depth for length (Wei 2022, Kojima 2022).
<i>SE example</i>	“First list the APIs you’ll call, then write the code” (§13.6) — catches API hallucinations before code.
<i>Evidence cue</i>	“think step by step,” “plan first,” “explain your reasoning.”
<i>When / how it fails</i>	Multi-step reasoning, planning. Fails on simple lookups (overhead); cascades wrong steps (Card 12).

Reasoning Models do have this internalized already!

Visible CoT is not an interpretability tool.

SE CoT: plan then code

SE pattern

“First list the external APIs you need. Then check: does each API actually exist? Then write the implementation.”

CoT catches the API hallucination **before** it becomes code. Connects Block A's failure (§4) to Block B's control.

When CoT hurts: four failure modes

Simple lookups

Pure overhead — don't ask for step-by-step on "what day is it?"

First-token cascade

Wrong intermediate step cascades — errors compound.

No knowledge

Reasoning can't create facts the model doesn't have.

Faithfulness

Visible reasoning $\neq$ internal process. Tests verify; CoT explains partially.

CoT expressivity: stated, not proved

Theoretical depth

Feng et al. 2023, Merrill & Sabharwal 2023: CoT extends transformer expressivity beyond fixed-depth computation. Stated, not proved here.

Reasoning models: productised CoT

Reasoning models (o1, DeepSeek-R1): productised test-time compute. The model does CoT internally — you do not need to prompt for it.

Over-prompting CoT on a reasoning model **degrades** quality. Prompt for goals, constraints, criteria.

Self-consistency: recipe and mechanism

Step	What happens	L5 mechanism
1	Sample N CoT paths at $T > 0$	Stochastic sampling explores different reasoning routes
2	Each path reaches a final answer	Auto-regressive generation completes
3	Majority vote across N answers	—
4	Return the mode	Distribution over paths is broad; over answers concentrates

*Wang et al. (2022), [Self-Consistency Improves Chain of Thought Reasoning in Language Models](#)

TECHNIQUE CARD: Self-consistency

Self-consistency

<i>What it is</i>	Sample N CoT paths at $T > 0$, return majority answer.
<i>Mechanism (L5)</i>	Variance reduction: distribution over paths is broad, over answers concentrates (Wang 2022).
<i>SE example</i>	Hard one-off evaluation task; eval pipeline where correctness $>$ latency. In production: usually via reasoning models.
<i>Evidence cue</i>	Multiple independent generations, voting/aggregation step.
<i>When / how it fails</i>	Costly ($N \times$ inference). Fails when the wrong answer is more common than the right one (mode $\neq$ correct).

Variance reduction and consensus signal

Variance reduction

Individual CoT paths are noisy. But the **final answers** concentrate. Self-consistency exploits the gap between path variance and answer variance.

Consensus as diagnostic

If 8/10 paths agree: confidence is high. If 5/5 split: the task is ambiguous — escalate to human review.

Checkpoint: prompting patterns

Recap

1. Zero-shot = baseline, relies on RLHF.
2. Few-shot = induction heads + format $>$ content.
3. CoT = auto-regressive scratchpad, depth for length.
4. Self-consistency = variance reduction over paths.
5. Each has failure modes. Next: beyond prompting, then the pattern library.

Structured outputs and constrained decoding

Constrained decoding

The decoder only samples from tokens that produce valid JSON/schema. An FSM-based mask at sampling time — **enforcement at the sampling layer**, not prompting.

Moves reliability from the influence layer (prompt) to the enforcement layer (sampling mask). Exercise Card 8 connects here.

When prompting is the wrong tool

Situation	Why prompting is wrong	Use instead
Deterministic logic	Business rules need code, not probability	Code / rule engine
Safety-critical output	Cannot tolerate probabilistic drift	Schema + validation
Verifiable correctness	Reasoning $\neq$ proof	Tests, formal checks
Credential handling	Tokens in context are visible	Never in prompt

Exercise Cards 5, 7, 10 test this judgment: “don’t prompt for this.”

Visible reasoning $\neq$ trustworthy reasoning

Faithfulness caveat

The model's visible reasoning is not necessarily a faithful record of its internal computation. Tests and code review verify output; CoT explains it only partially.

Model-class decision matrix

Model class	Strengths	Prompting style	Pitfall
Standard	General tasks	All five patterns apply	Over-prompting CoT
Reasoning	Multi-step	Goals + constraints	CoT prompts degrade
Coding agent	End-to-end	Task spec + tests	Micro-managing steps
Tool-using	External APIs	Tool definitions	Missing schema

Security boundary: prompt injection

OWASP LLM01

Untrusted input in the prompt can override instructions. The model treats all tokens equally — CLAUDE.md tokens and user-injected tokens look the same to attention.

The influence layer has no security boundary. Enforcement (input sanitisation, output validation) is the defence.

TECHNIQUE CARD: Role / persona priming

Role priming

<i>What it is</i>	Assign a role to set perspective/register (“Act as a senior backend engineer”).
<i>Mechanism (L5)</i>	RLHF taught the model that role tokens bias the distribution; persona shapes response register and coverage.
<i>SE example</i>	Code review (Card 3): “senior backend engineer + security-conscious reviewer” + rubric + structured output.
<i>Evidence cue</i>	“Act as...”, “You are...”, explicit role + rubric.
<i>When / how it fails</i>	Judgment tasks; pair with rubric + structured output. Fails as theatre if the role only changes tone.

Role alone is insufficient — rubric and structured output amplify it.

Pattern summary: six rows

Pattern	What you do	Mechanism (L5)	Citation
Temperature	Scale logits before softmax	Reshapes sampling distribution	L5 §14
Zero-shot	Task + input, no demos	RLHF priors	Brown 2020
Few-shot	k demo pairs	Induction heads	Olsson 2022
CoT	Ask for reasoning	Depth for length	Wei 2022
Self-consistency	N CoT paths, majority vote	Variance reduction	Wang 2022
Role priming	Assign role + rubric	RLHF persona bias	—

Decision tree: choosing a pattern

When the task is...	Use this pattern
Well-known & standard (<code>validate_email</code> , <code>summary</code> , <code>translation</code>)	Zero-shot. Temperature 0–0.7 depending on determinism vs. graded quality.
Project-specific in style or format (your schema, codebase pattern, house style)	Few-shot, 2–4 examples. Match the format strictly; put the best example last.
Requires intermediate reasoning (multi-step math, planning, bug tracing)	Zero-shot CoT (“Let’s think step by step”). If reasoning is wrong, escalate to few-shot CoT, 1–2 demos.
Accuracy-critical & worth the cost (evals, hard one-offs, high-stakes)	Self-consistency on top of CoT (~5–10× compute, $T > 0$). Otherwise skip.
Combine where useful	Few-shot CoT = standard for evals. Few-shot CoT + self-consistency = heavyweight option for hard problems.

How each pattern fails

Pattern	Failure mode	L5 mechanism	Mitigation
Temperature	First-token cascade	Greedy commits to wrong argmax	$T \approx 0.3$; CoT
Zero-shot	Flat dist. on novel tasks	Weak context	Add examples
Few-shot	Misleading examples	Induction copies wrong pattern	Curate examples
CoT	Cascading wrong steps	Wrong intermediate token	Verification
Self-consistency	Wrong mode	Wrong answer most common	Escalate to human
Role priming	Theatre	Tone without substance	Add rubric

No pattern is uniformly safe

- Better prompting: frequent » occasional.
- Better enforcement: occasional » caught.
- Better human review: caught » addressed.

How to diagnose a prompt: the 4-field lens

Field	Question
1. Name	Which strategy is being used?
2. Evidence	What phrase reveals it?
3. Appropriateness	Is this the right strategy for this task?
4. Anti-pattern	How does this strategy fail here?

Four questions, applied to any prompt. The exercise applies this lens to 12 real-world prompts across 11 SE task categories.

Exercise walkthrough: Card 3 (role priming)

Field	Card 3: Code review
Name	Role/persona priming + rubric + structured output
Evidence	“Act as a senior backend engineer,” explicit rubric, structured output format
Appropriate?	Yes — judgment task, perspective matters, rubric covers review criteria
Anti-pattern risk	If role is only tone (“be professional”) with no rubric = theatre

Three ingredients: **role** (who), **rubric** (what to check), **structured output** (how to format). Remove any one and the prompt degrades.

Exercise walkthrough: Card 12 (CoT overuse)

Field	Card 12: Simple lookup
Name	Chain-of-thought (overuse)
Evidence	“think step by step” on a simple lookup task
Appropriate?	No — task doesn’t need multi-step reasoning; CoT adds overhead and cascade risk
Anti-pattern	CoT on a task better served by zero-shot + validation. More technique $\neq$ better.

The deeper lesson: more technique $\neq$ better. Connect to the faithfulness caveat.

Supporting techniques overview

Rubric

What to check.
Pairs with role
priming.

Structured output

How to format.
Pairs with any
pattern.

Context grounding

Paste relevant
code/docs.
Pairs with few-
shot.

Validation

Tests, schemas,
CI. The enforce-
ment layer.

Prompt Strategy Diagnosis

Exercise

12 real-world prompts across 11 SE task categories. Apply the four-field diagnostic lens: name, evidence, appropriateness, anti-pattern.

Duration: 25 min (pairs) / 45–60 min (extended) **Modality:** pairs or small groups **Expected output:** Completed diagnosis for 4–6 cards + reflection

06 Prompt Strategy Diagnosis: 12 Cards