Prompting

Zero-shot · Few-shot · Chain-of-Thought · Self-Consistency · ReAct

The prompting revolution

Old paradigm

Collect labeled data
Train/fine-tune model
Deploy for one task

Weeks of work per task

GPT-3 (2020)

New paradigm

Write a prompt
Get an answer
No training needed

Minutes per task

Key insight: at sufficient scale, language models can perform tasks they were never explicitly trained on — just by describing the task in natural language (the "prompt")

What is a prompt? The text input given to an LLM to elicit a desired response. Includes: instructions, context, examples, constraints, output format specifications.

"Prompt engineering" = the art and science of crafting effective prompts

Zero-shot prompting

Zero-shot: give the model a task description and input — **no examples**.
The model relies entirely on knowledge from pre-training and instruction tuning.

Sentiment classification:

```
Classify the text as positive,
negative, or neutral.
Text: ‘‘The movie was okay.’’
Sentiment:
```

→ **Neutral**

Translation:

```
Translate to French:
‘‘The weather is beautiful
today.’’
```

→ **“Le temps est magnifique aujourd’hui.”**

Strengths:

No example curation needed
Task-agnostic, simplest approach
Works well on common tasks
Modern instruction-tuned models excel

Limitations:

Struggles with complex reasoning
No way to “show” desired format
Smaller models underperform
Domain-specific tasks may fail

First demonstrated: GPT-2 (2019) · Dramatically improved: GPT-3 (2020)
Now standard in instruction-tuned models

Few-shot prompting (in-context learning)

Few-shot: provide examples of (input, output) pairs in the prompt. The model infers the pattern and applies it — **no gradient updates, no training**.

```
Text: ‘‘This movie was absolutely fantastic!’’ -> Positive
Text: ‘‘I hated every minute of it.’’ -> Negative
Text: ‘‘The food was okay, nothing special.’’ -> Neutral

Text: ‘‘I can’t believe how great this product is!’’ ->
```

→ **Positive**

GPT-3 findings:

Biggest jump: 0-shot → 1-shot
2–5 examples is the sweet spot
Diminishing returns beyond 5
Few-shot scales faster with model size than zero-shot

Why it works:

Model “learns” task format from examples in the prompt context
No weights are updated
Think of it as meta-learning: the model learned *how to learn*

Brown et al. (2020), “Language Models are Few-Shot Learners” — NeurIPS 2020

Few-shot is surprisingly fragile

Zhao et al. (2021): few-shot accuracy can vary from **near-chance to near-SOTA** depending on example choice, order, and format — same model, same task!

Majority label

If 3/4 examples are "Positive", model is biased toward predicting "Positive" for all inputs

Recency bias

Model favors the label of the *last* example in the prompt.

Order matters a lot!

Common token

Model prefers tokens that are common in pre-training data.

"Yes" > "Affirmative"

Fix — Contextual Calibration: feed a content-free input ("N/A"), measure the bias, then calibrate the output distribution to be uniform. Up to **+30% accuracy improvement**.

Best practices: balance label distribution · test different orderings
choose diverse, representative examples · use examples similar to the target input

Zhao et al. (2021), "Calibrate Before Use" — ICML 2021

Chain-of-Thought prompting (Wei et al., 2022)

Standard prompting:

```
Q: Roger has 5 tennis balls.
He buys 2 cans of 3 balls
each. How many now?
A: 11.

Q: The cafeteria had 23
apples. Used 20, bought 6
more. How many?
A:
```

Chain-of-Thought:

```
Q: Roger has 5 tennis balls.
He buys 2 cans of 3 each.
A: 2 cans of 3 = 6 balls.
5 + 6 = 11. Answer: 11.

Q: Cafeteria had 23 apples.
Used 20, bought 6 more.
A:
```

GSM8K results (PaLM 540B): standard prompting 18% → CoT **57%**
Using just 8 chain-of-thought exemplars achieved state-of-the-art accuracy

Emergent ability: CoT helps only in large models (∼100B+ params).
In smaller models, chain-of-thought can actually **hurt** performance.

Zero-shot CoT: "Let's think step by step"

"Let's think step by step."

Kojima et al. (2022) — that's it, that's the whole technique

```
Q: A juggler can juggle 16 balls. Half are golf balls, and half
of the golf balls are blue. How many blue golf balls?
A: Let's think step by step.

There are 16 balls total.
Half are golf balls: 16 / 2 = 8 golf balls.
Half of the golf balls are blue: 8 / 2 = 4.
The answer is 4.
```

Few-shot CoT vs. Zero-shot CoT

	Few-shot CoT	Zero-shot CoT
Examples needed	Yes (with reasoning)	**None**
Trigger	Exemplar chains	"Let's think step by step"
Performance	Stronger	Slightly weaker
Effort	Craft exemplars per task	Single phrase for all tasks

Kojima et al. (2022), "Large Language Models are Zero-Shot Reasoners" — NeurIPS 2022

Why does Chain-of-Thought work?

Decomposition

Breaks complex problems into manageable sub-steps

"How many total?" becomes:
"Step 1: count A"
"Step 2: count B"
"Step 3: add"

Scratchpad

Generated text serves as external working memory

Intermediate results are written down, not lost in hidden activations

Arithmetic grounding

Each computation is explicit: "5 + 6 = 11" not just "11"

Reduces cascading errors from skipped steps

Interpretability

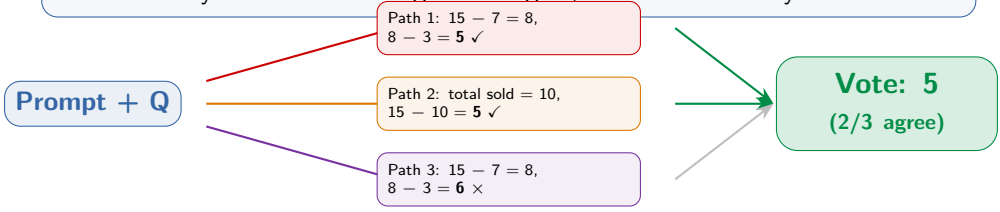
The reasoning chain is visible and debuggable

You can see *where* the model goes wrong and fix the prompt

Deeper insight: transformers compute a fixed amount per token (constant depth). CoT gives them more "compute steps" — each reasoning token is an extra forward pass.

Self-Consistency (Wang et al., 2022)

Idea: sample **multiple** reasoning paths,
then take a **majority vote** on the answer.
If many different reasoning chains agree, the answer is likely correct.

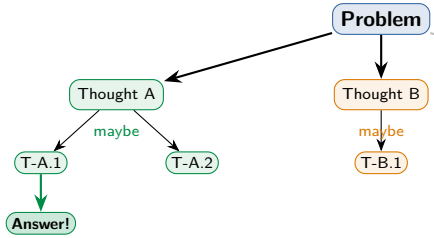


Algorithm: 1. Sample N reasoning paths (temperature > 0)
2. Extract answer from each 3. Return mode$(\{a_1, \ldots, a_N\})$

PaLM-540B results: GSM8K +17.9% · SVAMP +11.0% · AQuA +12.2%
No extra training. Just sample more and vote. Cost scales linearly with N (typically 5–40).

Tree of Thoughts (Yao et al., 2023)

Idea: generalize CoT from a **single chain** to a **tree** of reasoning steps.
Explore multiple branches, self-evaluate, and **backtrack** from dead ends.



Components:
1. Thought generator
2. State evaluator
   (sure / maybe / impossible)
3. Search: **BFS** or **DFS**
4. Backtracking

Game of 24 (GPT-4):
Standard CoT: 4% success
Tree of Thoughts: **74%** success

Cost:
Many more LLM calls than CoT
Best for problems with clear evaluation criteria

ReAct: Reasoning + Acting (Yao et al., 2022)

Idea: interleave **reasoning** (Thought) with **actions** (Act) that query external tools. Observations (Obs) from the environment feed back into the next reasoning step.

Q: What is the elevation range for the eastern sector of the Colorado orogeny?

Thought 1: I need to find what area the eastern sector extends into.
Action 1: Search[Colorado orogeny]
Obs 1: . . . The eastern sector extends into the **High Plains**.
Thought 2: Now I need the elevation range of the High Plains.
Action 2: Search[High Plains]
Obs 2: rise from around 1,800 to 7,000 ft **Action 3:** Finish[**1,800 to 7,000 ft**]

Foundation of modern AI agents: LangChain, OpenAI function calling, Claude tool use all follow this Thought → Action → Observation pattern

ReAct: why reasoning AND acting both matter

Reason only (CoT)

Thinks internally but can't verify facts

Prone to hallucination
Error propagation

"The capital of Australia is Sydney"

Act only

Calls tools blindly without strategic planning

No error recovery
Repetitive actions

Searches the same thing 5 times

ReAct (both)

Plans *then* acts, verifies *then* continues

Grounded in facts
Strategic planning

Reasons about what to search *and* what the result means

Results: ReAct outperforms both CoT-only and Act-only on HotpotQA, FEVER
ALFWorld (+34% over imitation learning), WebShop (+10%)

Available actions in practice: Search · Calculator · Code interpreter
Database query · API calls · File read/write · Web browse

Yao et al. (2022), "ReAct: Synergizing Reasoning and Acting in Language Models" — ICLR 2023

Prompt engineering: best practices

Be specific

Bad: "Make it better"

Good: "Improve readability by using shorter sentences and

Use delimiters

Separate instructions from content using ‘‘‘, ---, or XML tags

Prevents prompt injection and

Specify output format

"Return as JSON with keys: name, age, occupation"

"Answer in 2–3 sentences"

"Use a markdown table"

Role prompting

"You are a senior Python developer with 15 years of experience. Review this code for bugs and style issues."

Common pitfalls:

Ambiguity (vague instructions) · Overloading (too many tasks in one prompt)
Negative instructions ("don't do X" weaker than "do Y instead")
Missing context · Not iterating · Ignoring prompt injection risks

Prompt structure: Context (background) + Objective (what to do) + Constraints (limits) + Output (format)

System prompts and temperature

System prompt:

```
You are a helpful math
tutor for high school
students. You explain
concepts step by step
using simple language.
Always show your work.
```

Sets behavior, persona, and constraints for the entire session. Processed before user messages.

Temperature (T)

Controls randomness of output.

$$p_i = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)}$$

$T = 0$	Deterministic
$T = 0.3$	Focused, factual
$T = 0.7$	Balanced
$T = 1.0$	Creative
$T > 1$	Very random

Temperature guidelines:

$T \approx 0$

Classification
Code generation
Fact retrieval
Math

$T \approx 0.5$

General Q&A
Summarization
Analysis
Explanations

$T \approx 0.9$

Creative writing
Brainstorming
Poetry
Self-consistency sampling

Rule of thumb: lower temperature = more reliable, higher = more diverse.
For self-consistency, use $T \in [0.5, 1.0]$ to get diverse reasoning paths.

Structured output prompting

Problem: LLM outputs are free-form text.
Software pipelines need structured data.
Solution: guide the model to produce JSON, XML, or other parseable formats.

JSON prompting:

```
Extract info as JSON:
{''name'': str, ''age'': int}
''Dr. Sarah Chen, 42,''
''is a neurologist.''
→ {''name'': ''Sarah Chen'',
  ''age'': 42}
```

XML tags:

```
<instructions>
 Summarize in 3 bullets.
</instructions>
<article>...</article>
```

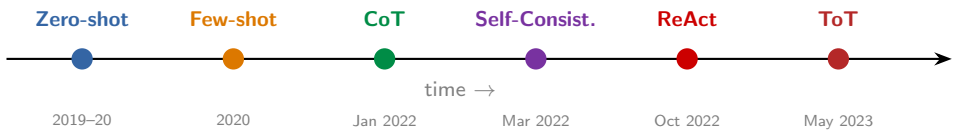
Especially effective with Claude.
Tags structure both input and output.

Ensuring compliance:

Method	Guarantee level
Explicit instructions in prompt	Mostly works, not guaranteed
Few-shot examples of format	Stronger, still not 100%

Caveat: strict format constraints can **degrade reasoning** quality.
For reasoning-heavy tasks: let the model
think freely first, then format in a second call.

The evolution of prompting techniques



Three generations of prompting:

Direct
Zero-shot
Few-shot
Just describe the task
or show examples

Reasoning
CoT, Self-Consistency
Tree of Thoughts
Elicit step-by-step
thinking from the model

Agentic
ReAct, tool use
Modern agents
Combine reasoning
with external actions

Prompting techniques compared

Technique	Examples needed	LLM calls	Best for	Key paper
Zero-shot	None	1	Simple, common tasks	Radford 2019
Few-shot	2–5 demos	1	Format-sensitive tasks	Brown 2020
Chain-of-Thought	Reasoning chains	1	Math, logic, multi-step	Wei 2022
Zero-shot CoT	None ("step by step")	1	Quick reasoning boost	Kojima 2022
Self-Consistency	Reasoning chains	N (5–40)	Verifiable answers	Wang 2022
Tree of Thoughts	Task-specific	Many	Search/planning problems	Yao 2023
ReAct	1–2 demos	Multiple	Knowledge-grounded QA	Yao 2022

Decision guide:

Simple task? $\rightarrow$ Zero-shot Need specific format? $\rightarrow$ Few-shot
Reasoning required? $\rightarrow$ CoT High stakes? $\rightarrow$ Self-Consistency
Need facts? $\rightarrow$ ReAct + tools Complex search? $\rightarrow$ Tree of Thoughts

These techniques are **composable**: you can use few-shot CoT with self-consistency and tool use simultaneously

Practical: designing a prompt step by step

Task: build a prompt that extracts key info from medical notes

Step 1: Context + Role

```
You are a medical data
extraction specialist.
Extract structured info
from clinical notes.
```

Step 2: Output format

```
Return JSON with keys:
patient_name, age,
diagnosis, medications,
follow_up_date
```

Step 3: Constraints

```
If a field is not mentioned,
use null. Do not infer or
guess. Use exact names and
dosages from the text.
```

Step 4: Few-shot example

```
Note: ‘‘Mr. Smith, 67,
diagnosed with HTN...’’
-> {‘‘patient_name’’:
‘‘Smith’’, ‘‘age’’: 67, ...}
```

Assemble: system prompt (role) + format specification + constraints + 1–2 examples

Then iterate: test on edge cases, refine instructions, add constraints as needed

Key principle: prompt engineering is **iterative**. Start simple, test, add detail. The best prompt is the simplest one that reliably produces correct output.

Questions?

Next: Hallucination & Grounding