Core Prompt Patterns

Prompt patterns are reusable strategies for structuring your instructions to an LLM. Choosing the right pattern is the single highest-leverage decision you make — it determines whether the model reasons carefully or guesses wildly.

Pattern	Description	Best For
Zero-shot	No examples, just instruction	Simple tasks, capable models
One-shot	Single example provided	Format demonstration
Few-shot	2-6 examples provided	Complex formatting, edge cases
Chain-of-Thought (CoT)	"Think step by step"	Math, logic, reasoning
Tree-of-Thought (ToT)	Explore multiple reasoning paths	Complex problem solving
Self-consistency	Sample multiple CoT, majority vote	High-stakes reasoning
ReAct	Reason + Act in alternating steps	Tool-using agents
Skeleton-of-Thought	Outline first, then expand	Long-form generation

Zero-Shot vs Few-Shot Decision

The most common mistake in prompt engineering is providing too many examples when none are needed, or providing none when the model clearly needs guidance. This decision tree helps you choose the right approach for your task.

```
Is the task straightforward and well-defined?
  YES -> Zero-shot (add "Think step by step" if reasoning needed)
  NO  -> Does the output need a specific format?
           YES -> Few-shot (2-3 examples showing format)
           NO  -> Does it involve nuanced judgment?
                    YES -> Few-shot with edge cases
                    NO  -> One-shot with clear instructions
```

Chain-of-Thought Templates

Chain-of-thought prompting forces the model to show its reasoning before giving an answer. This dramatically improves accuracy on math, logic, and multi-step reasoning tasks — often turning a wrong answer into a correct one.

Basic CoT

```
Q: {question}
A: Let's think step by step.
```

Structured CoT

```
Solve the following problem. Show your reasoning in these steps:
1. Identify the key information
2. Determine the approach
3. Execute step by step
4. Verify the answer

Problem: {problem}
```

Self-Consistency (with CoT)

```
Generate 5 independent solutions to this problem.
For each, think step by step.
Then compare all answers and select the most common result.
```

System Prompt Architecture

The system prompt is your contract with the model — it defines who it is, what it does, and how it behaves. A well-structured system prompt eliminates 80% of output quality issues before they happen.

Anatomy of an Effective System Prompt

```
You are {role} with expertise in {domain}.

## Task
{What to do}

## Constraints
- {Constraint 1}
- {Constraint 2}

## Output Format
{Exact format specification}

## Examples
Input: {example_input}
Output: {example_output}
```

System Prompt Checklist

Element	Purpose	Required?
Role definition	Sets persona and expertise	Yes
Task description	Core instruction	Yes
Output format	Controls structure	Yes
Constraints	Boundaries and rules	Recommended

Element	Purpose	Required?
Examples	Demonstrates expected behavior	Recommended
Error handling	What to do with bad input	Optional
Tone/style	Voice and register	Optional

Temperature and Sampling Parameters

Temperature and sampling parameters control how the model selects the next token. Getting these wrong means either robotic, repetitive outputs (too low) or hallucinated, incoherent ones (too high). Most production issues trace back to incorrect parameter settings.

Parameter	Range	Low Value Effect	High Value Effect
`temperature`	0.0 - 2.0	Deterministic, focused	Creative, diverse
`top_p`	0.0 - 1.0	Narrow token selection	Wider token selection
`top_k`	1 - N	Very focused	More diverse
`frequency_penalty`	-2.0 - 2.0	Allows repetition	Penalizes repetition
`presence_penalty`	-2.0 - 2.0	Allows topic revisits	Encourages new topics

Recommended Settings by Task

Task	Temperature	top_p	Notes
Code generation	0.0 - 0.2	0.95	Deterministic preferred
Data extraction	0.0	1.0	Exact outputs needed
Creative writing	0.7 - 1.0	0.95	Diversity matters
Conversation	0.5 - 0.7	0.9	Balanced
Brainstorming	0.9 - 1.2	1.0	Maximum creativity
Classification	0.0	1.0	Consistency needed
Translation	0.1 - 0.3	0.95	Accuracy first

Rule of thumb: Set temperature OR top_p, not both. Use temperature for most cases.

Prompt Templates by Use Case

These are battle-tested templates for the most common LLM tasks. Copy, paste, and customize — they handle the structural patterns so you can focus on your specific domain requirements.

Classification

```
Classify the following text into exactly one category:
Categories: {list}

Text: "{input}"

Respond with only the category name, nothing else.
```

Extraction

```
Extract the following fields from the text below.
Return as JSON. Use null for missing fields.

Fields: {field_list}

Text: "{input}"
```

Summarization

```
Summarize the following text in {N} bullet points.
Each bullet should be one concise sentence.
Focus on: {key_aspects}

Text: "{input}"
```

Code Generation

```
Write a {language} function that {description}.

Requirements:
- {req1}
- {req2}

Input: {input_type}
Output: {output_type}

Include error handling and type hints.
```

Output Formatting Tricks

Unstructured LLM output is the #1 source of downstream bugs in production systems. These formatting techniques ensure your model returns parseable, consistent output every time.

Technique	Prompt Fragment	Use Case
JSON mode	"Respond in valid JSON only"	Structured data
XML tags	"Wrap answer in `<answer>` tags"	Easy parsing
Markdown table	"Format as a markdown table"	Comparisons

Technique	Prompt Fragment	Use Case
Numbered list	"List exactly N items"	Controlled output
Delimiter	"Separate sections with ---"	Multi-part output
Schema	Provide JSON Schema	Strict structure

Advanced Techniques

When a single prompt can't handle the full complexity of your task, these techniques let you decompose, delegate, and iterate. Prompt chaining alone can turn a 60% accuracy task into a 95% one.

Prompt Chaining

```
Step 1: Extract key entities -> {entities}
Step 2: For each entity, gather context -> {context}
Step 3: Synthesize into final answer using {entities} + {context}
```

Meta-Prompting

```
You are a prompt engineer. Write the optimal prompt for an LLM
to accomplish this task: {task_description}

The prompt should include:
- Clear role definition
- Step-by-step instructions
- Output format specification
- 2-3 examples
```

Negative Prompting

```
Do NOT:
- Include disclaimers or warnings
- Use bullet points
- Exceed 100 words
- Mention that you're an AI
```

Token Optimization

Every token costs money and adds latency. In production systems handling millions of requests, a 30% reduction in prompt tokens can save thousands of dollars per month without sacrificing quality.

Strategy	Savings	Trade-off
Abbreviate instructions	20-40%	May reduce clarity
Remove examples	30-50%	Lower output quality

Strategy	Savings	Trade-off
Use shorter delimiters	5-10%	Minimal
Compress system prompt	15-30%	Harder to maintain
Reference by name	10-20%	Needs model knowledge

Common Pitfalls

These are the mistakes every practitioner makes at least once. Bookmark this table — when your LLM output looks wrong, the fix is almost always here.

Pitfall	Problem	Fix
Vague instructions	Inconsistent outputs	Be specific about format and content
Too many constraints	Model ignores some	Prioritize, reduce to essentials
No output format	Unparseable responses	Always specify expected format
Conflicting instructions	Confused model	Review for contradictions
Over-long prompts	Token waste, diluted focus	Trim to essentials
No error handling	Crashes on edge cases	Add "If unsure, respond with..."
Assuming model knowledge	Incorrect outputs	Provide context, don't assume
Temperature too high for factual tasks	Hallucinations	Use 0.0-0.2 for factual work
Few-shot with bad examples	Model learns wrong patterns	Verify examples are correct
Ignoring token limits	Truncated output	Calculate: input + expected output < max

Evaluation Quick Checks

Before shipping any prompt to production, run these five checks. A prompt that works once in a playground can fail spectacularly at scale — these checks catch the gaps.

Check	Method
Consistency	Run same prompt 5x, compare outputs
Edge cases	Test with empty, long, adversarial input
Format compliance	Parse output programmatically
Factual accuracy	Spot-check claims
Instruction following	Does output match all constraints?

Quick Reference: Prompt Length Guidelines

Context windows are not infinite — and models don't attend equally to all content. Understanding your token budget prevents truncated outputs and ensures the model actually reads your most important instructions.

Model Context	Effective Prompt Budget	Leave for Output
4K tokens	2K-3K	1K-2K
8K tokens	5K-6K	2K-3K
32K tokens	20K-25K	7K-12K
128K tokens	80K-100K	28K-48K
200K tokens	120K-160K	40K-80K

Remember: Longer context windows do not mean the model attends equally to all content. Place critical instructions at the beginning and end of the prompt.