A BEGINNER'S FIELD GUIDE

10 CHAPTERS

Prompt Engineering for New Software Engineers

How to write prompts that produce code, reviews, documentation and images you can actually use, explained with everyday engineering examples.

§1 Foundations

NO.	CHAPTER	WHAT IT COVERS	PAGE
01	**What Is a Prompt?**	What a prompt contains, why specific prompts beat vague ones, and the four building blocks.	p. 2
02	**What Is Prompt Engineering?**	Designing, testing and refining prompts in a loop that works much like debugging.	p. 3
03	**Best Practices for Writing Prompts**	Clarity, context, specificity and role, each shown as a weak prompt rewritten into a strong one.	p. 4
04	**Tools for Prompt Engineering**	The main kinds of tools, features worth looking for, and treating prompts like code.	p. 5

§2 Prompting techniques for text

NO.	CHAPTER	WHAT IT COVERS	PAGE
05	**Text Prompt Techniques**	Six techniques for reliable text output, the feedback loop, and zero-shot vs. few-shot prompting.	p. 6
06	**The Interview Pattern**	Let the model ask clarifying questions first, shown with a database-schema design.	p. 7
07	**The Chain of Thought Approach**	Teach a line of reasoning with a worked example, and when reasoning models change the advice.	p. 8
08	**The Tree of Thought Approach**	Explore, score and prune several options, shown with a caching decision.	p. 9

§3 Prompting for images

NO.	CHAPTER	WHAT IT COVERS	PAGE
09	**Image Prompt Techniques**	Style, quality terms, repetition, weights and negative prompts, and which tools support them.	p. 10

§4 Working safely

NO.	CHAPTER	WHAT IT COVERS	PAGE
10	**Using AI Safely as an Engineer**	Verifying generated code, protecting secrets, system prompts, structured output and prompt injection.	p. 11

How to use this guide: each chapter is a one-page visual summary. Read them in order, or select a chapter above to jump to it.

Chapters 1–4 cover the foundations, 5–8 techniques for text and code, 9 prompting for images, and 10 working safely with AI.

What Is a Prompt?

The prompt is the part of a request you can change every time you call a model. This chapter covers what that text can contain and how to structure it.

§1 Defining a prompt

DEFINITION

Everything the model receives

A prompt is the full input for a request: instructions, background, data and examples. The model combines it with what it learned in training and with any system prompt the app adds.

SINGLE REQUEST

One instruction, one result

For example: `Write a unit test for the parse_date function below.` Useful when the task is small and well defined.

CONVERSATION

A series of turns

The app resends the earlier messages on every turn (the model keeps no memory of its own), so a follow-up such as "now handle time zones" refines the last answer. Very long chats can push early details out of the context window.

§2 What a prompt can contain

QUESTION

A direct ask

`Why does this loop never exit?`

CONTEXT

Background

Language and version, framework, the error message, what you already tried.

EXAMPLES

The shape of the answer

A sample of the output format you expect, such as one JSON object or one test case.

PARTIAL INPUT

Something to continue

A function signature or half-written query to finish. IDE autocomplete continues it directly; in a chat, ask explicitly for it to be completed.

§3 Vague prompts vs. specific prompts

A vague prompt leaves the model to guess the language, constraints and format. The guesses are often reasonable, but rarely what you needed.

GOAL	VAGUE PROMPT	SPECIFIC PROMPT
Write a function	**"Write a sort function."**	"Write a Python function `sort_users(users)` that sorts dicts by `last_login`, newest first. Users without `last_login` go last. Include type hints and a docstring."
Explain an error	**"Why is my code broken?"**	"This Express route returns HTTP 500 when the id query parameter is missing. The handler and stack trace are below. Explain the cause and suggest a fix that returns 400 instead."
Generate test data	**"Make some fake users."**	"Generate 5 user records as a JSON array with id (UUID), email and created_at (ISO 8601, during 2025). Return only the JSON."

§4 The four building blocks

01 · INSTRUCTION

What to do

The task, stated as a clear action: review, explain, write, convert.

02 · CONTEXT

Why and where

The situation the answer has to fit: the system, the audience, the constraints.

03 · INPUT DATA

What to work on

The code, log, schema or text the task applies to.

04 · OUTPUT FORMAT

What to return

The structure, length and level of detail you expect back.

§5 One prompt, built block by block

A code-review request grows from one line into a prompt the model can answer well.

INSTRUCTION

Start with the task

"Review this pull request diff for bugs."

+ CONTEXT

Set the priorities

"It is part of a payment service. Correctness matters more than style."

+ INPUT DATA

Attach the material

The diff itself, pasted below the instructions.

+ OUTPUT FORMAT

Define the result

"List issues as a table: file, line, severity, explanation. Skip formatting nits."

Takeaway: A prompt is the complete input for a request. Instruction, context, input data and output format each remove a guess the model would otherwise make.

Start with the instruction, then add only the context, data and format details the task needs.

What Is Prompt Engineering?

Prompt engineering is the practice of designing, testing and refining prompts until a model reliably produces what you need. The workflow is close to debugging.

§1 Defining prompt engineering

DEFINITION

Designing prompts on purpose

Writing, testing and revising prompts so a model's output meets a defined goal, instead of accepting whatever the first attempt returns.

WHY IT MATTERS

Vague in, vague out

An underspecified prompt can produce generic code, or confident answers that are wrong, such as calls to functions that don't exist in the library.

THE SKILLS

Familiar engineering habits

Clear technical writing, knowledge of the problem domain and a testing mindset: define the expected result, then check against it.

§2 Example: asking for a regular expression

Each addition removes an assumption, and the last one makes the answer easy to verify.

FIRST ATTEMPT

Too broad

"Give me a regex for emails." The model has to guess the language, the strictness and how the pattern will be used.

+ CONTEXT

Say where it runs

"JavaScript, used only for an inline form hint. The server does the real validation."

+ OUTPUT

Say what to return

"Return the pattern, three strings it matches, three it rejects, and its known limitations."

§3 The prompt engineering process

Steps 3 to 5 repeat, the same way you rerun a test after each code change.

01

Define the goal

Decide what a correct answer looks like.

02

Draft the prompt

Write the first version.

03 · LOOP

Run it

Send the prompt and read the full response.

04 · LOOP

Evaluate

Compare the response with the goal.

05 · LOOP

Refine

Add context, constraints or examples.

06

Repeat

Loop until the output meets the goal.

§4 Worked example: a README section

STAGE	PROMPT OR FINDING
Goal	A README section that explains how to run the project locally.
First prompt	"Write a 'Getting started' section for our README."
Evaluate	Generic. It assumed npm, skipped the database and never mentioned the required environment variables.
Refined prompt	"Write a 'Getting started' section. We use pnpm and Node 20. PostgreSQL runs in Docker via docker compose. Required env vars: DATABASE_URL and API_KEY. Use numbered steps."
Final prompt	Same, plus: "Add a short troubleshooting list for a port that is already in use and a database that hasn't started."

§5 Why it matters

TIME

Fewer round trips

A precise prompt usually needs fewer tries to reach a usable answer.

QUALITY

Better results, same model

Good prompts improve output without retraining or switching models.

LIMITS

You learn where models fail

Testing prompts shows which tasks a model handles well and where it needs checking.

SAFETY

Less risk from wrong output

Clear constraints and verification steps make it less likely that a wrong answer reaches production.

Takeaway: Prompt engineering is a loop: define the goal, draft, run, evaluate and refine.

Treat a prompt like code under test. Decide what correct looks like first, then iterate until the output passes.

Best Practices for Writing Prompts

Four dimensions of a good prompt, each shown as a weak request rewritten into a strong one, using everyday engineering tasks.

§1 The four dimensions

01

Clarity

Plain, unambiguous wording that states the task.

02

Context

The background and details the task depends on.

03

Specificity

Exact requirements, with an example of the output when it helps.

04

Role

A perspective that sets the tone and depth of the answer.

§2 Clarity

Before

- **Prompt:** "Do the thing with the dates so it's not weird."
- Doesn't name the change
- "Weird" could mean format, time zone or sorting

After

Prompt: "Convert every timestamp in this function from local time to UTC before it is saved to the database."

Names the exact change and where it applies.

§3 Context

Before

- **Prompt:** "Why is the build failing?"
- No log or error message
- No information about what changed

After

Prompt: "Our GitHub Actions build started failing at the npm ci step after we upgraded to Node 22. The log is below. What changed, and how do we fix it?"

Gives the symptom, the recent change and the evidence.

§4 Specificity

Before

- **Prompt:** "Write better variable names."
- No naming convention
- Unclear whether behavior may change

After

Prompt: "Rename the variables in this function to descriptive camelCase names, for example `d` → `createdDate`. Keep the behavior identical and return only the code."

Sets the convention, shows an example and constrains the scope.

§5 Role

Before

- **Prompt:** "Review my SQL."
- No criteria for the review
- No sense of what matters most

After

Prompt: "Act as a senior database engineer reviewing a new team member's query. Point out performance problems such as missing indexes or repeated queries in a loop, and explain each one in a sentence or two."

The named criteria do most of the work; the role mainly sets tone and depth. A persona on its own does not reliably make answers more accurate.

Takeaway: Clear wording, enough context, specific requirements and a useful role give you control over the content, depth and format of the answer.

When an answer misses, check which of the four dimensions the prompt left out.

Tools for Prompt Engineering

The main kinds of tools for writing, testing and managing prompts, and a simple workflow that treats prompts like code.

§1 What prompt tools do

PURPOSE

Write and test faster

Tools give you a place to try prompts, compare results and adjust settings without writing an application first.

FOR TEAMS

Keep prompts consistent

Shared, versioned prompts mean everyone on a team uses the same tested wording.

IN PRODUCTION

Catch regressions

Test sets and evaluations show when a prompt change or model update makes results worse.

§2 Common kinds of tools

Products change quickly. The categories stay fairly stable, so choose by category first and product second.

CHAT ASSISTANTS

Explore ideas

General chat apps from model providers. Good for quick questions and rough drafts.

PLAYGROUNDS

Test precisely

Provider consoles where you set the system prompt, temperature and output length, and compare models.

IDE ASSISTANTS

Prompt next to your code

Editor integrations that see the open files, so prompts can refer to real code.

LIBRARIES

Prompts in your app

Frameworks with templates, variables and chaining for building prompts into software.

§3 Features worth looking for

VERSIONING

History of every change

See which wording produced which result, and roll back.

TEMPLATES

Variables instead of copy-paste

One prompt with placeholders such as `{language}` and `{code}`.

SETTINGS

Control over generation

Temperature, maximum length and stop sequences in one place.

COMPARISON

Side-by-side results

The same prompt against several models or versions at once.

EVALUATION

Repeatable checks

A fixed set of test inputs with expected properties to score each change.

SHARING

Team libraries

Reviewed prompts that others can reuse.

§4 A simple workflow

01

Draft

Iterate in a playground until the output looks right.

→

02

Template

Save the prompt with variables in the repository.

→

03

Test cases

Collect inputs with the output you expect.

→

04

Evaluate

Run the cases on every change.

→

05

Ship

Deploy the prompt like any other code change.

§5 Treat prompts like code

Version control

Keep prompts in the repository so changes show up in diffs.

Code review

Review prompt changes the way you review logic changes.

Tests

Rerun the evaluation set before merging a prompt or model update.

Takeaway: Chat apps, playgrounds, IDE assistants and libraries each fit a different stage of the work.

Whatever the tools, the habits matter most: version prompts, test them and review changes.

Text Prompt Techniques

Six techniques that make a language model's text output more reliable, a feedback loop for refining it, and when to include examples in the prompt.

§1 Why technique matters

THE PROBLEM

Output varies

Answers change with phrasing, and even the same prompt can return different answers because output is sampled. A temperature of 0 reduces this but does not remove it entirely.

THE LEVER

The prompt

Precise instructions narrow the range of answers toward the one you need.

THE LIMIT

Two inputs decide quality

Results depend on both the prompt and the model's capability. A prompt can supply missing facts, such as docs or code, but it can't make up for a task the model isn't capable of.

§2 Six techniques

TASK

Name the task

"Convert this Python 2 code to Python 3."

CONTEXT

Add the situation

"…It runs as an AWS Lambda function, so keep cold-start time low."

DOMAIN TERMS

Use precise vocabulary

"Explain the difference between optimistic and pessimistic locking in PostgreSQL."

NEUTRALITY

Ask for balance

"Compare REST and GraphQL for a mobile backend. List pros and cons of each without recommending one."

CONSTRAINTS

Set the boundaries

"Summarize this design doc in five bullet points for a reviewer."

FEEDBACK

Iterate on the answer

Respond to each draft with specific corrections until it fits.

§3 The feedback loop in practice

01 · YOU

Ask

"Write a function that fetches a user by ID."

→

02 · MODEL

Drafts

Returns a version that uses callbacks.

→

03 · YOU

Correct

"Use async/await and throw a NotFoundError if missing."

→

04 · MODEL

Revises

Returns the updated function.

→

05 · YOU

Accept

Stop once it meets the requirements.

§4 Zero-shot vs. few-shot prompting

ZERO-SHOT

No examples

Works for common, well-defined tasks. "Classify this log line as INFO, WARN or ERROR: 'Connection refused: database at db:5432 is unreachable'." The label is clear from the instruction alone. For borderline lines, state your thresholds.

FEW-SHOT

Show examples first

Include two or three examples when the format is specific. Show two commit messages in your team's style, then ask for a message for a new diff, and the model follows the same pattern. Models also copy the length and wording of examples, so vary them.

§5 What you gain

Predictable output

Clear tasks and formats produce answers you can parse and reuse.

Easier review

When the prompt states the requirements, you can check the answer against them.

More trust

Seeing how instructions shape the output makes the model's behavior easier to reason about.

Takeaway: Name the task, give context, use precise terms, ask for balance, set constraints and iterate.

Add examples (few-shot) when the output has to follow a specific format or style.

The Interview Pattern

Instead of writing one long prompt, ask the model to interview you first. It asks for the details it needs, then answers with them in mind.

§1 What it is

DEFINITION

The model asks first

You tell the model to ask clarifying questions before it produces an answer.

WHEN TO USE IT

Open-ended tasks

Useful when you don't know in advance which details matter, such as design decisions or planning.

THE RULE OF THUMB

Better answers in, better answer out

The more complete your answers to its questions, the more specific the final result.

§2 How it works

01 · YOU

Set it up

Describe the goal and ask for questions first.

→

02 · MODEL

Asks

One question at a time.

→

03 · YOU

Answer

Give facts, constraints and preferences.

→

04 · MODEL

Proposes

Produces a result based on your answers.

§3 Example: designing a database schema

Questions the model might ask. The final schema reflects your answers instead of a generic template.

The setup prompt

"You are helping me design a database schema for a small online bookstore. Before proposing anything, ask me up to five questions, one at a time, about the data, how it is read and written, and the expected scale. When you have enough information, propose the tables, keys and indexes, and explain your choices."

Q1

Entities

Which things do you need to store: books, authors, customers, orders?

Q2

Relationships

Can a book have several authors? Can an order contain several books?

Q3

Access patterns

What are the most frequent queries, and are reads or writes more common?

Q4

Scale

Roughly how many orders per day, and how much growth do you expect?

Q5

Constraints

Which database are you using, and are there fields that must be unique?

Q6

Data rules

How long must order history be kept, and is any data sensitive?

§4 Compared with a single prompt

SINGLE PROMPT

You must think of everything

Any detail you forget, the model fills in with an assumption you may not notice.

INTERVIEW PATTERN

The model helps find the gaps

Its questions surface missing requirements before any answer is written.

Takeaway: When a task depends on details you haven't worked out yet, let the model ask for them.

Set the goal, ask for one question at a time, answer fully, then review the final proposal.

The Chain of Thought Approach

Show the model a solved problem with its reasoning written out step by step, then ask a similar question. The model follows the same line of reasoning.

§1 What it is

DEFINITION

Reasoning in the prompt

The prompt includes a worked example with each step of the reasoning, not just the final answer.

THE METHOD

Break the problem down

A multi-step problem is solved as a series of smaller steps, each building on the last.

THE PAYOFF

Steps you can review

Visible steps help you spot a wrong assumption. They don't always reflect how the model actually reached its answer, so still verify the result.

§2 How it works

01 · YOU

Show a solved example

A problem similar to yours.

→

02 · YOU

Write out the steps

Each reasoning step, then the answer.

→

03 · YOU

Ask the new question

One that the same reasoning can solve.

→

04 · MODEL

Follows the pattern

Works through the steps, then answers.

§3 Example: estimating time complexity

The worked example

Question: What is the time complexity of a function that compares every item in a list with every other item using two nested loops?

Reasoning: The outer loop runs n times. For each pass, the inner loop also runs n times. That is n × n comparisons, so the complexity is $O(n^2)$.

The new question

What is the time complexity of a function that loops over a list of n items and, for each item, runs a binary search on an already-sorted list of n items?

Expected reasoning: The loop runs n times. Each binary search takes O(log n) steps. So the total is O(n log n).

The prompt has three parts:

1 · A similar solved problem → **2 · Its step-by-step reasoning** → **3 · The new problem**

§4 When to use it

A simpler variant helps on many models: ask the model to work through the problem step by step. Reasoning models, which think internally before answering, already do this; for them, providers generally recommend clear goals over step-by-step instructions or worked examples.

Multi-step problems

Calculations, complexity analysis, tracing what a piece of code does.

Debugging

Ask the model to reason through the code path step by step before naming the bug.

Verification

Visible steps make it easier to spot a wrong assumption before you rely on the answer.

Takeaway: Show one solved problem with its reasoning, then ask the next one.

The worked steps teach the model the approach, and they give you steps to review. Still verify the final answer.

The Tree of Thought Approach

Ask the model to explore several possible approaches, evaluate each one, drop the weak ones and continue with the strongest. Useful for decisions with real trade-offs.

§1 What it is

DEFINITION

Chain of thought, branched

An extension of chain of thought: instead of one line of reasoning, the model works out several options and compares them.

THE STRUCTURE

A prompt shaped like a tree

The prompt asks for multiple options, a step-by-step evaluation of each, and explicit rules for discarding options. This single prompt approximates the original research technique, which runs as a search loop over many model calls.

BEST FOR

Trade-offs and constraints

Design and architecture choices where several answers are plausible and hard requirements rule some out.

§2 How it works

01 · BRANCH

Generate options

Several distinct approaches to the problem.

→

02 · EXPLORE

Reason about each

Work through how each one would play out.

→

03 · SCORE

Rate them

Score each one. Treat scores as a ranking, not a measurement.

→

04 · PRUNE

Keep the best

Drop options that fail, then recommend one.

§3 Example: choosing a caching strategy

1 · The instructions

"Propose three different approaches to the problem below. For each one, reason one step at a time: how it works, its main risk, and a score from 1 to 10 for how well it fits. Discard any approach that breaks a hard requirement. Then recommend the strongest remaining approach and explain why."

2 · The problem

"Our product-page API takes about two seconds to respond. Product data changes every few minutes. We run three application servers behind a load balancer, and all servers must return the same data."

Branches the model might produce:

BRANCH	APPROACH	EVALUATION
A	**In-memory cache on each server**	Pruned: each server would hold a different copy, which breaks the requirement that all servers return the same data.
B	**Shared cache (for example Redis) with a short expiry**	Kept: all servers read the same entries, and a few-minute expiry matches how often data changes.
C	**CDN caching of the full response**	Kept with limits: fast for anonymous visitors, but harder to use for personalized pages.

§4 Chain of thought vs. tree of thought

CHAIN OF THOUGHT

One line of reasoning

Follows a single path from the question to the answer.

TREE OF THOUGHT

Several paths, compared

Explores alternatives, scores them and keeps only those that meet the requirements.

§5 What it gets you

Considered alternatives

You see options you might not have thought of.

Explicit trade-offs

Each option's risks are written down, not hidden in a single recommendation.

Requirements enforced

Options that break a hard constraint are removed before the final answer.

Takeaway: For decisions with trade-offs, ask for several options, an evaluation of each, and a reason for the final pick.

The model's evaluation is a starting point. Check the reasoning against your own system before you act on it.

Image Prompt Techniques

Five techniques for prompting image models, with examples aimed at the images engineers often need: app illustrations, icons, diagrams for docs and placeholder art.

§1 What an image prompt is

THE PROMPT

A description of the image

From a short phrase to a detailed description of subject, composition, colors and style.

WHERE ENGINEERS USE IT

Everyday needs

Onboarding illustrations, empty-state art, icons, blog headers and mockup placeholders.

THE GOAL

Usable, consistent images

Images that fit the product's style and need little editing afterwards.

§2 Five techniques at a glance

01

Style modifiers

Set the visual style.

02

Quality terms

Ask for detail and sharpness.

03

Repetition

Reinforce the main idea.

04

Weights

Make some terms count more.

05

Negative prompts

Rule out common defects.

§3 Style modifiers and quality terms

01 · STYLE MODIFIERS

Match the product's look

Describe the medium and style instead of an artist: "flat vector illustration", "isometric", "line art", "soft gradients", "brand colors navy and teal".

02 · QUALITY TERMS

Ask for a clean result

Terms such as "high resolution", "sharp focus", "clean edges" and "simple background" push toward images that work at small sizes and on screens.

Example prompt

"Flat vector illustration of a developer at a desk reviewing code on two monitors, isometric view, brand colors navy and teal, simple light background, clean edges, high resolution."

§4 Repetition and weights

The weights in the table below are illustrative. How weights are written depends on the tool.

03 · REPETITION

Say it again for emphasis

Repeating a key idea, such as "minimal, very minimal, few details", can pull the result toward it in keyword-style tools. Newer models respond better to one clear description. It can also help to generate several variations and pick the best.

04 · WEIGHTS

Turn terms up or down

Mainly diffusion-based tools, and Midjourney through its own syntax, let you weight terms. Chat-based image tools usually do not, so describe what you want instead.

TERM	WEIGHT	EFFECT
isometric server rack	high	The main subject is emphasized.
soft gradient background	low	Present, but kept in the background.
photorealistic	negative	Pushes the image away from a photo look.

§5 Negative prompts

Common defects

- Distorted hands and faces.
- Garbled text. Newer models handle text better, but results still vary.
- Blur, watermarks and cluttered backgrounds.

05 · THE FIX

Say what to leave out

Diffusion-based tools often have a negative-prompt field, for example "blurry, watermark, text, extra fingers". It reduces these defects, though it can't remove them entirely. In tools without one, describe what you want instead: writing "no text" can make text appear. Add any text yourself afterwards.

Takeaway: Style, quality terms, repetition, weights and negative prompts give you control over images you'd otherwise get by chance.

Describe the style in plain terms, keep text out of generated images, and check each tool's syntax and its license terms before using images commercially.

Using AI Safely as an Engineer

Habits that keep AI-assisted work correct and secure: verifying generated code, protecting secrets, using system prompts and structured output, and defending against prompt injection.

§1 Why it matters

CORRECTNESS

Confident is not correct

Models can invent functions, use outdated library versions or miss edge cases, and still sound certain.

DATA

What you paste leaves your machine

Prompts are sent to the provider, and depending on its terms may be logged or retained.

ATTACK SURFACE

Apps add new risks

When software builds prompts from untrusted text, that text can try to steer the model.

§2 Verify generated code

01

Read it

Understand every line before using it.

→

02

Run it

Execute it in a safe environment.

→

03

Test it

Add tests, including edge cases and failures.

→

04

Check the APIs

Confirm functions and versions against the official docs.

→

05

Review it

Treat it like any pull request from a teammate.

§3 Protect secrets and code

Never paste into unapproved tools

- API keys, passwords, tokens and private keys.
- Customer or personal data.
- Proprietary code your company has not cleared for that tool.

WHAT TO DO INSTEAD

Redact and use approved tools

Use the tools your company has approved. Replace secrets with placeholders such as `DATABASE_URL=<redacted>`, and share the smallest snippet that shows the problem.

§4 System prompts and structured output

SYSTEM PROMPT

Rules for the whole conversation

Instructions set by the application developer that apply to every turn, such as tone, scope and output rules. User messages carry the individual requests. Put stable rules in the system prompt.

STRUCTURED OUTPUT

Answers your code can parse

Ask for JSON that matches a schema. Many APIs offer a structured-output or JSON mode that enforces this. Validate the result before using it anyway.

§5 Prompt injection

The risk

- Your app puts untrusted text into a prompt: a web page, an email, an uploaded file.
- That text contains instructions, and the model may follow them instead of yours.

MITIGATIONS

Reduce the damage

Treat model output as untrusted input. Limit which tools and actions the model can trigger. Mark untrusted text clearly with delimiters. Require a human to confirm sensitive actions. These reduce the risk; none removes it.

Takeaway: Verify what the model writes, keep secrets out of prompts, and treat model output as untrusted input.

The same habits you apply to code from any unfamiliar source apply to code from a model.