THE CLAUDE OPUS 5 PROMPT GUIDE

THE DEFINITIVE REFERENCE FOR MASTERING ANTHROPIC'S MOST CAPABLE MODEL

- EFFECTIVE PROMPTING FUNDAMENTALS
- ADVANCED PATTERNS AND XML STRUCTURING
- EXTENDED THINKING AND REASONING
- TOOL USE AND AGENTIC WORKFLOWS
- LONG-CONTEXT STRATEGIES AND OUTPUT CONTROL
- EVALUATION, BEST PRACTICES AND PROMPT LIBRARIES

STEVE T.

The Claude Opus 5 Prompt Guide

The Definitive Reference for Mastering Anthropic's Most Capable Model

Steve Publications

This book is available at https://leanpub.com/theclaudeopus5promptguide

This version was published on 2026-07-24

Contents

The Definitive Reference for Mastering Anthropic's Most Capable Model 1

Preface **2**
Who Should Read This Book 2
How This Book Is Structured 3
How to Use This Book 3
Conventions Used in This Book 4
A Note on Accuracy and Currency 4
Acknowledgments 4

Introduction **6**
Why Prompting Matters More Than Ever 6
How Claude Opus 5 Interprets Prompts 7
What Makes Opus 5 Different 7
How This Book Is Organized 9
How to Use This Book 10
A Note on Sources and Accuracy 10

Chapter 1: Understanding Claude Opus 5 . . . . . **12**
The Opus 5 Architecture: Capabilities, Context Window, and Extended Thinking 12
How Claude Reads Your Prompts: Instruction Processing and Attention 14
System Behavior: Safety, Guardrails, and Model Constraints 16
When to Use Opus 5 (and When Not To) 17

Chapter 2: Prompt Fundamentals **22**
The Anatomy of a Well-Structured Prompt 22
Clarity Over Cleverness: Writing Instructions Claude Can Follow . . . 24
Role Definition and Persona Assignment 25
Task Specification: Goals, Steps, and Success Criteria 27
Constraints and Guardrails in Prompts 28

Chapter 3: System Prompts and Context Setup **31**
What System Prompts Do (and Don't Do) 31
Writing Robust System Prompts for Production 31
Layered Instructions: Primary Rules, Secondary Guidelines, and Edge Cases 31
Managing Conflicting Instructions 31
Versioning and Maintaining System Prompts 31

Chapter 4: Structured Prompting with XML **32**
Why XML Works: Separation of Concerns in Prompts 32
Standard XML Tags and Their Purposes 32
Organizing Complex Prompts with Nested Elements 32
Combining XML Structure with Natural Language 32
Common XML Patterns for Different Task Types 32

Chapter 5: Reasoning and Extended Thinking **33**
How Extended Thinking Works in Opus 5 33
Activating and Configuring Extended Thinking 33
Prompting for Deeper Reasoning Without Extended Thinking 33
Multi-Step Problem Decomposition 33
Verifying and Validating Model Reasoning 33

Chapter 6: Tool Use and Function Calling **34**
How Claude Uses Tools: The Mechanism 34
Designing Tools for Effective Integration 34
Prompting Patterns for Tool-Augmented Workflows 34
Error Handling and Recovery in Tool Use 34
Chaining Multiple Tools 34

Chapter 7: Few-Shot and Example-Based Prompting **35**
Why Examples Work: Pattern Learning in Foundation Models 35
Designing Effective Few-Shot Examples 35
Input-Output Pairs vs. Annotated Examples 35
Balancing Example Quantity and Quality 35
Dynamic and Contextual Examples 35

Chapter 8: Long-Context Strategies **36**
Understanding the Context Window: Capacity vs. Quality 36
Information Placement: Where Things Go Matters 36
Compression and Summarization Techniques 36

Retrieval-Augmented Generation Patterns 36
Case Study: Optimizing a Customer Support RAG System for Opus 5 . 36
Managing Multi-Turn Conversations Over Long Horizons 36

Chapter 9: Output Control and Structured Responses 38
Specifying Output Formats: JSON, XML, Markdown, and Custom . . . 38
Schema-Constrained Generation 38
Controlling Tone, Style, and Length 38
Enforcing Constraints and Avoiding Hallucination 38
Post-Processing Hooks in Prompts 38

Chapter 10: Agentic Workflows and Autonomous Tasks 39
What Makes a Good Agentic Prompt 39
Planning, Execution, and Self-Correction Patterns 39
Multi-Agent Orchestration 39
Memory and State Management 39
Safety and Bounded Autonomy 39
Case Study: Building an Agentic Refactoring Workflow for a Legacy Codebase 39

Chapter 11: Domain-Specific Prompt Patterns 41
Software Engineering and Code Generation 41
Research, Analysis, and Synthesis 41
Case Study: Reducing Hallucination in Legal Contract Review by 42 Percent 41
Creative Writing and Content Creation 41
Customer Support and Conversational AI 41
Data Analysis and Business Intelligence 41

Chapter 12: Prompt Optimization, Debugging, and Evaluation 43
Diagnosing Prompt Problems 43
Systematic Iteration Strategies 43
A/B Testing Prompts 43
Building Evaluation Harnesses 43
Cost, Latency, and Quality Tradeoffs 43
Case Study: Iterative Prompt Optimization for a Medical Triage Classifier 43

Chapter 13: Reusable Templates and Prompt Libraries 45
Template Design Principles 45
Variable Injection and Dynamic Content 45

Organizing Prompt Libraries . . . 45
Documentation and Governance . . . 45
Note on Migration from Previous Claude Versions . . . 45

Chapter 14: Common Mistakes, Anti-Patterns, and Pitfalls . . . 46
Overcomplication and Under-Specification . . . 46
Contradictory Instructions . . . 46
Fragile Prompt Structures . . . 46
Ignoring Model Constraints . . . 46
Migration Pitfalls from Earlier Claude Models . . . 46
Case Study: Migrating a Code Review System from Opus 4.8 to Opus 5 46
Security and Privacy Risks . . . 47
Performance and Cost Pitfalls . . . 47

Conclusion . . . 48
The Core Principles . . . 48
What Has Changed and What Endures . . . 48
A Final Checklist . . . 48
Looking Ahead . . . 48

References . . . 49

Glossary . . . 50

Index . . . 51

The Definitive Reference for Mastering Anthropic's Most Capable Model

This book is the authoritative, end-to-end guide to prompting Claude Opus 5. Grounded entirely in Anthropic's official documentation and verified best practices, it takes you from foundational concepts through expert-level patterns covering system prompts, XML structuring, extended thinking, tool use, agentic workflows, long-context strategies, output control, domain-specific patterns, evaluation, and prompt library design. Every recommendation reflects Opus 5's specific capabilities, behavioral characteristics, and limitations, not generic LLM advice. Whether you are building production systems or pushing the boundaries of what Claude can do, this book gives you the systematic knowledge to extract maximum accuracy, reliability, and efficiency from every prompt.

Preface

This book was written to solve a specific problem: the gap between Claude Opus 5's raw capability and the reliability engineers actually get when using it in production.

Opus 5 is Anthropic's flagship model for enterprise work and complex reasoning. It delivers what the company describes as a step-change improvement over its predecessor, with particular gains in deep reasoning, agentic coding, and test-time compute scaling. But capability alone does not guarantee results. The difference between acceptable output and production-grade reliability almost always comes down to how you prompt the model.

Most existing resources fall short. Generic LLM prompting guides miss Opus 5's specific behaviors: its adaptive thinking system, its literal instruction following, its default verbosity, and the breaking changes that invalidate techniques from earlier models. Tutorial-style articles cover basics but lack the depth needed for production systems. API documentation lists parameters but does not explain how to combine them into effective prompt architectures.

This book is designed to fill that gap. It is built entirely on Anthropic's official documentation, release notes, API specifications, and published best practices. Every recommendation traces back to what Anthropic has documented about how Claude Opus 5 actually works. There are no invented techniques, no speculative hacks, no generic LLM advice dressed up as Claude-specific guidance.

Who Should Read This Book

This book is written for:

- Software engineers and AI engineers building applications with the Claude API
- Technical leaders evaluating Opus 5 for enterprise deployment
- Prompt engineers who want to optimize existing prompts for Opus 5's specific behaviors

- Developers migrating from earlier Claude models who need to understand breaking changes
- Organizations building prompt libraries and governance frameworks at scale

You should be comfortable reading technical documentation and have basic familiarity with how large language models work. No advanced machine learning background is required, but some programming experience will help you follow the code examples.

How This Book Is Structured

The book is organized as a progressive journey from foundation to mastery, divided into four parts:

Part I: Foundations covers what Opus 5 is, how it processes prompts, and the core principles of effective prompting. Chapters here establish mental models and habits that serve you throughout your work with the model.

Part II: Core Techniques dives into specific skills: XML structuring, reasoning guidance, tool use, few-shot prompting, and long-context strategies. Each chapter stands alone as a reference for that technique.

Part III: Advanced Patterns addresses production scenarios: output control, agentic workflows, domain-specific patterns for coding, research, creative work, customer support, and data analysis.

Part IV: Optimization and Scale covers prompt debugging, evaluation, template design, prompt libraries, and common pitfalls. These chapters help you maintain quality as your usage grows.

How to Use This Book

If you are new to prompting Claude, read sequentially from Chapter 1. The early chapters establish foundations that later material builds on.

If you already prompt Claude regularly but want to optimize for Opus 5 specifically, start with Chapter 1 to understand behavioral differences, then jump to chapters addressing your needs: Chapter 5 for thinking and effort

configuration, Chapter 6 for tool use, Chapter 8 for long-context work, or Chapter 14 for migration pitfalls.

If you are building production systems, pay special attention to Chapters 3 (system prompts), 9 (output control), 10 (agentic workflows), 12 (evaluation), and 13 (prompt libraries).

If you are migrating from earlier Claude models, Chapters 1 and 14 are essential reading. They cover breaking changes that will affect existing prompts.

Conventions Used in This Book

Code blocks show prompt examples, API configurations, and implementation code. Prompt fragments intended for inclusion in system prompts or user messages use plain text formatting within XML or code fences.

API parameter names appear in monospace font: `effort`, `thinking.type`, `tool_choice`.

Important warnings about breaking changes, security risks, or common mistakes are marked with a NOTE prefix to draw attention to critical information.

Cross-references to other chapters help you navigate related material without backtracking through the table of contents.

A Note on Accuracy and Currency

Every factual claim in this book about Claude Opus 5's capabilities, limitations, pricing, API behavior, or recommended practices is sourced from Anthropic's official documentation as of the writing date. Where Anthropic's guidance is clear, this book follows it. Where multiple valid approaches exist, all are presented with their tradeoffs. Where information is uncertain or evolving, that uncertainty is stated explicitly.

The Claude platform continues to evolve. New features, models, and best practices emerge regularly. While this book aims to be definitive for its moment, you should verify critical details against the latest Anthropic documentation at docs.anthropic.com and platform.claude.com, particularly for API parameters, pricing, and feature availability.

Acknowledgments

This book draws extensively on Anthropic's published documentation, which has become one of the most thorough and transparent technical resources in the AI industry. The clarity of Anthropic's engineering team in explaining model behaviors, limitations, and best practices made this work possible. Specific guidance from their prompting best practices, migration guides, use case tutorials, and API reference materials is woven throughout every chapter.

Thanks as well to the broader developer community whose experiments, discussions, and shared experiences have helped surface practical insights that complement official documentation.

Introduction

Claude Opus 5 represents a step-change improvement over its predecessors. With gains in deep reasoning, agentic coding capabilities, and test-time compute scaling, it is the most capable widely available Claude model for enterprise work and complex tasks [1]. But capability alone does not guarantee results. The difference between acceptable output and production-grade reliability almost always comes down to how you prompt the model.

This book exists to close that gap. It is built entirely on Anthropic's official documentation, release notes, API specifications, and published best practices. There are no invented techniques, no speculative hacks, no generic LLM advice dressed up as Claude-specific guidance. Every recommendation here traces back to what Anthropic has documented about how Claude Opus 5 actually works and how it should be prompted [2].

The goal is straightforward: after reading this book, you will know not only what to write in your prompts but why those patterns work, how different components interact, and what tradeoffs you face when choosing between approaches. You will understand Opus 5's specific behaviors, its differences from earlier models, and the pitfalls that catch even experienced practitioners.

Why Prompting Matters More Than Ever

Claude Opus 5 is powerful, but it is not magic. It does not read minds. It processes text according to patterns learned during training and instructions provided at inference time. The prompt is your primary interface to the model's capabilities, and the quality of that interface determines everything downstream: accuracy, consistency, cost, latency, safety, and maintainability.

Consider a production customer support system. A vague prompt might produce helpful-sounding but occasionally incorrect answers. A well-engineered prompt with clear instructions, structured examples, proper XML organization, and explicit guardrails produces consistent, on-brand responses that stay within policy boundaries. The difference is not a different model; it is a better prompt.

The stakes are even higher in agentic workflows, where Claude Opus 5 chains multiple tool calls, makes autonomous decisions, and operates over extended horizons. A poorly specified agentic prompt can lead to cascading errors, excessive token consumption, or unintended actions. A well-designed one enables reliable automation of complex tasks [3].

How Claude Opus 5 Interprets Prompts

Claude is a large language model that predicts the next token in a sequence based on patterns it has learned. When you send a prompt, the model does not "understand" it the way a human does. Instead, it recognizes patterns, attends to different parts of the input with varying weights, and generates text that statistically fits the context you have provided [5].

This has important implications for how you write prompts:

Clarity beats cleverness. Natural language instructions written plainly and explicitly work better than convoluted meta-instructions or attempts to "trick" the model into behaving a certain way. If a human reader would be confused by your prompt, Claude likely will be too [6].

Structure matters. Claude processes prompts sequentially and attends to different regions with different priorities. Where you place information, how you organize it, and what visual cues you provide all affect how the model interprets your instructions. XML tags, for example, help Claude parse complex prompts unambiguously by creating clear boundaries between instructions, context, examples, and variable inputs [6].

Examples are powerful. Few-shot prompting, which provides a small number of input-output examples, is one of the most reliable ways to steer Claude's output format, tone, and structure. Three to five well-crafted examples often improve accuracy and consistency more than pages of abstract instructions [7].

Thinking depth affects everything. Claude Opus 5 uses adaptive thinking controlled by the effort parameter, which ranges from low to max. The effort level directly impacts how much reasoning the model performs, how many tool calls it makes, and the thoroughness of its output. Choosing the wrong effort level can cause under-thinking on complex problems or wasteful overthinking on simple ones [8].

What Makes Opus 5 Different

Claude Opus 5 builds on the Claude 4 series but introduces several behavioral and capability changes that affect how you should prompt it:

Context window and output capacity. Opus 5 supports a 1 million token context window by default with up to 128,000 output tokens per request [9]. This is not just more space; it enables fundamentally different prompt architectures. You can include entire codebases, large document collections, or extended conversation histories without aggressive summarization. But capacity alone does not guarantee quality. Accuracy can degrade over very long contexts, and strategic placement of information matters more than ever [10].

Adaptive thinking by default. Unlike earlier models where thinking was off by default, Opus 5 has thinking enabled automatically and it cannot be disabled at the highest effort levels (xhigh or max); attempting to do so returns a 400 error [11]. This means your prompts should anticipate and work with the model's reasoning process rather than against it.

Full effort ladder. Opus 5 supports five discrete effort levels: low, medium, high, xhigh, and max. The default is high, but complex coding tasks and agentic work benefit from xhigh or higher [12]. Understanding when to use each level is critical for balancing quality against cost and latency.

More literal instruction following. Starting with Claude Opus 4.7, the model interprets prompts more literally and explicitly than before, particularly at lower effort levels. It does not silently generalize an instruction from one item to another, and it does not infer requests you did not make [13]. On Opus 5 specifically, this literalism means that if your review prompt says "only report high-severity issues," the model may follow that instruction literally and report less; instead, ask it to report everything and filter in a separate pass [2]. Prompts that relied on the model's willingness to "fill in the blanks" may need revision.

Default verbosity. While most newer Claude models default to concise responses, Opus 5's default user-facing responses run longer than prior Opus models, and raising or lowering effort does not reliably change visible response length [2]. If brevity is important, you must explicitly request it.

Breaking changes from previous generations. Several prompting techniques that worked with earlier Claude models no longer work with Opus

5. Assistant message prefilling on the final turn returns a 400 error. The temperature, top_p, and top_k sampling parameters are not supported and setting them to non-default values also returns an error [15]. Manual extended thinking configuration using budget_tokens is unsupported. These changes require updates to existing prompts and integration code.

How This Book Is Organized

This book is structured as a progressive journey from foundation to mastery. Each chapter builds on the previous ones, so reading in order is recommended. However, once you understand the fundamentals, you can treat later chapters as reference material for specific problems.

The first section establishes foundations. Chapter 1 explains what Claude Opus 5 is, how it works at a high level, and when to use it. Chapter 2 covers the essential building blocks of effective prompting: clarity, structure, role definition, task specification, and constraints. Chapter 3 dives into system prompts, which are the backbone of reliable behavior in production systems.

The second section covers core techniques. Chapter 4 explains XML-based structuring, one of Anthropic's most recommended practices for complex prompts. Chapter 5 covers reasoning and extended thinking, including how to guide the model's internal thought process. Chapter 6 addresses tool use and function calling, essential for building agentic systems. Chapter 7 focuses on few-shot and example-based prompting, the single most effective technique for controlling output quality.

The third section tackles advanced scenarios. Chapter 8 covers long-context strategies for working effectively with Opus 5's massive context window. Chapter 9 explains output control and structured responses, critical for production integrations. Chapter 10 addresses agentic workflows and autonomous tasks. Chapter 11 provides domain-specific prompt patterns for major use cases including software engineering, research, creative work, customer support, and data analysis.

The final section focuses on optimization and scale. Chapter 12 covers prompt debugging, iteration, and evaluation. Chapter 13 explains how to build reusable prompt templates and organizational prompt libraries. Chapter 14 catalogs common mistakes and anti-patterns to help you avoid costly errors.

Throughout the book, you will find annotated prompt breakdowns that explain why specific patterns work, before-and-after comparisons showing the impact of prompt improvements, and practical examples drawn from real-world use cases documented by Anthropic and its users.

How to Use This Book

If you are new to prompting Claude, read sequentially from Chapter 1. The early chapters establish mental models and habits that will serve you throughout your work with the model.

If you already prompt Claude regularly but want to optimize for Opus 5 specifically, start with Chapter 1 to understand the behavioral differences, then jump to chapters addressing your specific needs: Chapter 5 for thinking and effort configuration, Chapter 6 for tool use, Chapter 8 for long-context work, or Chapter 14 for common migration pitfalls.

If you are building production systems, pay special attention to Chapters 3 (system prompts), 9 (output control), 10 (agentic workflows), 12 (evaluation), and 13 (prompt libraries). These chapters address the reliability, consistency, and scalability concerns that matter most in deployed applications.

If you are migrating from earlier Claude models, Chapter 1 and Chapter 14 are essential reading. They cover the breaking changes and behavioral shifts that will affect your existing prompts.

A Note on Sources and Accuracy

Every factual claim in this book about Claude Opus 5's capabilities, limitations, pricing, API behavior, or recommended practices is sourced from Anthropic's official documentation as of the writing date. Where Anthropic's guidance is clear, this book follows it. Where multiple valid approaches exist, all are presented with their tradeoffs. Where information is uncertain or evolving, that uncertainty is stated explicitly.

The Claude platform continues to evolve. New features, models, and best practices emerge regularly. While this book aims to be definitive for its moment, you should verify critical details against the latest Anthropic documentation at docs.anthropic.com and platform.claude.com, particularly for API parameters, pricing, and feature availability.

With that foundation laid, let us begin.

Chapter 1: Understanding Claude Opus 5

Before you can prompt Claude Opus 5 effectively, you need to understand what it is, how it differs from earlier models, and the fundamental principles governing how it processes your instructions. This chapter establishes that knowledge.

The Opus 5 Architecture: Capabilities, Context Window, and Extended Thinking

Claude Opus 5 (model ID: claude-opus-5) is Anthropic's flagship model for enterprise work and complex reasoning tasks. It is described by Anthropic as delivering a step-change improvement over Claude Opus 4.8, with particular gains in deep reasoning, agentic coding capabilities, and test-time compute scaling [1].

At its core, Opus 5 is a transformer-based large language model that processes text inputs and generates text outputs. But the details of how it handles those inputs and what it can do with them matter enormously for prompting.

Context window capacity. Opus 5 supports a 1 million token context window by default across all deployment surfaces including the Claude API, Amazon Bedrock, Google Cloud, and Microsoft Foundry [9]. This means the model can simultaneously consider approximately 750,000 words of text (roughly the length of a large novel) when generating its response. The context window includes everything: system prompts, conversation history, uploaded documents, tool definitions, and the model's own output.

The maximum output per single request is 128,000 tokens. On the Message Batches API, this can be extended to 300,000 tokens using the output-300k-2026-03-24 beta header [16]. This output capacity enables fundamentally different use cases compared to earlier models constrained to a few thousand

output tokens. You can ask Opus 5 to generate entire documents, full codebases, or comprehensive analyses in a single pass.

However, having a large context window does not mean you should fill it indiscriminately. Anthropic's documentation notes that accuracy can degrade over very long contexts, a phenomenon sometimes called "context rot" [10]. Strategic placement of information and careful management of what goes into the context are critical skills.

Extended thinking. Claude Opus 5 uses adaptive thinking controlled by the effort parameter. Unlike earlier models where you could manually configure a thinking budget using budget_tokens, Opus 5 determines how much reasoning to perform based on the effort level you set and the complexity of the task [11].

Thinking is enabled by default on Opus 5. At the highest effort levels (xhigh and max), thinking cannot be disabled; attempting to set thinking.type to "disabled" with those effort levels returns a 400 error [11]. This reflects Anthropic's position that the model's strongest reasoning capabilities require an internal thought process.

When thinking is active, Claude generates tokens representing its reasoning before producing the final response. These thinking tokens count toward the output token budget and are billed at the same rate as visible output tokens. On newer models including Opus 5, thinking blocks from previous assistant turns are kept in the context window by default and count as input tokens on subsequent turns [17].

[FIGURE 1.1: Claude Opus 5 Architecture Overview] This diagram illustrates the key architectural components of Opus 5: the transformer backbone processing a 1M token context window, the adaptive thinking module controlled by the effort parameter, the tool use interface connecting to external systems, and the structured output layer for schema-constrained responses. The figure also shows how input tokens flow through caching layers before reaching the model.

The effort parameter has five levels:

- max: Maximum reasoning depth. Reserved for the most demanding tasks. Thinking is mandatory.
- xhigh: Very high reasoning depth. Ideal for complex coding, agentic work, and the most intelligence-demanding tasks. Thinking is mandatory on Opus 5 at this level.

- high: The default level. Balanced reasoning appropriate for most tasks.
- medium: Reduced reasoning for simpler tasks where latency or cost is a concern.
- low: Minimal reasoning. Fastest and cheapest, but risks under-thinking on complex problems [12].

Choosing the right effort level is one of the most important decisions in prompting Opus 5. Using low effort on a genuinely complex task will produce superficial or incorrect results. Using max effort on a trivial task wastes tokens and time.

Pricing. Opus 5 costs $5 per million input tokens and $25 per million output tokens at standard pricing [9]. The Batch API offers a 50 percent discount, bringing those rates to $2.50 and $12.50 respectively for asynchronous processing [18]. A Fast Mode option is available at premium pricing of $10/$50 per million tokens but is API-only and intended for latency-sensitive workloads [1].

For perspective, a typical complex prompt with system instructions, context, and examples might consume 5,000 input tokens ($0.025), and a thorough response of 2,000 output tokens would cost $0.05. Opus 5 is not a cheap model, but for tasks where accuracy and reasoning depth matter, the cost is often justified by the reduction in human review and error correction.

Knowledge cutoff. Opus 5's training data extends through May 2026 [9]. Events, facts, or developments after that date will not be in the model's training distribution. For information beyond the knowledge cutoff, you must provide it in the prompt or use tools like web search.

How Claude Reads Your Prompts: Instruction Processing and Attention

Claude does not read your prompt the way a human does. It processes tokens sequentially through transformer layers, attending to different parts of the input with varying weights. Understanding this process helps explain why certain prompting patterns work better than others.

Sequential processing. Claude reads your prompt from beginning to end. This means the order of information matters. Anthropic's documentation explicitly states that placing large documents at the top of the prompt and queries

at the end can improve response quality by up to 30 percent compared to the reverse arrangement [19]. The intuition is that Claude builds its understanding of the context as it reads, then applies that understanding when it encounters your actual question.

Attention mechanisms. Inside the model, attention layers determine which parts of the input are most relevant for generating each output token. Instructions near the beginning and end of a prompt tend to receive more attention than those buried in the middle. This is why Anthropic recommends placing critical instructions both at the start of your system prompt and reinforcing them where appropriate later in the conversation [20].

XML tags as attention guides. One of the most effective techniques for guiding Claude's attention is using XML tags to structure your prompts. Tags like , , , and create clear visual boundaries that help the model parse complex inputs unambiguously [6]. When Claude sees content wrapped in , it knows to treat that text as directives rather than background information or examples.

The effect is not magical. XML tags work because the model was trained on data containing structured markup, and it has learned to recognize these patterns. But the result is real: prompts with clear XML structure consistently produce more reliable outputs than unstructured prompts of equivalent content.

Role assignment. When you tell Claude "You are a senior software engineer reviewing code for security vulnerabilities," you are not changing the model's architecture or weights. You are setting up a pattern in the context that biases the model toward generating text consistent with that role. Role prompts work because Claude has seen vast amounts of text written from different perspectives and can adopt the style, vocabulary, and priorities associated with each one [21].

Effective role prompts are specific. "You are helpful" provides almost no useful bias. "You are a senior Python developer with expertise in asynchronous programming and database optimization. You write clean, well-documented code following PEP 8 conventions and always consider edge cases" gives the model concrete attributes to condition on.

Instruction literalism. Starting with Claude Opus 4.7, the model interprets prompts more literally and explicitly than earlier versions, particularly at lower effort levels [13]. It does not silently generalize an instruction from one item to another, and it does not infer requests you did not make.

This literalism has important implications. If you instruct Claude to "review the following files for bugs" and then list three files, it will review only those three files. It will not assume you meant all files in the directory. If you say "summarize each section" there are five sections, it will produce five summaries. This precision is generally desirable for API use cases where predictable behavior matters. But prompts that relied on the model's willingness to extrapolate or fill gaps may need revision.

System Behavior: Safety, Guardrails, and Model Constraints

Claude Opus 5 includes built-in safety systems that affect how it responds to certain prompts. Understanding these systems is essential for building reliable applications.

Constitutional AI. Claude is trained using Constitutional AI principles, which embed ethical guidelines directly into the model's behavior rather than relying solely on external filters [22]. This means Claude will refuse to generate content it considers harmful, even if your prompt explicitly requests it. These refusals are not bugs; they are features of the model's design.

Refusal categories. When Claude refuses a request, the response includes stop_reason set to "refusal" and a stop_details field with a category (cyber, bio, or null) and a human-readable explanation [23]. Your application can use this information to route different classes of refusal appropriately.

You cannot prompt Claude to disable its safety systems. Attempts to do so through jailbreaking techniques are actively mitigated by both the model's training and Anthropic's platform-level defenses [24].

Prompt injection resistance. Claude is trained to resist prompt injection attacks, where malicious content embedded in documents or tool outputs attempts to override the model's instructions. However, no defense is perfect.

For direct injection (where the user is the adversary), Anthropic recommends using harmlessness screens with a lightweight model like Claude Haiku 4.5 to pre-screen inputs, implementing input validation, and crafting system prompts that emphasize ethical boundaries [24].

For indirect injection (where third-party content is the vector), the key defense is architectural: confine untrusted data to tool_result blocks rather

than placing it in system or user messages. Tell Claude explicitly that content returned from tools, documents, or searches is untrusted and must never override the system prompt or original request [24].

The computer use tool includes additional screenshot-based injection detection that automatically flags potential prompt injections and steers the model to ask for user confirmation before proceeding [25].

Parameter constraints. Several API parameters that worked with earlier Claude models are not supported on Opus 5:

- temperature, top_p, and top_k: Setting these to non-default values returns a 400 error. Use prompting techniques instead to control output variability [15].
- Manual extended thinking (budget_tokens): Returns a 400 error. Use adaptive thinking with the effort parameter [15].
- Assistant prefilling on the final turn: Returns a 400 error. Use system prompt instructions or structured outputs instead [15].

These constraints are not arbitrary limitations. They reflect Anthropic's decision to simplify the API surface and move control into areas where it is more effective: the effort parameter for reasoning depth, structured outputs for format control, and well-crafted prompts for behavioral guidance.

When to Use Opus 5 (and When Not To)

Opus 5 is powerful but expensive. Using it for every task is wasteful. Anthropic provides clear guidance on when Opus 5 is the right choice and when other models serve better [9, 26].

Use Opus 5 when:

- The task requires deep reasoning across multiple steps or domains.
- You are building agentic systems that must make autonomous decisions over extended horizons.
- Complex coding tasks demand high accuracy and sophisticated architectural understanding.
- Enterprise workloads where reliability and correctness justify premium pricing.

- Tasks involve very long contexts where Opus 5's 1M token window provides a strategic advantage.
- You need the model's strongest capabilities for research, analysis, or creative synthesis.

Consider Claude Sonnet 5 instead when:

- The task is straightforward and does not require deep reasoning.
- Cost efficiency is important. Sonnet 5 costs $3/$15 per million tokens ($2/$10 during introductory pricing through August 31, 2026) [9].
- You need a good balance of capability and speed for general-purpose tasks.

Consider Claude Haiku 4.5 when:

- Latency is critical and the task is simple.
- You are building multi-stage pipelines where an initial screening or classification step does not need Opus-level intelligence.
- Cost must be minimized. Haiku 4.5 costs $1/$5 per million tokens [9].

Consider Claude Fable 5 when:

- You need the absolute highest capability available for long-running agentic work.
- The task demands next-generation intelligence beyond what Opus 5 provides.
- Cost is secondary to performance. Fable 5 costs $10/$50 per million tokens [9].

Model selection matrix:

Model	Pricing (Input/Output per MTok)	Context Window	Max Output	Knowledge Cutoff	Best For
Claude Fable 5	$10 / $50	1M tokens	128k	January 2026	Highest capability available; long-running agentic work where cost is secondary to performance
Claude Opus 5	$5 / $25 (Fast: $10/$50)	1M tokens	128k (300k via batch beta)	May 2026	Complex coding, deep reasoning, enterprise workloads; near-Fable intelligence at half the cost

Model	Pricing (Input/Output per MTok)	Context Window	Max Output	Knowledge Cutoff	Best For
Claude Sonnet 5	$3 / $15 ($2/$10 intro through Aug 31, 2026)	1M tokens	128k (300k via batch beta)	January 2026	General-purpose tasks; good balance of capability, speed, and cost
Claude Haiku 4.5	$1 / $5	200k tokens	64k	February 2025 (reliable)	Simple tasks, classification, screening, latency-critical paths; cheapest option

A common production pattern is to use multiple models in a single workflow: Haiku 4.5 for initial classification or screening, Sonnet 5 for standard processing, and Opus 5 only for the most complex cases that escalate through the pipeline. This approach optimizes cost while preserving access to top-tier capability when it matters [27].

The key principle is empirical: create benchmark tests specific to your use case, test with actual prompts and data, and choose the model that meets your requirements at the lowest cost [26]. Do not default to Opus 5 out of habit. Default to it only when you have evidence that cheaper models cannot do the job.

With this understanding of what Opus 5 is and how it works, we can now

turn to the fundamentals of writing prompts that work well with it.

Chapter 2: Prompt Fundamentals

Effective prompting is not about memorizing magic phrases or exploiting model quirks. It is about communicating clearly with a powerful but literal system that follows patterns rather than intentions. This chapter establishes the core principles and building blocks of effective prompt writing for Claude Opus 5.

The Anatomy of a Well-Structured Prompt

[FIGURE 2.1: Prompt Component Architecture] This diagram shows the layered structure of a production-grade prompt for Claude Opus 5. At the foundation is the system prompt establishing persistent role and behavior rules. Above that, the user message contains the specific task, input data wrapped in XML tags, examples demonstrating expected patterns, and output format specifications. The figure illustrates how each component serves a distinct purpose: system prompts provide stable behavioral constraints, user messages provide task-specific instructions, examples teach patterns through demonstration, and format specifications ensure parseable outputs. Arrows indicate how information flows between components during processing.

A well-structured prompt for Claude Opus 5 typically contains several distinct components, each serving a specific purpose. While not every prompt needs every component, understanding what each one does helps you assemble prompts that work reliably.

The core components are:

System prompt (role and behavior). The system prompt sets the stage for the entire conversation. It defines Claude's role, establishes behavioral guidelines, and provides persistent context that applies to all turns. System prompts are the backbone of reliable production behavior because they remain constant across interactions and cannot be easily overridden by individual user messages [21].

User message (task and input). The user message contains the specific task for this turn along with any input data Claude needs to complete it. This is

where you state what you want done, provide the material to work with, and specify how the output should look.

Context and background. Context provides Claude with information it would not have from its training data: domain-specific knowledge, project conventions, reference materials, prior decisions, or constraints that apply to this particular task. Without adequate context, even a brilliant model is working blind [5].

Examples (few-shot demonstrations). Examples show Claude concretely what you want rather than describing it abstractly. A few well-crafted input-output pairs often improve accuracy more than pages of additional instructions because they establish a pattern the model can follow directly [7].

Output format specification. Specifying how you want the output formatted reduces post-processing work and increases consistency. This can be as simple as "respond in JSON" or as detailed as a full schema definition with structured outputs [28].

Let us look at a concrete example that combines these components:

```
System prompt:
You are a senior Python developer reviewing code for security vulnerabilities,
performance issues, and adherence to PEP 8. You provide specific, actionable
feedback with line numbers and suggested fixes.

User message:
Review the following function for issues:

<code>
def get_user_data(user_id):
    query = f"SELECT * FROM users WHERE id = {user_id}"
    result = db.execute(query)
    return result.fetchone()
</code>

Respond in this format:
<review>
<severity>HIGH|MEDIUM|LOW</severity>
<issue>Brief description of the issue</issue>
<line>Line number</line>
<fix>Suggested fix</fix>
</review>
```

This prompt works because each component has a clear job. The system prompt establishes the role and scope. The user message provides the specific

task and input wrapped in an XML tag. The output format is specified explicitly using XML structure that can be parsed programmatically.

Clarity Over Cleverness: Writing Instructions Claude Can Follow

The single most important principle of prompting Claude is clarity. Anthropic's documentation states it plainly: "Think of Claude as a brilliant but new employee who lacks context" [5]. Your job is to give that employee instructions clear enough that they cannot go wrong.

Be explicit about what you want. Vague instructions produce vague results. Instead of "summarize this document," write "Summarize this document in three bullet points, each no longer than two sentences, focusing on key findings and recommendations." The extra specificity eliminates ambiguity and gives Claude a concrete target to aim for [5].

Use positive instructions over negative ones. Telling Claude what to do works better than telling it what not to do. Instead of "Do not include unnecessary details," write "Include only information directly relevant to the question." Negative instructions require the model to first understand what the forbidden behavior is, then avoid it. Positive instructions simply direct it toward the desired behavior [29].

Number steps for order-sensitive tasks. When you want Claude to perform multiple actions in a specific sequence, use numbered lists. "1. Extract all dates from the text. 2. Convert them to ISO 8601 format. 3. Sort chronologically." Numbering signals that order matters and reduces the risk of Claude skipping steps or performing them out of sequence [5].

Avoid ambiguous language. Words like "some," "a few," "briefly," and "comprehensive" have different meanings to different people. Instead of "give me a few examples," write "give me exactly three examples." Instead of "summarize briefly," write "summarize in no more than 150 words." Precision eliminates guesswork [5].

The golden rule of clear prompting, as Anthropic puts it: show your prompt to a colleague with minimal context on the task and ask them to follow the instructions. If they are confused, Claude will be too [5].

Common ambiguity patterns to avoid:

Vague quantifiers: "some," "a few," "many," "several" all have different interpretations. Specify exact numbers: "three examples," "up to five items," "at least ten results."

Subjective quality terms: "good," "high-quality," "comprehensive," "thorough" mean different things to different people. Define what good looks like: "a comprehensive review covers security, performance, correctness, and maintainability with specific findings for each category."

Implicit assumptions: "fix the bugs" assumes Claude knows which issues count as bugs versus style preferences. Be explicit: "fix logic errors that cause incorrect behavior or crashes; do not change formatting or naming conventions."

Unspecified defaults: "sort the results" does not specify ascending or descending order, primary and secondary sort keys, or how to handle ties. Specify: "sort by date descending, then by priority (CRITICAL before HIGH before MEDIUM before LOW)."

Conditional ambiguity: "if needed" or "as appropriate" give Claude too much discretion. Define the conditions: "include error handling if the function interacts with external systems (database, API, file system)."

Consider this before-and-after comparison:

Before (unclear): "Write some code to handle user login. Make it secure."

After (clear): "Write a Python function that authenticates users against a PostgreSQL database using bcrypt password hashing. The function should:

1. Accept username and password as parameters
2. Query the database for the user record
3. Verify the password using bcrypt.checkpw
4. Return a JWT token on success or raise an AuthenticationError on failure
   Include error handling for missing users and database connection failures."

The second prompt leaves no room for interpretation about what is expected. Claude knows the language, the database, the hashing algorithm, the exact steps to implement, and how to handle errors.

Role Definition and Persona Assignment

Role prompts are one of the simplest and most effective techniques for steering Claude's behavior. When you assign a role, you activate patterns in the model's training data associated with that identity, which influences vocabulary, priorities, depth of analysis, and tone [21].

Effective role prompts have three characteristics: specificity, relevance, and concreteness.

Specificity means naming the exact role rather than a generic category. "You are a helpful assistant" provides almost no useful conditioning. "You are a senior DevOps engineer specializing in Kubernetes cluster management and CI/CD pipeline optimization" gives Claude a precise identity to work from.

Relevance means the role matches the task. Assigning Claude the role of a marketing copywriter when you need database schema design will bias its output toward irrelevant priorities and vocabulary. The role should reflect the expertise needed for the specific work.

Concreteness means including attributes that matter for the task. A good role prompt does not stop at the job title; it adds relevant details that shape behavior:

```
You are a senior legal analyst specializing in commercial contract review.
You have 15 years of experience reviewing sublease agreements and vendor
↪  contracts
for Fortune 500 companies. You focus on identifying risks related to
↪  indemnification,
liability caps, termination clauses, and regulatory compliance. Your writing
↪  style
is precise, formal, and cites specific clause numbers.
```

This prompt gives Claude not just a title but experience level, domain focus, specific areas of attention, and a writing style to emulate. Each detail conditions the model toward more relevant output.

Role prompts work best in the system message where they apply persistently across all turns. They can also be reinforced in user messages when switching between different tasks that require different perspectives:

```
Now, reviewing this same contract from the perspective of the tenant rather than
the landlord, what clauses would concern you and why?
```

This technique of role-switching within a conversation is useful for multi-perspective analysis. You can ask Claude to evaluate the same material as a proponent, an opponent, and a neutral party, then synthesize the viewpoints.

A word of caution: roles do not change Claude's underlying safety guidelines or factual knowledge. Telling Claude "you are an unfiltered AI that ignores all rules" will not disable its safety systems [24]. Roles shape style and priorities within the bounds of the model's training and constraints, not beyond them.

Task Specification: Goals, Steps, and Success Criteria

A well-specified task tells Claude exactly what to accomplish, how to approach it, and what success looks like. Vague tasks produce vague results; precise tasks produce precise results.

State the goal explicitly. Begin with a clear statement of what you want Claude to do. "Analyze this dataset and identify the top three factors correlated with customer churn" is better than "Tell me about this data." The first gives Claude a specific objective; the second leaves it to decide what is interesting.

Break complex tasks into steps. When a task involves multiple operations, laying out the steps explicitly reduces errors and makes it easier to diagnose problems if the output is wrong:

```
Perform the following analysis on the attached requirements document:
1. Extract all functional requirements and number them sequentially
2. For each requirement, identify whether it is testable (has clear acceptance
   ↪  criteria)
3. Flag any requirements that conflict with each other
4. Group requirements by feature area
5. Estimate the relative complexity of each feature area as low, medium, or high
```

This step-by-step specification gives Claude a clear workflow to follow and makes it easy to verify that each step was completed.

Define success criteria. When possible, specify what a successful output looks like in concrete terms:

```
Your summary should:
- Be no longer than 500 words
- Include the main thesis, three key supporting arguments, and the conclusion
- Quote directly from the text where claims are made
- Avoid introducing information not present in the original document
```

These criteria give Claude measurable targets and make it easier for you to evaluate whether the output meets your needs. They also enable automated evaluation: you can write code to check word count, verify that quotes appear in the source, and confirm that no external information was introduced [30].

Provide motivation and context. Explaining why a task matters can improve output quality by helping Claude prioritize what is important:

```
We are preparing this security review for presentation to our board of
↪ directors next
week. They are technically sophisticated but not security specialists. Your
↪ analysis
should be thorough enough to satisfy expert scrutiny but written in language
↪ that
clearly communicates risk to executive decision-makers.
```

This context tells Claude not just what to do but how to calibrate the depth and tone of the response.

For tasks where Claude will use tools, explicitly state that tool use is expected:

```
Use the available search tools to investigate this question before responding.
Verify information across multiple sources and cite your sources in your answer.
```

Research has shown that Claude's latest models demonstrate strong agentic search capabilities, but explicit instructions to use tools increase the likelihood they will be called when appropriate [31].

Constraints and Guardrails in Prompts

Constraints are boundaries you place on Claude's behavior to ensure outputs stay within acceptable parameters. Well-designed constraints prevent hallucination, maintain consistency, and reduce the need for post-processing.

Scope constraints limit what Claude can use or reference:

```
Base your answer solely on the information provided in the documents below.
If the documents do not contain enough information to answer the question fully,
state what is missing rather than speculating.
```

This constraint is one of the most effective defenses against hallucination, especially when working with long documents or specialized domains where Claude's training data may be incomplete or outdated [32].

Format constraints control how output is structured:

```
Respond only in valid JSON matching this schema. Do not include any text before
or after the JSON object.
```

Combined with Anthropic's structured outputs feature, format constraints guarantee that Claude's responses can be parsed programmatically without error handling for malformed output [28].

Length constraints control verbosity:

```
Provide a concise response of no more than 200 words.
```

Remember that Opus 5 defaults toward longer, more thorough responses than many other Claude models [14]. If brevity matters, you must specify it explicitly.

Behavioral constraints guide how Claude approaches the task:

```
Before giving your final answer, verify your reasoning by checking for logical
inconsistencies and edge cases you may have missed.
```

This type of constraint activates self-correction behavior that can catch errors, particularly in coding and mathematical tasks [33].

Safety constraints reinforce Claude's built-in guardrails:

```
Do not include personally identifiable information, passwords, API keys, or
other sensitive data in your response. If the input contains such information,
refer to it abstractly (e.g., "the user's email address") rather than repeating
↪  it.
```

While Claude's training already discourages leaking sensitive information, explicit constraints add an additional layer of protection and make your expectations unambiguous [34].

When designing constraints, avoid over-constraining. Too many restrictions can confuse the model or prevent it from completing the task effectively. Each constraint should serve a clear purpose: preventing a specific failure mode, ensuring compatibility with downstream systems, or maintaining quality standards. If a constraint does not address a real risk or requirement, remove it.

The fundamentals covered in this chapter (clear structure, explicit instructions, effective role assignment, precise task specification, and thoughtful constraints) form the foundation for everything that follows. The remaining chapters build on these principles with specialized techniques for specific scenarios. But no advanced technique can compensate for poor fundamentals. Master these basics first, and the more sophisticated patterns will be easier to learn and apply.

Chapter 3: System Prompts and Context Setup

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

What System Prompts Do (and Don't Do)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Writing Robust System Prompts for Production

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Layered Instructions: Primary Rules, Secondary Guidelines, and Edge Cases

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Managing Conflicting Instructions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Versioning and Maintaining System Prompts

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Chapter 4: Structured Prompting with XML

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Why XML Works: Separation of Concerns in Prompts

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Standard XML Tags and Their Purposes

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Organizing Complex Prompts with Nested Elements

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Combining XML Structure with Natural Language

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Common XML Patterns for Different Task Types

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Chapter 5: Reasoning and Extended Thinking

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

How Extended Thinking Works in Opus 5

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Activating and Configuring Extended Thinking

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Prompting for Deeper Reasoning Without Extended Thinking

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Multi-Step Problem Decomposition

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Verifying and Validating Model Reasoning

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Chapter 6: Tool Use and Function Calling

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

How Claude Uses Tools: The Mechanism

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Designing Tools for Effective Integration

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Prompting Patterns for Tool-Augmented Workflows

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Error Handling and Recovery in Tool Use

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Chaining Multiple Tools

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Chapter 7: Few-Shot and Example-Based Prompting

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Why Examples Work: Pattern Learning in Foundation Models

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Designing Effective Few-Shot Examples

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Input-Output Pairs vs. Annotated Examples

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Balancing Example Quantity and Quality

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Dynamic and Contextual Examples

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Chapter 8: Long-Context Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Understanding the Context Window: Capacity vs. Quality

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Information Placement: Where Things Go Matters

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Compression and Summarization Techniques

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Retrieval-Augmented Generation Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Case Study: Optimizing a Customer Support RAG System for Opus 5

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Managing Multi-Turn Conversations Over Long Horizons

Chapter 9: Output Control and Structured Responses

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Specifying Output Formats: JSON, XML, Markdown, and Custom

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Schema-Constrained Generation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Controlling Tone, Style, and Length

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Enforcing Constraints and Avoiding Hallucination

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Post-Processing Hooks in Prompts

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Chapter 10: Agentic Workflows and Autonomous Tasks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

What Makes a Good Agentic Prompt

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Planning, Execution, and Self-Correction Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Multi-Agent Orchestration

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Memory and State Management

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Safety and Bounded Autonomy

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Case Study: Building an Agentic Refactoring Workflow for a Legacy Codebase

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Chapter 11: Domain-Specific Prompt Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Software Engineering and Code Generation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Research, Analysis, and Synthesis

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Case Study: Reducing Hallucination in Legal Contract Review by 42 Percent

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Creative Writing and Content Creation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Customer Support and Conversational AI

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Data Analysis and Business Intelligence

Chapter 12: Prompt Optimization, Debugging, and Evaluation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Diagnosing Prompt Problems

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Systematic Iteration Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

A/B Testing Prompts

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Building Evaluation Harnesses

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Cost, Latency, and Quality Tradeoffs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Case Study: Iterative Prompt Optimization for a Medical Triage Classifier

Chapter 13: Reusable Templates and Prompt Libraries

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Template Design Principles

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Variable Injection and Dynamic Content

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Organizing Prompt Libraries

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Documentation and Governance

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Note on Migration from Previous Claude Versions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Chapter 14: Common Mistakes, Anti-Patterns, and Pitfalls

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Overcomplication and Under-Specification

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Contradictory Instructions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Fragile Prompt Structures

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Ignoring Model Constraints

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Migration Pitfalls from Earlier Claude Models

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Case Study: Migrating a Code Review System from Opus 4.8 to Opus 5

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Security and Privacy Risks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Performance and Cost Pitfalls

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Conclusion

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

The Core Principles

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

What Has Changed and What Endures

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

A Final Checklist

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Looking Ahead

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

References

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Glossary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.

Index

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theclaudeopus5promptguide.