BEYOND PROMPTING

Mastering Claude Opus 5.5 for Complex Work

A Practical Guide to Reliable AI-Assisted Professional Work

CODING | RESEARCH | ANALYSIS | WRITING | AUTOMATION

AARON HAYES

Beyond Prompting: Mastering Claude Opus 5.5 for Complex Work

A Practical Guide to Reliable AI-Assisted Professional Work

Steve Publications

This book is available at https://leanpub.com/beyondpromptingmasteringclaudeopus55forcomplexwork

This version was published on 2026-09-23

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

Contents

A Practical Guide to Reliable AI-Assisted Professional Work

This book teaches you how to consistently obtain high-quality, reliable, and controllable results from Claude Opus 5.5 on demanding professional tasks. It goes far beyond prompt templates and collects of tips, instead building your understanding of how to architect instructions, manage context, decompose complex work, design robust workflows, and verify outputs systematically. Whether you use Claude for coding, research, analysis, writing, or multi-step automation, you will learn principles and patterns that transfer to unfamiliar tasks rather than relying on memorized examples. The guidance is grounded in Anthropic's official documentation, supplemented by independent research and real-world practice, with careful attention to what is confirmed versus speculative.

Chapter 1: Introduction: The Problem with Prompting

Why Most Prompts Fail at Complex Work

You have probably written thousands of prompts. Some of them worked perfectly on the first try. Others required several rounds of adjustment. And some of them produced results that sounded impressive but contained errors, omissions, or fabricated details that you might not have noticed until it was too late.

There is a fundamental reason for this variability. Most prompting advice treats prompts as instructions to be optimized for a single outcome. You tweak the wording, add a role, throw in some examples, and hope the model produces what you want. This approach works fine for simple tasks. Summarize this article. Rewrite this email. Generate a SQL query. For straightforward requests with a narrow scope, casual prompting is often sufficient.

But when you move into complex professional work, the template-and-tweak approach breaks down. Complex work has requirements that are not immediately obvious until you are deep into a task. It involves trade-offs between competing priorities. It depends on context that cannot all fit into a single prompt. It must be reliable enough that other people can build on it. And it often has consequences when the output is wrong.

Consider a few examples:

You need Claude to review a 3,000-line code change for subtle bugs, explain its reasoning, and produce a report your team can act on. A simple prompt like "review this code" will produce a generic response. A longer prompt listing things to check might work better, but will it follow all the constraints? Will it catch the edge cases? What happens when it misidentifies something as a bug?

You need Claude to synthesize twenty research papers into a coherent literature review. You paste all the papers into the context window and ask for a

summary. The result may look professional, but you have no way to know which claims are grounded in the papers versus drawn from the model's training data.

You need Claude to analyze your company's quarterly financial data and produce insights for an executive presentation. The numbers matter. A fabricated metric, even one buried in a paragraph, could mislead a decision.

In each case, the problem is not that Claude is incapable. The problem is that the prompts treat complex work as if it were simple. They assume that stating the desired outcome is enough to guide the model toward a reliable result. They do not account for ambiguity, error, conflicting requirements, or the reality that professional work demands something closer to a specification than a suggestion.

This book exists because there is a gap between what most people know about prompting and what is required to use models like Claude Opus 5.5 for serious work. Bridging that gap does not require discovering secret formulas or memorizing clever tricks. It requires understanding a set of principles about how to design instructions and workflows that produce consistent, high-quality results. Those principles are the subject of this book.

From Template Collector to Prompt Architect

Many people who work with AI assistants collect prompts the way some people collect recipes. They find a prompt online that works well for a particular task, save it, and use it whenever the situation arises. The collection grows over time until you have a folder full of carefully preserved templates for everything from writing emails to generating code.

There are problems with this approach. Templates are fragile. They are optimized for a specific task under specific conditions with a specific model version. When any of those factors changes, the template may stop working well or stop working at all. Templates do not transfer. Having a perfect prompt for writing technical documentation does not help you with data analysis, contract review, or architectural design, even though the underlying skills overlap. And templates hide the reasoning behind their design. You know what the prompt says, but you may not understand why it is structured the way it is or when it would fail.

This book takes a different approach. It treats prompting as a design discipline rather than a collection of tricks. The goal is not to give you a library

of copy-and-paste prompts but to help you develop the ability to architect instructions for any task, even ones you have never encountered before.

Think of it like this. You could memorize a hundred specific SQL queries for different reporting needs, or you could learn how to write SQL so that you can construct any query you need from first principles. This book aims to do the latter for prompt design.

To be clear about terminology: the word "prompting" is a poor label for what we are actually doing with models like Claude Opus 5.5 on complex tasks. "Prompting" suggests a casual interaction, a question-and-answer exchange. But the design work required for professional-grade results is closer to software engineering, to system design, to writing specifications. A better word might be "instruction architecture" or "workflow design." This book will use "prompting" for familiarity but will treat it as the more substantial activity it has become.

What This Book Will Teach You

Here is a straightforward description of what you will learn:

First, you will understand Claude Opus 5.5 as a system, not a magic box. You will know its capabilities, its constraints, its behavioral patterns, its strengths and weaknesses. This matters because prompt design that ignores model behavior is prompt design that will fail in unpredictable ways.

Second, you will learn how to design individual prompts systematically. This includes how to structure instructions so the model follows them reliably, how to provide context without overwhelming it, how to specify outputs precisely, how to use examples effectively, and how to handle ambiguity. You will learn why some instructions work and others do not, even when they seem to say the same thing.

Third, you will learn how to handle complex tasks that cannot fit into a single prompt. This means decomposing work into manageable steps, maintaining coherence across those steps, managing state and context, handling errors when individual steps fail, and designing workflows that are robust rather than brittle.

Fourth, you will learn how to build reliability into your prompts and workflows. This includes techniques for reducing hallucinations and grounding

outputs in evidence, designing verification steps, anticipating failure modes, and testing prompts before depending on them in production.

Fifth, you will learn how to optimize for efficiency, which in practice means getting better results with fewer unnecessary iterations, less wasted context, and more deliberate design. Efficiency is not just about cost. It is about reducing friction in your work so that Claude becomes genuinely useful rather than a convenient distraction.

Sixth, you will see how these principles apply to specific domains. The book includes detailed treatment of research workflows, software development, writing and editing, data analysis, document processing, multi-step automation, and tool integration. Each domain has its own challenges and patterns, but the underlying principles remain consistent.

Finally, the book emphasizes trade-offs. Good prompt design is not about making every prompt more elaborate. Sometimes a simpler prompt produces better results. Sometimes additional context hurts more than it helps. Sometimes the right solution is not a better prompt but a different approach to the problem entirely. Understanding when and why to use each technique is what separates someone who knows prompting from someone who masters it.

A note on evidence: throughout this book, you will see references to Anthropic's official documentation, to Anthropic's published research, and in some cases to independent observations and community practices. The distinction between these sources is deliberate. Official documentation reflects Anthropic's tested knowledge about how the models behave and how to work with them. Community practices may be useful but are less rigorously validated. Author analysis reflects patterns that hold up across tasks and models but are not explicitly documented. Being clear about these distinctions matters because treating speculation as fact is one of the fastest ways to build unreliable workflows.

How to Use This Book

This book is organized to support two different ways of reading it.

If you are new to prompt design or have primarily used Claude for simple tasks, read it from beginning to end. The chapters build on each other in a deliberate sequence. You will start with foundational concepts about how

Claude Opus 5.5 works, then move to individual prompt design, then to multi-step workflows, and finally to domain-specific applications and advanced patterns. Each chapter assumes the material that precedes it.

If you already have substantial experience using Claude and want to deepen your understanding or troubleshoot specific problems, you can read selectively. The domain-specific chapters in the later part of the book are relatively self-contained. You can jump to the chapter on coding workflows if that is your primary use case, or to the research chapter if you are doing knowledge work, or to the efficiency chapter if you are struggling with cost and latency. But for foundational material like instruction design, context management, and reliability, reading in order will help you avoid reinventing solutions to problems that are addressed early in the book.

A few practical notes:

The book uses inline citations in square brackets, such as [1], to reference sources. These are collected in the References section at the end. If you encounter a claim that seems important or surprising, you can look up the citation to verify it or explore further.

Many chapters include examples that compare weak, adequate, and optimized prompts side by side. These are not templates to copy but illustrations of design principles in action. Pay attention to the explanations of why each prompt is structured the way it is, not just the words themselves.

The book does not include exercises, quizzes, or homework. It assumes you will learn by applying the concepts to your own work. If you find a chapter particularly relevant to what you are doing right now, the most useful thing you can do is close the book and try the techniques on a real task.

One final expectation: this book assumes you are willing to be precise about what you want. If you are satisfied with vague instructions and unpredictable results, you do not need this book. But if you need Claude to produce work you can depend on, the investment of understanding and deliberate design that this book describes will pay for itself quickly.

Chapter 2: Understanding Claude Opus 5.5

Where Opus 5.5 Fits in the Claude Lineup

Claude Opus 5.5 was released by Anthropic on September 22, 2026 as the first model in the Claude 5.5 family and the current flagship of the Opus tier [1,2]. To understand it properly, you need to know where it sits relative to other Claude models and what trade-offs that position represents.

Anthropic's model lineup as of September 2026 consists of several tiers optimized for different purposes [3]. At the top are the Mythos-class models (Fable 5.1 and earlier), which represent the maximum reasoning capability Anthropic has publicly deployed. Opus models are positioned just below Mythos-class but with a different optimization profile: Opus models are designed for long-running agentic coding and knowledge work, balancing capability with practical considerations like cost and latency [4,5]. Below Opus are Sonnet models, which target cost-effective performance for routine work, and Haiku models, which prioritize speed and efficiency for simple tasks.

Opus 5.5 is explicitly positioned by Anthropic as the model to "start with for most workloads" [3]. It matches Claude Fable 5.1 on performance for most tasks while costing approximately 40 percent less to run than the previous Opus model (Opus 5) on typical workloads [1]. This positioning is significant. It means that for the majority of professional work, Opus 5.5 provides the best balance of capability and cost. You should reserve Fable 5.1 for cases where you have tested Opus 5.5 at higher effort levels and still fall short of your requirements [3].

The Opus tier itself has evolved rapidly. The lineage runs from Claude Opus 4 through a series of incremental releases (4.1, 4.5, 4.6, 4.7, 4.8) to Opus 5 and now Opus 5.5, with each release improving capabilities in agentic coding, knowledge work, and long-context performance [6,7]. Opus 5.5 continues this trajectory, leading in agentic coding benchmarks (66.4 percent on Terminal-Bench 4.0) and knowledge work benchmarks (1846 Elo on GDPval-AA) [1].

The model has demonstrated practical capability on extreme tasks: testers reported completing a 680,000-line code migration in under a day using Opus 5.5 [1].

Understanding this lineage matters because it shapes how you should approach prompt design. If you have existing prompts that work with Opus 5 or earlier Opus versions, Anthropic states that they generally continue to work with Opus 5.5 but may need tuning for optimal results [8]. The behavioral differences between versions are discussed later in this chapter.

Documented Capabilities and Strengths

Opus 5.5's capabilities are documented in Anthropic's model overview and system materials [4,5,9]. The following are confirmed characteristics based on official documentation and verified sources.

The model supports a one million token context window and up to 128,000 output tokens per request (300,000 in Batch API beta) [4,10]. This means you can load substantial amounts of material into a single request: an entire codebase with over 75,000 lines of code, dozens of research papers, or hundreds of pages of documents [11,12]. The knowledge cutoff is June 2026 [4], meaning the model's training data extends to that point and it may not be aware of events or releases after that date without external tools.

Thinking is always on for Opus 5.5. Unlike earlier models where thinking could be disabled, Opus 5.5 returns a 400 error if you attempt to set thinking to disabled [4,5]. Thinking operates in adaptive mode by default, meaning the model decides how much reasoning to perform based on the task. You control this through the effort parameter rather than through prompt instructions [8]. The default effort level is medium, which Anthropic states matches or exceeds the performance of Opus 5 at high effort for coding and knowledge tasks [8].

The model supports all core Claude capabilities: text and image input, text output, multilingual processing, vision (including improved accuracy on charts and screenshots compared to earlier models) [8], tool use, and file processing including PDFs [4]. Independent tool calls execute in parallel, which is relevant for multi-step workflows [13].

Opus 5.5 includes a new biology safety classifier and expanded refusal categories that include biology and cybersecurity domains [5,8]. Tasks in these domains may route to safer models unless users have joined Anthropic's

verification programs [1]. There is also a reasoning_extraction refusal category that prevents users from attempting to extract the model's reasoning in text form [8].

For tool use, Opus 5.5 has some restrictions compared to earlier versions. Forced tool use, where you set tool_choice to any or tool to require a tool call regardless of task, is no longer supported and returns a 400 error [5]. The model uses auto tool choice by default, calling tools for capability-specific tasks not already in context [14]. The computer_20251124 tool is unsupported on the Claude API and Google Cloud for Opus 5.5; you must use computer_toolset_-20260801 instead [5].

Opus 5.5 supports prompt caching with a 512-token minimum and has a cache read price of $0.20 per million tokens, which is 60 percent less than Opus 5 [1,4]. This is a significant efficiency feature for repeated workflows. The model also supports batch processing with a 50 percent discount and beta features including fast mode and inline tool definitions [4].

Documented Limitations and Known Weaknesses

Being clear about what Opus 5.5 cannot reliably do is as important as understanding what it can do. The following are limitations documented by Anthropic or consistently observed across reliable sources.

Hallucination is not eliminated. Anthropic's official guidance on reducing hallucinations explicitly states that their techniques "reduce but do not eliminate hallucinations" [15]. This is a fundamental characteristic of language models. The model can generate plausible-sounding but incorrect information, particularly when:

- Asked about facts outside its knowledge cutoff without access to external tools
- Working with very large contexts where relevant information is buried
- Given ambiguous or incomplete instructions
- Asked to perform tasks it was not specifically trained for, such as precise numerical calculation without tool use

Opus 5.5 cannot be configured to disable thinking. This is a hard limitation, not a best practice recommendation. If your integration or workflow requires

the model to produce only visible output without internal reasoning, Opus 5.5 is not the right choice. You would need an earlier model version [5,8].

The model does not guarantee deterministic output. Running the same prompt twice may produce different results, particularly at higher effort levels where more reasoning is involved. For production workflows, this means you need to design verification steps rather than relying on consistency.

There are specific API-level limitations. Non-default sampling parameters are rejected on newer models including Opus 5.5 [16]. Modifying system prompts or tools mid-conversation causes 400 errors for API accounts created on or after August 31, 2026, due to preserved thinking validation [5]. This affects how you design long-running workflows and agent loops.

Text between tool calls now returns as thinking blocks with type progress-update rather than as visible text [5,8]. If your harness expects text in these positions, it will break without modification. This is a breaking change from earlier models.

The model has refusal behaviors in specific domains. Biology, cybersecurity, and reasoning extraction are refusal categories where Opus 5.5 may decline to provide certain outputs or route to different models [1,5,8]. Workflows operating in these domains need fallback logic to handle refusals gracefully.

Behavioral Characteristics You Should Expect

Beyond capabilities and limitations, there are behavioral patterns in Opus 5.5 that affect how you should design prompts. Some of these are explicitly documented; others are patterns that emerge from how the model operates.

Opus 5.5 is more concise and proactive than earlier models [13]. Anthropic explicitly recommends dialing back instructions that try to prevent "laziness" or encourage "over-thoroughness," because newer models do not need these compensations and they can interfere with performance [13]. This is a significant shift from prompt design habits that developed around earlier versions.

The model treats instructions literally. It is less likely than earlier models to infer unstated requirements or fill in missing details. This is a strength when your instructions are precise and a weakness when they are vague. If you say "summarize this document," Opus 5.5 will produce a summary without trying to guess what kind of summary you want. If you want specific sections emphasized or particular details included, you must state that explicitly.

Opus 5.5 handles multi-step tasks differently depending on how they are structured. For simple sequential operations within a single prompt, the model generally executes them reliably when numbered or clearly ordered [17]. For more complex branching logic or conditional workflows, the model benefits from decomposition into separate prompts or subagent calls [17,18]. The model's adaptive thinking means it will perform more reasoning on complex tasks without being explicitly told to "think carefully" [8].

The model defaults to LaTeX formatting for mathematical content unless you specify otherwise [13]. It also defaults to generic design styles for visual or UI-related requests unless you explicitly constrain the aesthetic [8]. These defaults matter if you are producing outputs for non-technical audiences or specific platforms.

When working with large contexts, Opus 5.5 performs better when the query appears at the end of the prompt rather than at the beginning. Anthropic documents that placing queries at the end can improve response quality by up to 30 percent [13]. This is a consistent pattern across Claude models and should be treated as a basic practice for long-context prompts.

Opus 5.5 responds to examples more effectively when they are diverse and well-constructed. Providing three to five examples that cover different aspects of the task is generally more useful than providing many examples that repeat the same pattern [13,19]. This reflects how in-context learning operates: the model learns from examples by extracting patterns, and diverse examples expose more patterns.

Appropriate and Inappropriate Use Cases

Choosing when to use Opus 5.5 and when not to is an important part of mastering it. The model is optimized for specific kinds of work, and using it outside those bounds typically produces suboptimal results, higher costs, or both.

Opus 5.5 is most appropriate for:

- Complex coding tasks that require understanding large codebases, multi-file edits, or architectural reasoning [4,10]. The model's leading position on agentic coding benchmarks suggests it is well-suited for development workflows where reliability matters.

- Long-running agentic workflows that involve multiple tool calls, coordination across steps, and sustained task execution [1,4]. The model's design explicitly targets this use case, and features like preserved thinking support conversation continuity across extended sessions.
- Knowledge work that requires synthesizing large amounts of information, such as research synthesis, competitive analysis, or document review [1,4]. The one million token context window enables loading substantial material into a single request.
- Professional writing and editing tasks where tone consistency, structural quality, and adherence to specifications matter [20,21].
- Data analysis tasks that require structured reasoning, pattern recognition, and multi-step interpretation [22,23].

Opus 5.5 is less appropriate for:

- Simple, routine tasks where a cheaper model like Sonnet 5 or Haiku 4.5 would suffice. Using Opus 5.5 for straightforward questions or basic operations wastes resources and increases latency unnecessarily. Anthropic's model overview explicitly recommends starting with Opus 5.5 only for workloads that benefit from its capabilities [3].
- Tasks requiring deterministic output. If your application needs the same input to always produce the same output, language models are not the right tool. Opus 5.5 is no exception.
- Real-time user-facing features where latency is critical. Processing large contexts with Opus 5.5 takes time, and the thinking process adds additional latency. For interactive applications, consider using smaller models or caching strategies [24].
- Tasks that fall within refusal categories without appropriate safeguards. Biology and cybersecurity work may trigger refusals or model routing that disrupts workflows [1,8].
- Tasks that require guaranteed factual accuracy without external verification. Opus 5.5 is capable of producing highly accurate work, but it can also hallucinate. For high-stakes work, verification against trusted sources is essential regardless of the model.

A practical decision model: if your task is complex, benefits from sustained reasoning, involves large amounts of context, or requires coordination across multiple steps or tools, Opus 5.5 is likely the right choice. If your task is simple,

well-defined, and does not benefit from additional capability, use a cheaper model. The Anthropic documentation explicitly recommends measuring on your own workload using cost per completed task rather than cost per token when choosing models [25].

Separating Official Documentation from Speculation

The AI assistant ecosystem is filled with unverified claims about model behavior. It is essential to distinguish what Anthropic has officially documented from what community members have observed, hypothesized, or speculated.

Official documentation from Anthropic includes model overviews, API documentation, system cards, and published research [1-5,13-16]. These sources describe tested capabilities, confirmed limitations, and recommended practices. They are the most reliable basis for designing prompts and workflows.

Community observations come from practitioners who use the models in production. These can be valuable sources of practical insight, particularly for patterns that Anthropic has not explicitly documented. However, community claims should be treated as hypotheses rather than facts until independently verified. Different practitioners may have different experiences depending on their use cases, prompt designs, and evaluation methods.

Speculation and hype are everywhere. Claims about "secret" prompting techniques, "jailbreaks," or hidden capabilities are typically not verified and may reflect misunderstandings of how models work. This book treats such claims with skepticism and relies on documented or empirically verifiable patterns instead.

One area where confusion is common: the claim that XML tags are less necessary for modern Claude models. Anthropic's August 2026 blog post on prompt engineering best practices notes that "XML tags and heavy role prompting are less necessary for modern models" [19]. This does not mean XML tags are useless. It means that newer models follow instructions more reliably regardless of formatting, so the additional structure is not required for basic instruction following. For complex prompts with multiple components, XML tags still provide organizational clarity and help prevent the model from confusing instructions with examples or context. The distinction matters: XML tags are a structural aid, not a behavioral hack.

Another common misconception: that higher effort or more tokens always means better output. While increased effort generally improves performance on complex tasks, there are diminishing returns and trade-offs. Anthropic's optimization guidance explicitly recommends running an "effort sweep" on your workload to find the right balance rather than assuming maximum effort is optimal [25]. For some tasks, particularly simple or well-defined ones, lower effort produces acceptable results at significantly lower cost.

The safest approach when designing prompts and workflows is to start from official documentation, validate assumptions through testing on your actual tasks, and treat community claims as leads to investigate rather than facts to implement. This book follows that approach throughout.

Chapter 3: The Architecture of Effective Instructions

The Five Components of a Complete Instruction

Why Vague Instructions Produce Unreliable Results

The Principle of Minimum Sufficient Specification

Signal Versus Noise in Your Prompts

Reading Your Own Prompt as the Model Would

Chapter 4: Instruction Hierarchy and Design

When Instructions Conflict: Priority and Ordering Effects

Building Hierarchical Instruction Sets

Grouping Related Requirements

Handling Conditional and Contextual Instructions

Avoiding Over-Constraint and Prompt Bloat

Chapter 5: Context Engineering

The Relevance Principle: Why Less Context Is Often Better

Structuring Context for Maximum Utility

Positioning: Where Information Appears Matters

Compressing and Summarizing Context Without Losing Signal

When Long Context Is Necessary and How to Handle It

Common Context Engineering Mistakes

Chapter 6: Task Decomposition for Complex Work

Why Decomposition Works: Cognitive Load and Attention

Identifying Natural Break Points in Complex Tasks

Single-Pass Versus Multi-Step Strategies

Maintaining Coherence Across Steps

When Not to Decompose: Overengineering Simple Tasks

Error Propagation and Step Verification

Chapter 7: Constraints and Output Specifications

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/beyondpromptingmasteringclaudeopus55forcomplexwo

The Role of Constraints in Guiding Model Behavior

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/beyondpromptingmasteringclaudeopus55forcomplexwo

Writing Constraints That Are Actually Enforced

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/beyondpromptingmasteringclaudeopus55forcomplexwo

Positive Versus Negative Constraints

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/beyondpromptingmasteringclaudeopus55forcomplexwo

Format Specifications and Structural Requirements

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/beyondpromptingmasteringclaudeopus55forcomplexwo

Quality Criteria: Defining What Good Looks Like

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/beyondpromptingmasteringclaudeopus55forcomplexwo

When Constraints Hurt More Than They Help

Chapter 8: Examples, Demonstration, and Few-Shot Patterns

Why Examples Often Outperform Instructions Alone

Designing Effective Few-Shot Examples

One-Shot Patterns and Minimal Demonstration

When Examples Become Counterproductive

Balancing Examples With General Instructions

Edge Cases and Negative Examples

Chapter 9: Structured Outputs and Consistency

Why Structure Matters: From Human Reading to Machine Use

Schema Design for Complex Outputs

Delimiters, Markers, and Separation Strategies

Enforcing Consistency Across Multiple Outputs

Validation and Error Detection

Common Structured Output Failures and Fixes

Chapter 10: Managing Ambiguity and Edge Cases

Types of Ambiguity in Real-World Tasks

Preemptive Clarification: Asking the Right Questions First

Handling Uncertainty Without Inventing Answers

Designing for Edge Cases

Fallback Behaviors and Graceful Degradation

The Illusion of Clarity: Hidden Ambiguities in Your Prompts

Chapter 11: Iterative Refinement and Progressive Work

Why Iteration Is Not Failure

Effective Feedback: What to Say When Results Are Wrong

Progressive Elaboration Strategies

When to Iterate on the Prompt Versus the Task Design

Avoiding Degradation Across Iterations

Tracking Changes and Learning from Iteration History

Chapter 12: Self-Correction and Verification Workflows

When Self-Correction Works and When It Does Not

Designing Effective Self-Review Instructions

Checklists and Verification Criteria

Cross-Verification Strategies

External Verification: When the Model Cannot Trust Itself

The Cost of Self-Review: Efficiency Trade-Offs

Chapter 13: Anticipating Failure: Common Anti-Patterns

The Over-Specified Prompt: When Complexity Backfires

The Ambiguous Power User: False Precision

Conflicting Requirements and Priority Collisions

Assuming What You Did Not Specify

Diagnosing Failure: Prompt Problem or Model Problem or Task Problem

When a Sophisticated Prompt Performs Worse Than a Simple One

Chapter 14: Grounding Outputs and Reducing Fabrication

Understanding Model Fabrication: Causes and Patterns

Evidence-Constrained Responses

Citation Requirements and Traceability

Handling Knowledge Gaps Without Invention

Domain-Specific Fabrication Risks

Verification Against Provided Sources

Chapter 15: Efficiency: Doing More with Less

Token Costs and Their Practical Impact

Latency Considerations for Interactive Work

Achieving Quality With Minimal Context

When Extra Tokens Are Worth It

Compression Without Loss

Efficiency in Long-Running Workflows

Chapter 16: Reliability and Consistency in Professional Work

What Reliability Means in Practice

Testing and Validating Prompts Before Deployment

Maintaining Consistency Across Variants and Versions

Building Trust Through Predictability

When Reliability Requirements Exceed the Model

Operational Checklists for Professional Use

Chapter 17: Research and Knowledge Synthesis Workflows

Research Workflow Architecture

Providing and Managing Source Materials

Synthesis Versus Summarization

Maintaining Fidelity to Sources

Comparative Analysis and Cross-Source Reasoning

Verification and Citation in Research Outputs

Chapter 18: Software Development with Claude Opus 5.5

Coding Context: What to Provide and How

Code Generation With Reliable Structure

Code Review and Refinement Workflows

Debugging and Problem Diagnosis

Architecture Design and Technical Specification

Iterative Development Patterns

Chapter 19: Writing, Editing, and Content Workflows

Drafting Versus Editing: Different Modes, Different Prompts

Style, Tone, and Voice Specification

Iterative Editing Workflows

Structural Editing and Rewriting

Maintaining Factual Accuracy in Generated Text

Collaborative Authorship Patterns

Chapter 20: Data Analysis and Structured Reasoning

Providing Data in Analysis-Friendly Formats

Guiding Structured Reasoning Without Over-Directing

Calculation Verification and Numerical Accuracy

Scenario Analysis and Comparative Evaluation

Argument Construction and Logical Structure

Limits of Analytical Reasoning in Language Models

Chapter 21: Document Processing and Knowledge Management

Document Analysis and Extraction Patterns

Information Organization and Structuring

Multi-Document Synthesis

Building Structured Knowledge from Unstructured Sources

Long Document Strategies

Quality Control in Automated Processing

Chapter 22: Multi-Step Workflows and Automation Design

Workflow Design Fundamentals

State Management Across Steps

Context Carryover and Summarization Between Steps

Error Handling and Recovery Patterns

Human-in-the-Loop Decision Points

Building Repeatable Automation

Chapter 23: Tool Use and External Information Integration

Understanding Claude's Tool Use Capabilities

Designing Tool-Integrated Prompts

Orchestration: Model as Coordinator

Handling Tool Outputs and Errors

When Tools Are Necessary Versus When They Are Not

Verification of Tool-Assisted Results

Chapter 24: Designing Prompts for Unknown Tasks

Starting From Zero: Systematic Discovery

Exploratory Prompting: Testing Assumptions

Hypothesis-Driven Refinement

Building Patterns From Scratch

Documenting and Generalizing New Patterns

When to Abandon a Task Design

Chapter 25: Conclusion: Principles That Endure

The Principles That Matter

Adapting to Future Models

The Difference Between Prompting and Architecting

A Final Word on Judgment and Responsibility

References

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/beyondpromptingmasteringclaudeopus55forcomplexwo